

CONTENTS

Lecture 1

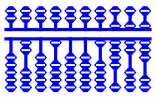
Introduction

Overview of Some User Subroutines	L1.2
Where User Subroutines Fit into ABAQUS/Standard	L1.6
User Subroutine Calls in the First Iteration	L1.10
Including User Subroutines in a Model	L1.11
Using Multiple User Subroutines in a Model	L1.12
Restart Analyses	L1.12
Writing Output from User Subroutines	L1.13
Path Names for External Files	L1.14
Compiling and Linking User Subroutines	L1.15
FORTRAN Compiler Levels	L1.17
Debugging Techniques and Proper Programming Habits	L1.18
Required FORTRAN Statements	L1.18
Naming Conventions	L1.20
Subroutine Argument Lists	L1.20
Solution-Dependent State Variables	L1.21
Testing Suggestions	L1.24

Lecture 2

User Subroutine: DLOAD

Introduction	L2.2
ABAQUS Usage	L2.3
DLOAD vs. UEL	L2.4
DLOAD Subroutine Interface	L2.5
Variables to be Defined	L2.6
Variables for Information Only	L2.6
Example: Transient Internal Pressure Loading	L2.8



Partial Input Data	L2.9
User Subroutine	L2.10
Remarks	L2.11
Example: Asymmetric Pressure Loads	L2.12
Partial Input Data	L2.14
User Subroutine	L2.15
Remarks	L2.16

Lecture 3

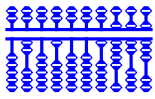
User Subroutine: FILM

Introduction	L3.2
ABAQUS Usage	L3.3
FILM Subroutine Interface	L3.4
Variables to be Defined	L3.5
Variables for Information Only	L3.6
Example: Radiation in Finned Surface	L3.8
Partial Input Data	L3.12
User Subroutine	L3.13
Remarks	L3.15
Workshop: User Subroutine FILM	L3.16
Goals	L3.16
Problem Description	L3.16

Lecture 4

User Subroutine: USDFLD

Introduction	L4.2
ABAQUS Usage	L4.3
Defining Field-Variable-Dependent Material Properties	L4.5
Defining Field Variables	L4.9
Accessing Solution Data at Material Points	L4.11
Explicit vs. Implicit Solution	L4.12

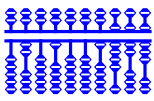


Using Solution-Dependent State Variables	L4.13
User Subroutine GETVRM	L4.15
GETVRM Subroutine Interface	L4.15
Variables Provided to GETVRM	L4.15
Variables Returned by GETVRM	L4.16
Elements Supported by GETVRM	L4.18
USDFLD Subroutine Interface	L4.19
Variables to be Defined	L4.20
Variables that may be Defined	L4.21
Variables for Information Only	L4.22
USDFLD and Automatic Time Incrementation	L4.24
Example: Laminated Composite Plate Failure.....	L4.27
Material Model	L4.29
Partial Input Data	L4.38
User Subroutine	L4.41
Results	L4.45
Remarks	L4.48

Lecture 5

User Subroutine: URDFIL

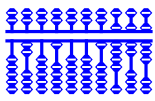
Introduction	L5.2
ABAQUS Usage.....	L5.4
Utility Routine POSFIL	L5.5
Utility Routine DBFILE	L5.7
URDFIL Subroutine Interface	L5.9
Variables to be Defined	L5.10
Variables for Information Only	L5.11
Example: Using URDFIL to Terminate an Analysis	L5.12
Input Data	L5.13
User Subroutine	L5.16
Remarks	L5.21



Lecture 6

Writing a UMAT or VUMAT

Overview	L6.2
Motivation	L6.4
Steps Required in Writing a UMAT or VUMAT	L6.12
UMAT Interface	L6.20
UMAT Variables	L6.25
UMAT Utilities	L6.28
UMAT Conventions	L6.29
UMAT Formulation Aspects	L6.30
Usage Hints	L6.32
Example 1: Isotropic Isothermal Elasticity	L6.33
Governing Equations	L6.33
Coding for Isotropic Isothermal Elasticity	L6.34
Remarks	L6.36
Example 2: Non-Isothermal Elasticity	L6.38
Governing Equations	L6.38
Coding for Non-Isothermal Elasticity	L6.39
Remarks	L6.43
Example 3: Neo-Hookean Hyperelasticity	L6.44
Governing Equations	L6.44
Coding for Neo-Hookean Hyperelasticity	L6.47
Remarks	L6.53
Example 4: Kinematic Hardening Plasticity	L6.54
Governing Equations	L6.54
Integration Procedure	L6.56
Coding for Kinematic Hardening Plasticity	L6.58
Remarks	L6.66
Example 5: Isotropic Hardening Plasticity	L6.69
Governing Equations	L6.69
Integration Procedure	L6.71
Coding for Isotropic Mises Plasticity	L6.73

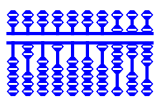


Remarks	L6.83
VUMAT Interface	L6.85
VUMAT Variables	L6.90
Comparison of VUMAT and UMAT Interfaces	L6.92
VUMAT Conventions	L6.94
VUMAT Formulation Aspects	L6.96
Example 6: VUMAT for Kinematic Hardening	L6.98
Coding for Kinematic Hardening Plasticity VUMAT	L6.99
Remarks	L6.105
Example 7: VUMAT for Isotropic Hardening	L6.107
Coding for Isotropic Hardening Plasticity VUMAT	L6.108
Remarks	L6.117

Lecture 7

Creating a Nonlinear User Element

Overview	L7.2
Motivation	L7.3
Defining a User Element	L7.8
Key Characteristics of a User Element	L7.8
Other Important Element Properties	L7.9
Defining the User Element Behavior	L7.10
UEL Interface	L7.13
ABAQUS Options	L7.13
Parameter Definition	L7.14
Data Lines	L7.15
More on Keywords and Parameters	L7.19
User Element Loads	L7.22
UEL Interface	L7.23
UEL Variables	L7.24
UEL Conventions	L7.27
UEL Formulation Aspects and Usage Hints	L7.28
Coding and Testing the UEL	L7.30

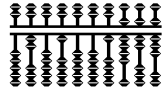


Example 1: Planar Beam Element with Nonlinear

Section Behavior	L7.32
Objective	L7.32
Coding Requirements	L7.33
Element Formulation	L7.35
Element Definition in the Input File	L7.39
Coding for Planar Beam Example	L7.40
Remarks	L7.45
Generalized Constitutive Behavior	L7.46
Remarks	L7.50
Example 2: Force Control Element	L7.51
Objective	L7.51
Element Formulation	L7.54
Element Definition in the Input File	L7.56
Coding for Force Control Element Example	L7.57
Remarks	L7.61
Using Nonlinear User Elements in Various Analysis Procedures	L7.63
Overview of Procedures	L7.63
Perturbation Procedures	L7.66
Transient Analysis	L7.69
Transient Heat Transfer Analysis	L7.70
Dynamic Analysis	L7.73

Workshops

Workshop Preliminaries	WP.1
User Subroutine FILM	W1.1
User Subroutine UMAT : Tangent Stiffness	W2.1



Lecture 1

Introduction

Overview

- Overview of Some User Subroutines
- Where User Subroutines Fit into ABAQUS/Standard
- Including User Subroutines in a Model
- Writing Output from User Subroutines
- Compiling and Linking User Subroutines
- Debugging Techniques and Proper Programming Habits

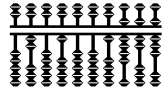
Overview of Some User Subroutines

- ABAQUS/Standard provides users with an extensive array of user subroutines that allow them to adapt ABAQUS to their particular analysis requirements.
- Chapter 24 of the ABAQUS/Standard Users' Manual details all 40+ user subroutines available in ABAQUS/Standard.

Some popular user subroutines include

CREEP: Use this subroutine to define time-dependent, viscoplastic deformation in a material. The deformation is divided into deviatoric behavior (creep) and volumetric behavior (swelling).

DLOAD: Use this subroutine to define nonuniform, distributed mechanical loads (pressures and body forces).

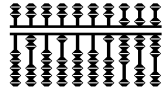


FILM: Use this subroutine to describe complex film coefficient behavior (temperature and field variable dependence) and complex sink temperature behavior.

FRIC: Use this subroutine when more complex models than those provided with the *FRICTION option are needed to describe the transmission of shear forces between surfaces. The models defined in this subroutine must be local models (information is provided only at the node making contact).

HETVAL: Use this subroutine to define complex models for internal heat generation in a material, such as might occur when the material undergoes a phase change.

UEL: Use this subroutine when it is necessary to create elements with an element formulation that is not available in ABAQUS/Standard.

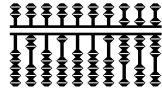


UEXPAN: Use this subroutine to define incremental thermal strains when the material's thermal expansion is too complex to model with the *EXPANSION option.

UEXTERNALDB: Use this subroutine to help manage external databases that may be used by other user subroutine or other software programs that are providing ABAQUS data and/or using data generated by ABAQUS.

UGENS: Use this subroutine to define complex, nonlinear mechanical behavior for shell elements directly in terms of the shell element's section stiffness.

UMAT: Use this subroutine to define any complex, constitutive models for materials that cannot be modeled with the available ABAQUS material models.



UPOREP: Use this subroutine to define the initial pore fluid pressures in a coupled pore fluid diffusion and stress analysis as a function of node location.

URDFIL: Use this subroutine to read the data from the results (`.fil`) file at the end of an increment. The information can be used to make decisions such as when to terminate the analysis or whether to overwrite the results of the previous increment.


USDFLD: Use this subroutine to define the values of field variables directly at the integration points of elements. The field variable values can be functions of element variables such as stress or strain.

UWAVE: Use this user subroutine to define complex wave kinematics in an ABAQUS/Aqua simulation or to determine when the configuration of the model should be updated in a stochastic wave analysis.

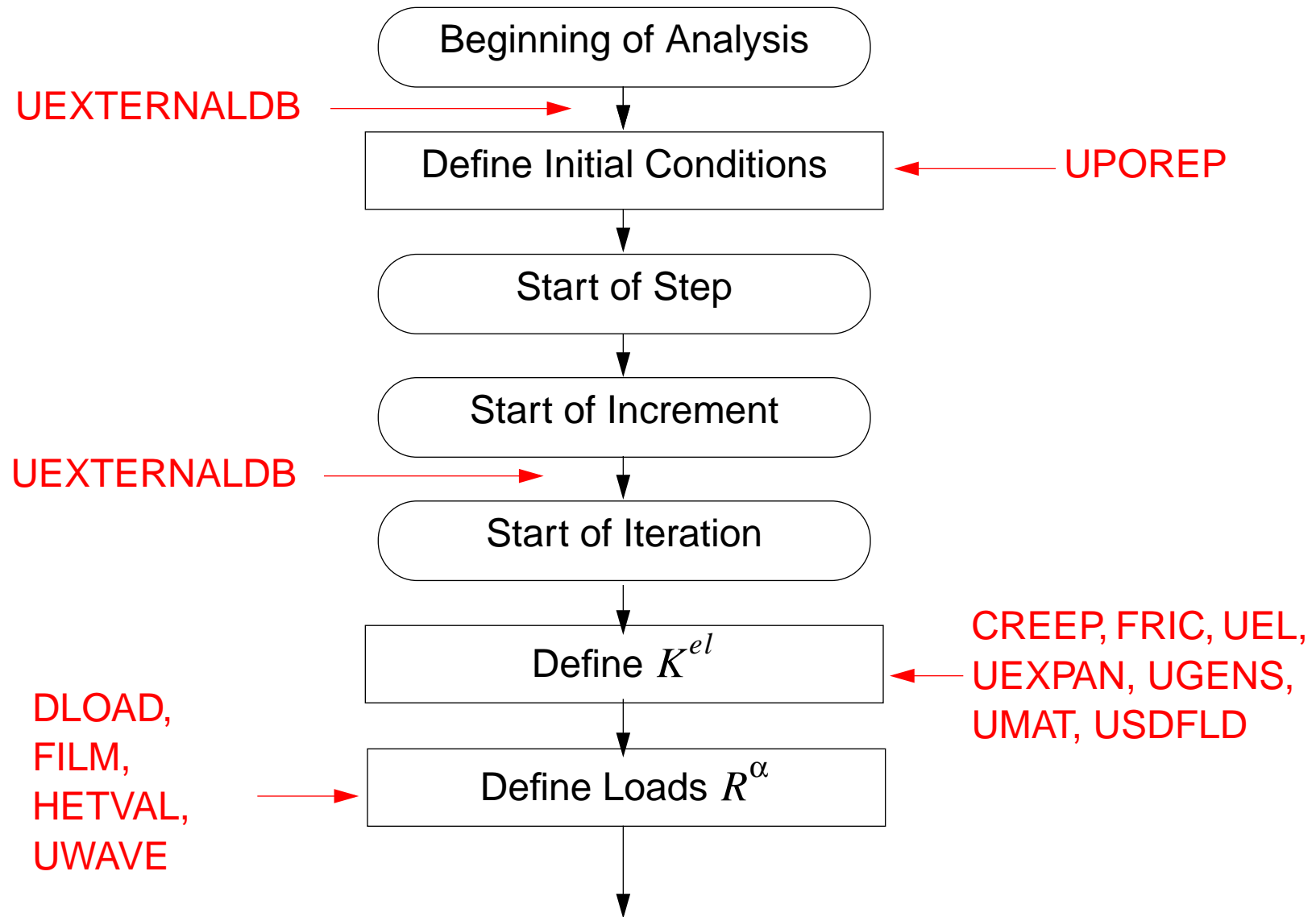
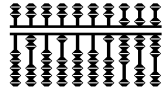
Where User Subroutines Fit into ABAQUS/Standard

While a complete understanding of the structure of ABAQUS is not required to develop a user subroutine, it helps if the developer has an understanding of at least the overall structure.

Figure 1–1 shows the basic flow of data and actions from the start of an ABAQUS/Standard analysis to the end of a step. Figure 1–2 shows a much more detailed accounting of how ABAQUS/Standard calculates the element stiffness during an iteration.

In the figures a  signifies a decision point in the code or a specific state (i.e., beginning of increment) during the analysis.

A  signifies an action that is taken during the analysis.



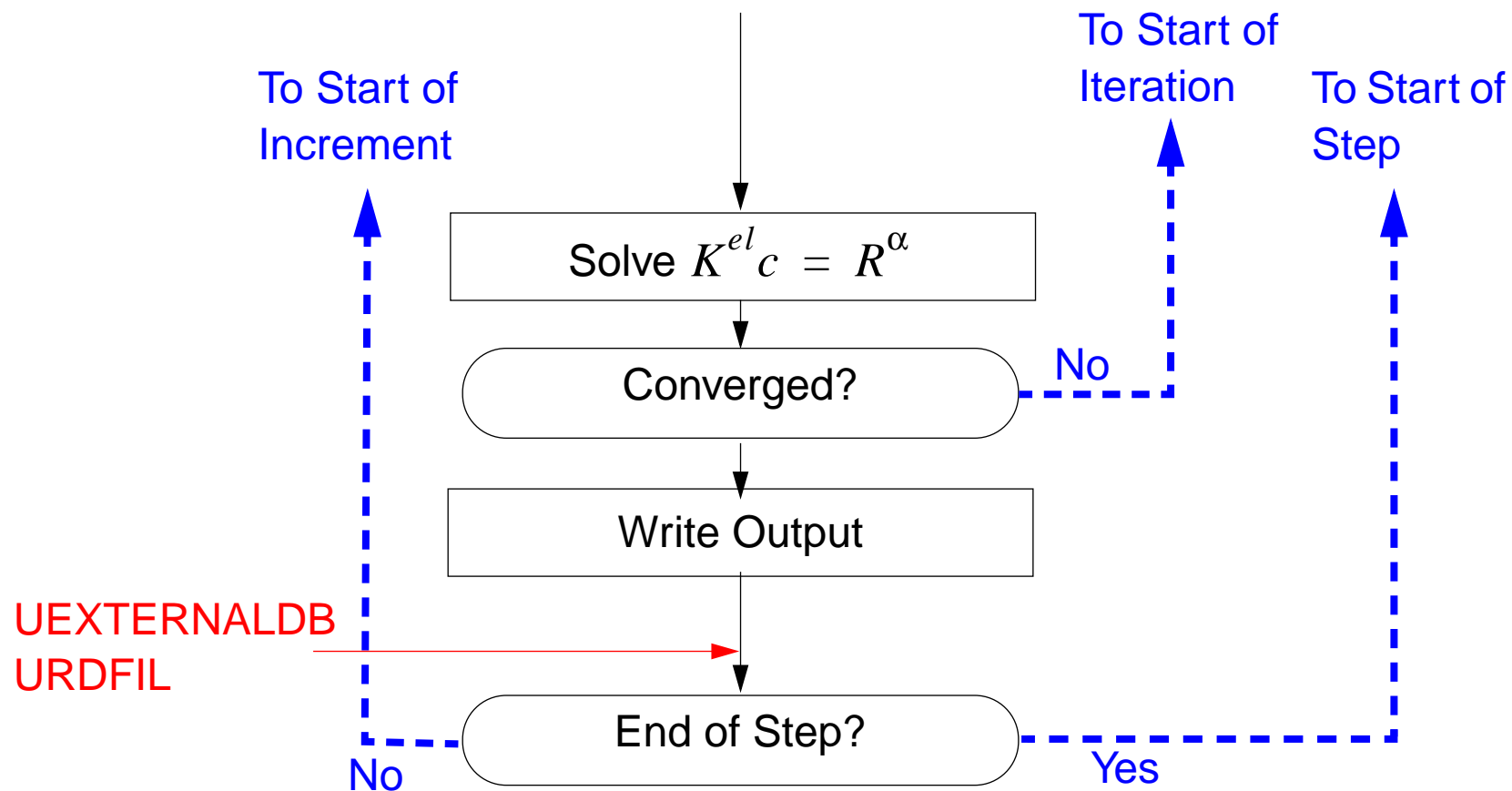
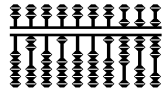


Figure 1–1. Global Flow in ABAQUS/Standard

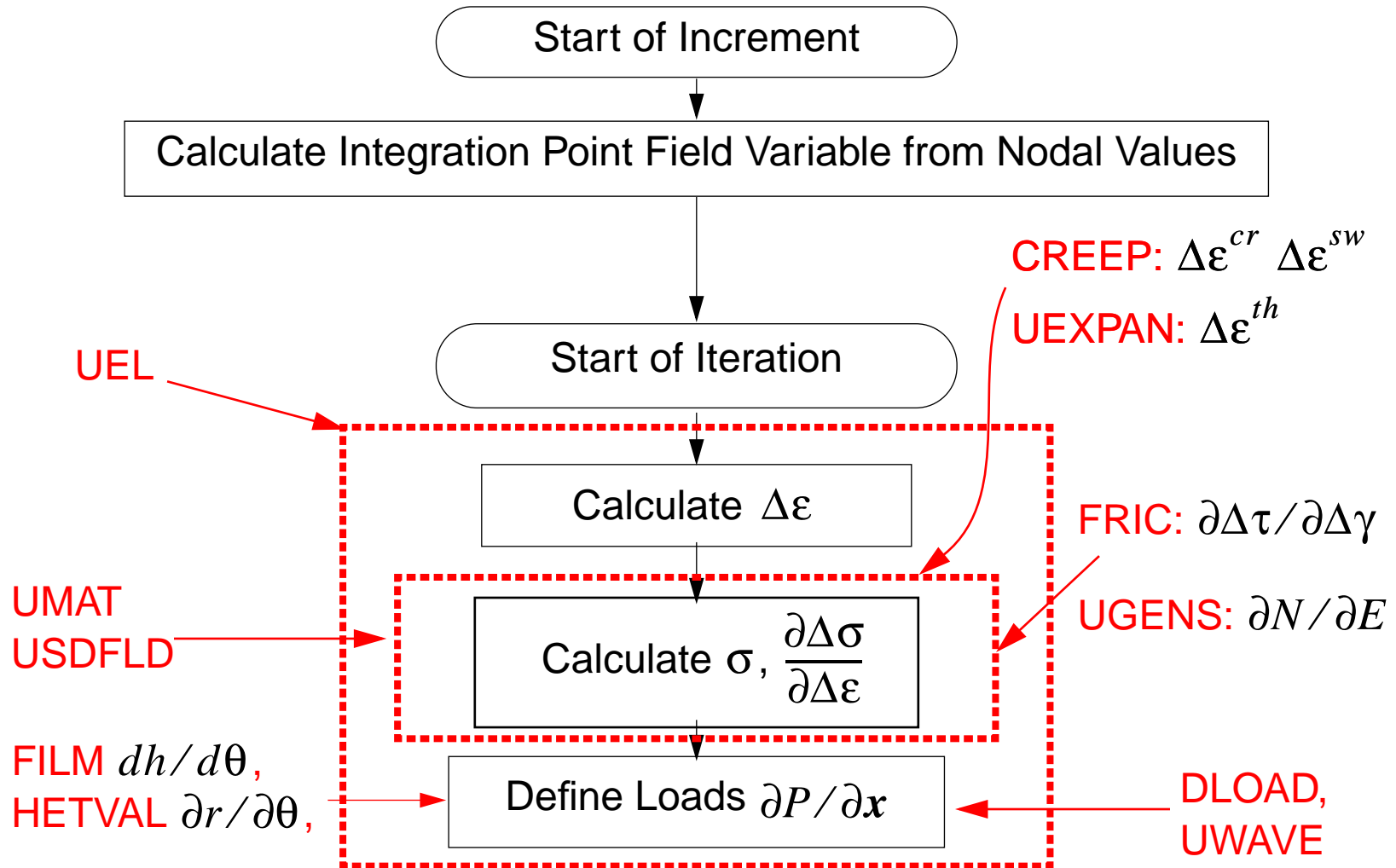
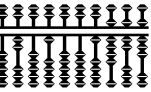


Figure 1–2. A More Detailed Flow of ABAQUS/Standard

User Subroutine Calls in the First Iteration

- The flow chart in Figure 1–2 is idealized. In the first iteration of an increment all of the user subroutines shown in the figure are called twice.
 - During the first call the initial stiffness matrix is being formed using the configuration of the model at the start of the increment.
 - During the second call a new stiffness, based on the updated configuration of the model, is created.
- In subsequent iterations the subroutines are called only once.
 - In these subsequent iterations the corrections to the model's configuration are calculated using the stiffness from the end of the previous iteration.

Including User Subroutines in a Model

- To include user subroutines in an analysis, specify the name of a file with the **user** parameter on the ABAQUS execution command.

abaqus job=my_analysis user=my_subroutine

The file should be either source code (**my_subroutine.f**) or an object file (**my_analysis.o**). The file extension can be included (**user=my_analysis.f**); otherwise, ABAQUS will determine automatically which type of file is specified.

Using Multiple User Subroutines in a Model

- When multiple user subroutines are needed in the analysis, the individual routines can be combined into a single file.
- A given user subroutine (such as **UMAT** or **FILM**) should appear only once in the specified user subroutine source or object code.

Restart Analyses

- When an analysis that includes a user subroutine is restarted, the user subroutine must be specified again because the subroutine object or source code is not stored on the restart (**.res**) file.

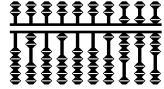
Writing Output from User Subroutines

- The following unit numbers can be used within a user subroutine to read and write data from files:

15–18

100+

- In ABAQUS/Standard user subroutines can write debug output to the message (**.msg**) file (unit 7) or to the printed output (**.dat**) file (unit 6).
 - These units do not have to be opened within the user subroutine—they are opened by ABAQUS.
 - These unit numbers cannot be used by user subroutines in ABAQUS/Explicit.



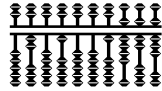
Path Names for External Files

- When a file is opened in a user subroutine, ABAQUS assumes that it is located in the scratch directory created for the simulation; therefore, full path names must be used in the OPEN statements in the subroutine to specify the location of the files.

Compiling and Linking User Subroutines

- When a model that contains user subroutines is submitted to ABAQUS, the correct compile and link commands should be used automatically.
 - HKS includes the correct compile and link commands for every platform on which ABAQUS is supported in the default environment file (**abaqus.env**) located in the *abaqus_dir/site* directory.

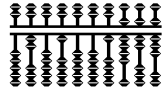
abaqus_dir is the directory in which ABAQUS was installed.



- For example on the Windows NT release, the following commands are defined in the *abaqus_dir\site\abaqus.env* file:

```
compile="fl32 /c"  
link="link /defaultlib:libf.lib libc.lib user32.lib  
netapi32.lib advapi32.lib mpr.lib libelm.lib  
/subsystem:console /out:"
```

- If you encounter compile or link errors, check that the *abaqus_dir\site\abaqus.env* file defines the compile and link commands. If it does not, please contact HKS. We will provide you with the correct commands for your system.



FORTRAN Compiler Levels

- The FORTRAN compiler levels used to create ABAQUS are shown at **`www.abaqus.com/support`** on the World Wide Web.
 - Digital Visual Fortran and Microsoft Visual C++ must be installed to run user subroutines on computers running Windows NT 4.0.
- If the version of your FORTRAN compiler does not correspond to that specified on the HKS web site, incompatibilities may occur.

Debugging Techniques and Proper Programming Habits

Some programming habits and suggested debugging techniques are discussed in this section.

Required FORTRAN Statements

- Every user subroutine in ABAQUS/Standard must include the statement

```
INCLUDE 'ABA_PARAM.INC'
```

as the first statement after the argument list.

- The file **ABA_PARAM.INC** is installed on the computer system by the ABAQUS installation procedure.
 - The file specifies either **IMPLICIT REAL*8 (A-H, O-Z)** for double precision machines or **IMPLICIT REAL (A-H, O-Z)** for single precision machines.
 - The ABAQUS execution procedure, which compiles and links the user subroutine with the rest of ABAQUS, will include the **ABA_PARAM.INC** file automatically.
 - It is not necessary to find this file and copy it to any particular directory: ABAQUS will know where to find it.
- Every user subroutine in ABAQUS/Explicit must include the statement
include 'vaba_param.inc'

Naming Conventions

- If user subroutines call other subroutines or use COMMON blocks to pass information, such subroutines or COMMON blocks should begin with the letter K since this letter is never used to start the name of any subroutine or COMMON block in ABAQUS.

Subroutine Argument Lists

- The variables passed into a user subroutine via the argument list are classified as either variables to be defined, variables that can be defined, or variables passed in for information.
- The user must not alter the values of the “variables passed in for information.”
 - Doing so will have unpredictable results.

Solution-Dependent State Variables

- Solution-dependent state variables (SDVs) are values that can be defined to evolve with the solution of an analysis.
 - An example of a solution-dependent state variable for the **UEL** subroutine is strain.
- Several user subroutines allow the user to define SDVs. Within these user subroutines the SDVs can be defined as functions of any variables passed into the user subroutine.
 - It is the user's responsibility to calculate the evolution of the SDVs within the subroutine; ABAQUS just stores the variables for the user subroutine.

- Space must be allocated to store each of the solution-dependent state variables defined in a user subroutine.
 - For most subroutines the number of such variables required at the integration points or nodes is entered as the only value on the data line of the *DEPVAR option.

***USER MATERIAL**

***DEPVAR**

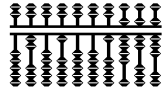
8

- For subroutines **UEL** and **UGENS** the **VARIABLES** parameter must be used on the ***USER ELEMENT** and ***SHELL GENERAL SECTION** options, respectively.

***USER ELEMENT, VARIABLES=8**

- For subroutine **FRIC** the number of variables is defined with the **DEPVAR** parameter on the ***FRICTION** option.

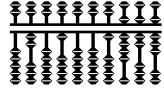
***FRICTION, USER, DEPVAR=8**



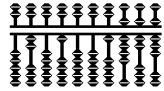
- There are two methods available for defining the initial values of solution-dependent variables.
 - The `*INITIAL CONDITIONS, TYPE=SOLUTION` option can be used to define the variable field in a tabular format
 - For complicated cases user subroutine **SDVINI** can be used to define the initial values of the SDVs. Invoke this subroutine by adding the `USER` parameter to the `*INITIAL CONDITIONS, TYPE=SOLUTION` option.

Testing Suggestions

- Always develop and test user subroutines on the smallest possible model.
- Do not include other complicated features, such as contact, unless they are absolutely necessary when testing the subroutine.
- Test the most basic model of the user subroutine before adding additional complexity to the subroutine.
 - Whenever a “new” feature is added to the model in a user subroutine, test it before adding an additional feature.
- When appropriate, try to test the user subroutine in models where only values of the nodal degrees of freedom (displacement, rotations, temperature) are specified. Then test the subroutine in models where fluxes and nodal degrees of freedom are specified.



- Ensure that arrays passed into a user subroutine with a given dimension are not used as if they had a larger dimension.
 - For example, if a user subroutine is written such that the number of SDVs is 10 but only 8 SDVs are specified on the *DEPVAR option, the user subroutine will overwrite data stored by ABAQUS; the consequences of this accident will be unpredictable.

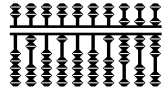


Lecture 2

User Subroutine: DLOAD

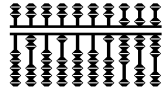
Overview

- Introduction
- ABAQUS Usage
- DLOAD Subroutine Interface
- Example: Transient Internal Pressure Loading
- Example: Asymmetric Pressure Loads



Introduction

- User subroutine **DLOAD** is typically used when a load is a complex function of time and/or position.
 - Loads that are simple functions of time can usually be modeled with the *AMPLITUDE option.
 - The subroutine can also be used to define a load that varies with element number and/or integration point number.



ABAQUS Usage

- The subroutine is called when the ***DLOAD** or ***DSLOAD** options contain a nonuniform load type label.

– For example,

***DLOAD**

ELTOP, P1NU, 10.0

Load type label

Load magnitude

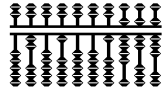
specifies that the elements in element set **ELTOP** will be subject to a force per area on the 1-face of solid (continuum) elements or a force per unit length in the beam local 1-direction when used with beam elements.

The magnitude specified on the data line is passed into the subroutine as the value of variable **F**.

- A list of the nonuniform distributed load types that are available for use with any particular element is given in the ABAQUS/Standard Users' Manual.
- The AMPLITUDE parameter cannot be used with the *DLOAD or *DSLOAD options when the user subroutine is used to define the magnitude of the distributed load.
- The distributed load magnitude cannot be written as output with the *EL FILE or *EL PRINT options.

DLOAD vs. UEL

- If the distributed load is dependent on the element's deformation rather than the position of the element, a stiffness is being defined and a user element subroutine (**UEL**), not user subroutine **DLOAD**, is required.



DLOAD Subroutine Interface

The interface to user subroutine DLOAD is

```
SUBROUTINE DLOAD(F, KSTEP, KINC, TIME, NOEL, NPT,  
1 LAYER, KSPT, COORDS, JLTYP, SNAME)
```

C

```
INCLUDE 'ABA_PARAM.INC'
```

C

```
DIMENSION TIME(2), COORDS(3)  
CHARACTER*80 SNAME
```

```
user coding to define F
```

```
RETURN
```

```
END
```

Variables to be Defined

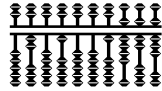
The user has to define only the variable **F**. It is the magnitude of the distributed load and has units that depend on the type of distributed load applied:

- FL^{-1} for line loads along one-dimensional (beam) elements,
- FL^{-2} for surface loads (e.g., pressures), and
- FL^{-3} for body forces (e.g., gravity, centripetal, acceleration).

Variables for Information Only

The following variables are passed into the subroutine:

- The step (**KSTEP**) and increment number (**KINC**) in which the routine is being called.
- The current value of the step (**TIME (1)**) and the total time (**TIME (2)**).



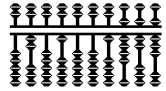
- The element number (**NOEL**) and integration point number (**NPT**).
- The layer (**LAYER**) and section point numbers (**KSPT**), where appropriate.
- The coordinates of the integration point (**COORDS**).
 - These are the current coordinates if the *STEP, NLGEOM option is used in this or a previous step.
- The identifier (**JLTYP**) specifying the type of load to be defined in this call to the user subroutine.
 - If multiple user-defined **DLOAD** types are specified in a model, the coding for all load types must appear in the subroutine, and this variable (**JLTYP**) must be used to test for which load type is to be defined when the subroutine is called.
- The surface name (**SNAME**) for a surface-base load definition (**JLTYP=0**).

Example: Transient Internal Pressure Loading

- The problem models the viscoelastic response in rocket propellant as the transient pressure load, due to rocket ignition, is applied to the inner diameter of the rocket motor.
 - The transient pressure load is an exponential function of time.

$$p = 10(1 - e^{-23.03t}) \text{ MPa.}$$

- The rocket motor is modeled with a single row of 21 elements.
- The model is fully described in Problem 2.2.7 of the ABAQUS Verification Manual.



Partial Input Data

***HEADING**

:

:

***BOUNDARY**

ALL, 2

***STEP, INC=50**

***VISCO, CETOL=7.E-3**

0.01, 0.5

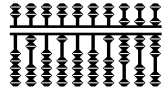
***DLOAD**

1, P4NU ← Apply nonuniform DLOAD to face 4 of element 1

:

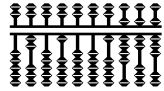
:

***END STEP**



User Subroutine

```
      SUBROUTINE DLOAD(F, KSTEP, KINC, TIME ,NOEL, NPT,  
1  LAYER, KSPT, COORDS, JLTYP, SNAME)  
  
C  
C      EXPONENTIAL PRESSURE LOAD  
C  
C      INCLUDE 'ABA_PARAM.INC'  
C  
  
      DIMENSION COORDS(3),TIME(2)  
      CHARACTER*80 SNAME  
      DATA TEN,ONE,CONST /10.,1.,-23.03/  
      F=TEN*(ONE-(EXP(CONST*TIME(1))))  
      IF(NPT.EQ.1) WRITE(6,*) ' LOAD APPLIED',F,' AT  
          TIME=',TIME(1)  
  
      RETURN  
      END
```



Remarks

- The load in this model is defined as a function of step time, **time(1)**.
- The load is applied only to the one element on the inner diameter of the rocket motor, element 1.
- The load is monitored by writing output to the printed output (**.dat**) file, once per iteration, when the distributed load value is defined at the first integration point.

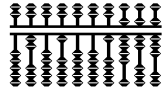
Example: Asymmetric Pressure Loads

- In this problem asymmetric pressure loads are applied to a cylindrical structure, which is modeled with the CAXA family of elements.
 - These elements have axisymmetric geometry but asymmetric deformation.
 - For CAXA elements the third coordinate, `COORD (3)`, of a point is its θ position around the circumference of the structure ($0 \leq \theta \leq 180$).
 - The radial stress distribution at the outer radius, R_o , is

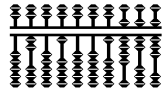
$$\sigma_{rr} = -p \cos \theta,$$

and at inner radius, R_i , it is

$$\sigma_{rr} = -\frac{R_o}{R_i} p \cos \theta.$$



- The magnitude of p is 10.E3.
- This model is described fully in Problem 1.3.31 of the ABAQUS Verification Manual.



Partial Input Data

*HEADING

ASYMMETRIC INTERNAL AND EXTERNAL PRESSURE LOADS

** 6-IN LONG CYLINDER OF 2-IN & 6-IN INNER AND OUTER RADII

:

*ELSET, ELSET=INWALL

1

R_i

R_o

*ELSET, ELSET=OUTWALL

10

:

*STEP

*STATIC

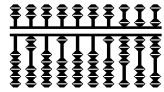
:

*DLOAD

INWALL, P4NU

OUTWALL, P2NU

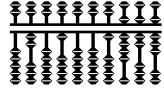
*END STEP



User Subroutine

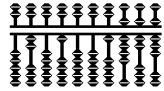
```
SUBROUTINE DLOAD(F, KSTEP, KINC, TIME, NOEL, NPT,  
1  LAYER, KSPT, COORDS, JLTYP, SNAME)  
  INCLUDE 'ABA_PARAM.INC'  
  DIMENSION COORDS(3)  
  CHARACTER*80 SNAME  
  C NOTE THAT COORDS(3) IS THE ANGULAR COORD IN DEGREES  
  PI=2.* ASIN(1.D0)  
  THETA=PI*COORDS(3)/180.D0  
  P=0.  
  IF(JLTYP.EQ.22) P=10.D3  ← Test for load type P4NU  
  IF(JLTYP.EQ.24) P=30.D3  
  F=P* COS(THETA)  
  RETURN  
  END
```

→ Test for load type P2NU



Remarks

- In this model subroutine **DLOAD** is used to define two different nonuniform distributed pressure loads.
 - In this model the load type label clearly determined which pressure distribution should be used.
 - In a different model perhaps the element number (**NOEL**) or radial position (**COORDS (1)**) would have to be used to identify which distribution to define.
 - What are some of the methods that could be used to allow the subroutine to define pressure distributions with different values of p on the inner and outer radii?

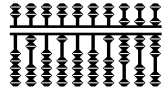


Lecture 3

User Subroutine: FILM

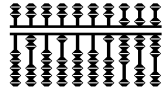
Overview

- Introduction
- ABAQUS Usage
- FILM Subroutine Interface
- Example: Radiation in Finned Surface
- Workshop: User Subroutine FILM



Introduction

- User subroutine **FILM** is typically used when either the film coefficient, h , or sink temperature, θ^s , is a complex function of time, position, and/or surface temperature.
 - h and θ^s that are simple functions of time usually can be modeled with the *AMPLITUDE option.
 - The subroutine can also be used to define a load that varies with element number and/or integration point number.



ABAQUS Usage

- The subroutine is called when the *FILM or *SFILM options contain a nonuniform load type label or when the USER parameter is used with the *CFILM option.

- For example,

***FILM**

ELLEFT, F6NU, 10.0, 1500

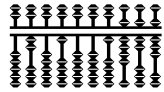
Load type label

θ^s

h

specifies that the elements in element set **ELLEFT** will be subject to a film load (convection boundary condition) on face six (6) of solid (continuum), heat transfer elements.

- The variable **H(1)** will be passed into the routine with the value of h specified on the data line of the *FILM option.
- The variable **SINK** will be passed into the routine with the value of θ^s specified on the data line of the *FILM option.



FILM Subroutine Interface

The interface to user subroutine **FILM** is:

```
SUBROUTINE FILM(H, SINK, TEMP, KSTEP, KINC, TIME,  
1 NOEL, NPT, COORDS, JLTYP, FIELD, NFIELD, SNAME,  
2 NODE, AREA)
```

C

```
INCLUDE 'ABA_PARAM.INC'
```

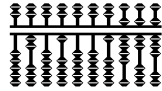
C

```
DIMENSION H(2), TIME(2), COORDS(3), FIELD(NFIELD)  
CHARACTER*80 SNAME
```

user coding to define H(1), H(2), and SINK

```
RETURN
```

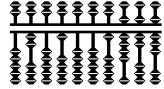
```
END
```



Variables to be Defined

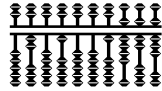
The user has to define the following variables:

- **H (1)** . The film coefficient, h , at this surface point.
Its units are $J T^{-1} L^{-2} \theta^{-1}$.
- **H (2)** . The rate of change of the film coefficient with respect to the surface temperature at this point ($\frac{dh}{d\theta}$). Its units are $J T^{-1} L^{-2} \theta^{-2}$.
 - The rate of convergence during the solution of the nonlinear equations in an increment is improved by defining this value, especially when the film coefficient is a strong function of surface temperature (**TEMP**).
- **SINK**, the sink temperature, θ^s .



Variables for Information Only

- The estimated surface temperature at this time at this point (**TEMP**).
- The step (**KSTEP**) and increment number (**KINC**) in which the routine is being called.
- The current value of the step (**TIME (1)**) and the total time (**TIME (2)**).
- The element number (**NOEL**) and integration point number (**NPT**).
- The coordinates of the integration point (**COORDS**).
 - These are the current coordinates if the *STEP, NLGEOM option is used in this or a previous step.



- The identifier (**JLTYP**) specifying the type of load to be defined in this call to the user subroutine.
 - If multiple user-defined **FILM** types are specified in a model, the coding for all film types must appear in the subroutine, and this variable (**JLTYP**) must be used to test for which film type is to be defined when the subroutine is called.
- The interpolated values of field variables, f_i , at this point (**FIELD**).
 - Any of the variables, h or θ^s , can be made functions of f_i .
- The number of field variables (**NFIELD**).
- The surface name (**SNAME**) for a surface-based film definition (**JLTYP**=0).
- The node number (**NODE**) and area (**AREA**) for a node-based film definition. The area is passed in as the value specified on the data line of the *CFILM option.

Example: Radiation in Finned Surface

- This problem is described in detail in Section 4.1.4 of the ABAQUS Example Problems Manual.
- This problem models the heat transfer through a finned structure.
 - In Step 1 the steady-state conditions are obtained.
 - In Step 2 the transient response of a 30-minute fire is obtained. The finned surface is exposed to the fire.
 - In Step 3 the 60-minute transient response of the structure when the steady-state boundary conditions are returned is obtained.

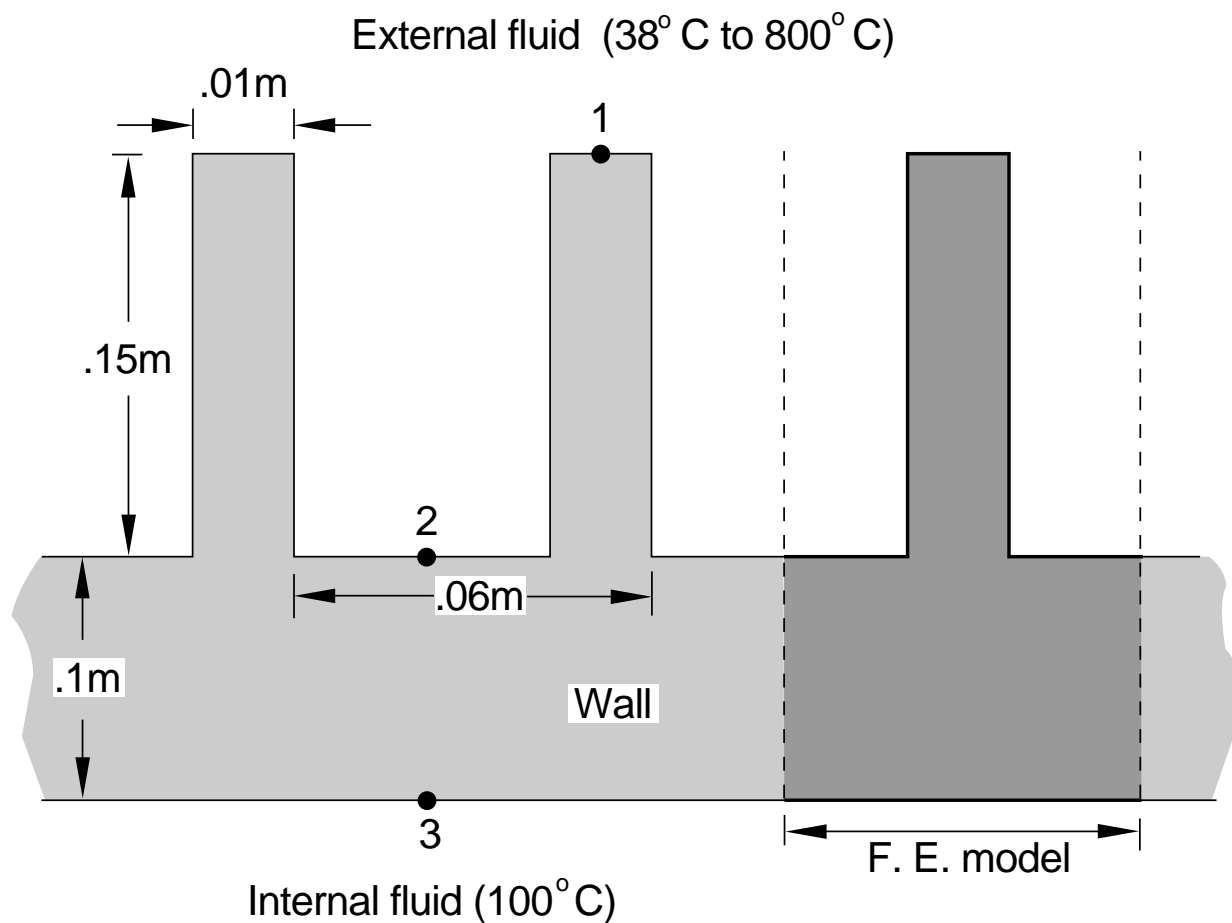
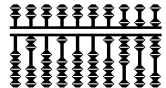


Figure 3–1. Sketch of Finned Structure

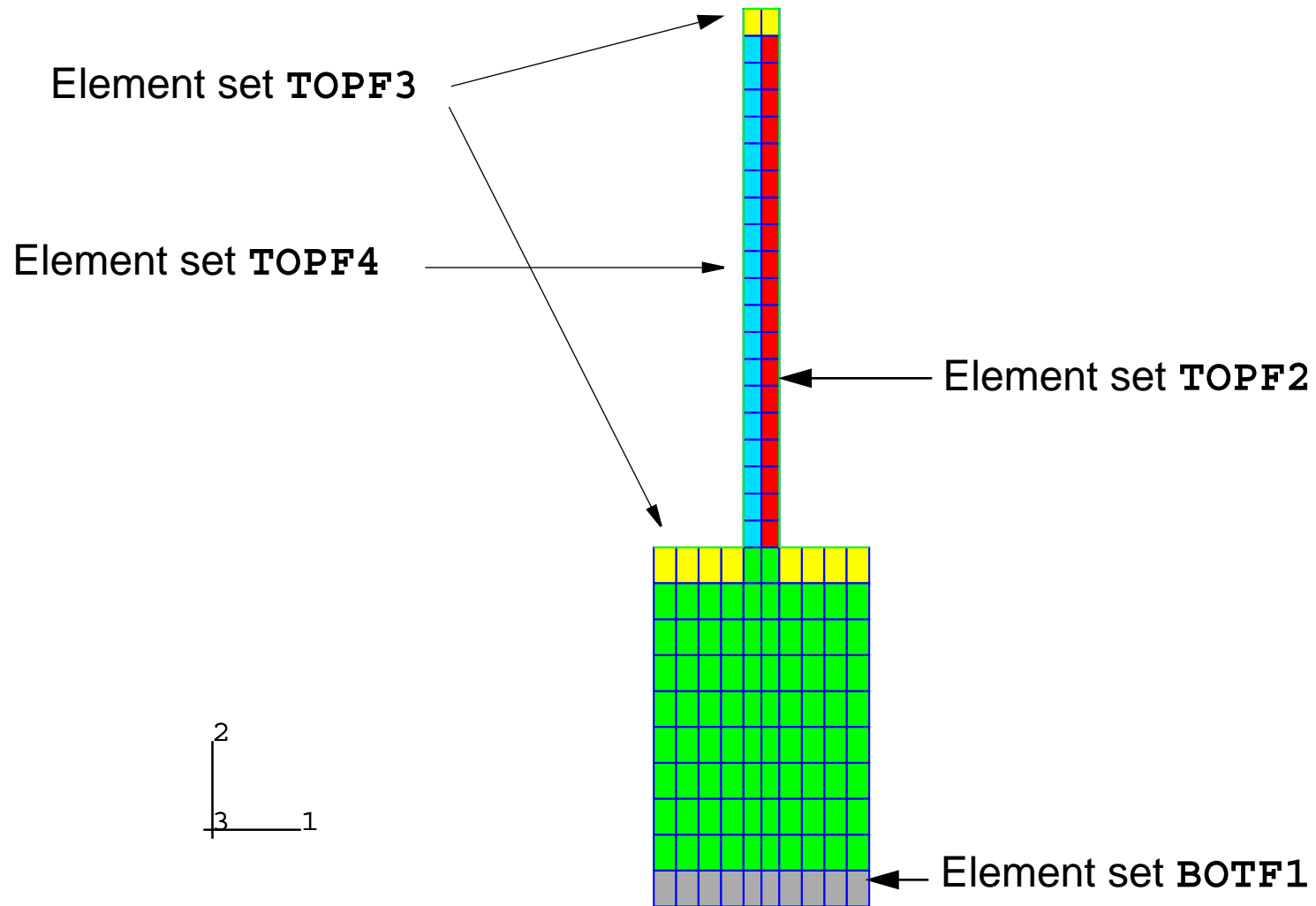
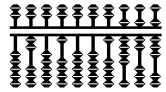
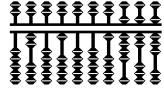


Figure 3–2. Finite Element Mesh and Element Sets That Use **FILM**



- Along the inner wall (element set **BOTF1**) natural convection transfers heat between the structure and surrounding fluid, which is at 100°C.

$$h(\theta) = 500 \left| \theta_w - \theta_i^s \right|^{1/3}$$

θ_w is the temperature of the wall of the structure.

θ_i^s is the internal fluid temperature.

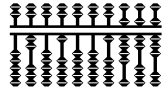
- Along the outer, finned surface, radiation and natural convection boundary conditions exist.

During Steps 1 and 3,

$$h(\theta) = 2 \left| \theta_w - \theta_f^s \right|^{1/3}.$$

θ_f^s is the external fluid temperature (38°C).

During Step 2 (the fire transient), $h = 10$.



Partial Input Data

***HEADING**

:

***STEP, INC=500**

***HEAT TRANSFER, STEADY STATE**

1.0

***BOUNDARY**

NAMB, 11, , 38.D0

***FILM**

BOTF1, F1NU

***FILM**

TOPF3, F3NU

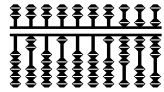
TOPF4, F4NU

TOPF2, F2NU

:

***END STEP**

User subroutine **FILM** used for all these element sets



User Subroutine

```
subroutine film(h, sink, temp, jstep, jinc, time,  
1 noel, npt, coords, jltyp, field, nfield, sname,  
node, area)
```

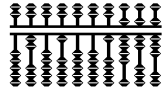
c

```
include 'aba_param.inc'  
dimension h(2), coords(3), time(2), field(nfield)  
character*80 sname  
parameter (two=2.0d0, third=1.0d0/3.0d0)
```

c

```
h(1) = 0.0d0  
h(2) = 0.0d0  
sink = 0.0d0  
a2    = 1.0d0  
if (noel.le.11) then  
    sink = 100.d0  
    a1 = sign(a2, temp-sink)  
    h(1) = 500.0d0*(abs(temp-sink)**third
```

Test to see if the elements are in
element set BOTF1



```

      h(2) = a1*third*500.0d0*
1      (abs(temp-sink))**(-two*third)
      else if (jstep.eq.1.or.jstep.eq.3) then
        sink = 38.d0
        a1 = sign(a2,temp-sink)
        h(1) = 2.0d0*(abs(temp-sink))**third
        h(2) = a1*third*2.0d0*
1      (abs(temp-sink))**(-two*third)
      else
        sink = 800.d0
        h(1) = 10.0d0
      end if

```

c

```

return
end

```

$h(\theta)$ and fluid temperature (θ_f^s) for the fire event

If the first condition is not satisfied, the element must be on the finned surface. The $h(\theta)$ for these elements varies from step to step.

Remarks

- It can be helpful to define constants, in this example 2.0 and 0.333 as parameters in user subroutines.
- Using element numbers as values for conditional statements can limit a user subroutine to a specific mesh layout.
 - This programming/design technique can make the use of the subroutine in general production work tedious.
 - Can you think of other techniques that can be used with this user subroutine?
- Defining $\frac{dh}{d\theta}$ correctly is extremely important in this application.
 - Analyses will converge very slowly if it is incorrect.

Workshop: User Subroutine FILM

Goals

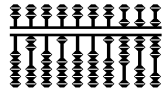
- To learn how to find the compile and link commands used on your system.
- To see how sensitive the rate of convergence is on the value of $\frac{dh}{d\theta}$.
- To see if the results are sensitive to the value of $\frac{dh}{d\theta}$.

Problem Description

- User subroutine **FILM** will be used to define

$$h(\theta) = 500 \left| \theta_w - \theta_i^s \right|^{1/3},$$

where $\theta_i^s = 100^\circ\text{C}$ is the fluid temperature.



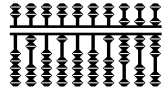
- The subroutine is tested on a two-element model.
 - All nodes initially have a temperature of 77°C ($\theta(t = 0) = 77$).
 - The nodes on one end are set to 277°C . The other end has the film condition applied.
 - The steady-state solution is obtained.
- The rate of convergence and results are compared with the following values of $\frac{dh}{d\theta}$:

```
h(2) = a1*third*500.d0*(abs(temp-sink))**(-two*third)
```

```
h(2) = third*500.d0*(abs(temp-sink))**(-two*third)
```

```
h(2) = a1*third*500.0d0*(abs(temp-sink))**(-third)
```

```
h(2) = a1*third*500.0d0*(abs(temp-sink))**(two*third)
```

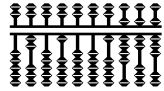



Lecture 4

User Subroutine: USDFLD

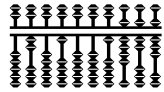
Overview

- Introduction
- ABAQUS Usage
- User Subroutine GETVRM
- USDFLD Subroutine Interface
- Example: Laminated Composite Plate Failure



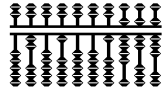
Introduction

- User subroutine **USDFLD** is typically used when complex material behavior needs to be modeled and the user does not want to develop a **UMAT** subroutine.
 - Most material properties in ABAQUS/Standard can be defined as functions of field variables, f_i .
 - Subroutine **USDFLD** allows the user to define f_i at every integration point of an element.
 - The subroutine has access to solution data, so $f_i(\sigma, \epsilon, \epsilon^{pl}, \dot{\epsilon}, \text{etc.})$; therefore, the material properties can be a function of the solution data.
- Subroutine **USDFLD** can be used only with elements that have material behavior defined with a ***MATERIAL** option [see **Elements Supported by GETVRM** (p. L4.18) for details].

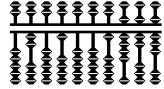


ABAQUS Usage

- Including user subroutine **USDFLD** in a model requires considerably more effort than what is needed for user subroutines **DLOAD** or **FILM**.
- Typically the user must define the dependence of material properties, such as elastic modulus or yield stress, as functions of field variables, f_i .
 - This can be accomplished using either tabular input or additional user subroutines.



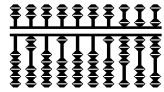
- The **USDFLD** routine is then written to define the values of f_i on an integration point by integration point basis.
 - The ***USER DEFINED FIELD** option is included in the material definition to indicate that the **USDFLD** subroutine will be called for those elements using that material definition.
 - The f_i can be defined as functions of solution data, such as stress or strain, available at the integration points.



Defining Field-Variable-Dependent Material Properties

There are two methods that can be used to create field-variable-dependent material properties:

- Using tabular definition for built-in ABAQUS material models.
- Using other user subroutines, such as **CREEP**, to define the material behavior as a function of f_i .



Tabular Definition:

- Use the DEPENDENCIES parameter on the material options to specify how many different field variables exist for a given material option:

```
*MATERIAL, NAME=POLYMER
```

```
*ELASTIC, DEPENDENCIES=1
```

```
2000., 0.3, 0., 0.00
```

```
1200., 0.3, 0., 0.02
```

```
1000., 0.3, 0., 0.04
```

```
*EXPANSION, DEPENDENCIES=2
```

```
5E-4, 0., 0.00, 0.0
```

```
3E-4, 0., 0.02, 0.0
```

```
1E-4, 0., 0.04, 0.0
```

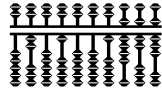
```
**
```

```
5E-5, 0., 0.00, 1.0
```

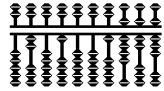
```
2E-5, 0., 0.03, 1.0
```

```
8E-6, 0., 0.04, 1.0
```

 f_1 f_2



- The elastic modulus (E) is a function of field variable #1, f_1 . As f_1 increases, E decreases— f_1 might represent damage to the material.
- The thermal expansion coefficient, α , is a function of both f_1 and field variable #2, f_2 .
- A change in the value of f_1 will affect both E and α .
- ABAQUS will use linear interpolation between data points in the tabular input and will use the last available material data if f_i is outside of the range specified—it does not extrapolate the data provided.
- The range of f_i does not have to be the same for each material property.



Defining Field Variable Dependence within a User Subroutine:

- The values of f_i defined in **USDFLD** are passed into the following user subroutines:

CREEP

HETVAL

UEXPAN

UHARD

UHYPEL

UMAT

UMATHT

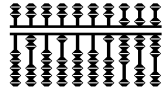
UTRS

UINTER

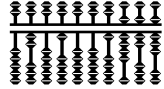
- The material properties defined in these subroutines can be made functions of the f_i .

Defining Field Variables

- Field variables (f_i) are normally considered nodal data by ABAQUS.
- When ABAQUS begins to calculate the element stresses and stiffness (i.e., the element loop), it interpolates the nodal values of f_i to the integration (material) points of the elements.
- When subroutine **USDFLD** is used, these interpolated f_i are replaced with the values defined in the **USDFLD** subroutine before the material properties of an element are calculated.
- The values defined by **USDFLD** are not stored by ABAQUS.
 - If you need access to previous values of f_i , you must save them as solution-dependent variables (SDVs) inside **USDFLD**.

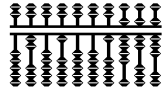


- If you bypass the **USDFLD** subroutine (perhaps because the material properties are not going to change in a given step), the integration points will use the interpolated values of f_i .
 - Typically these interpolated f_i are the initial values assigned to the nodes—the default in ABAQUS is to assign a value of 0.0 if no initial value is given explicitly.
 - It is quite possible that using these interpolated values when defining the material behavior will create incorrect results. Make sure you understand what ABAQUS is doing.
- The values of f_i at the element integration points can be written as output to the printed output (**.dat**), results (**.fil**), and output database (**.odb**) files using the output variable FV on the ***EL PRINT**, ***EL FILE**, and ***ELEMENT OUTPUT** options, respectively.
 - ABAQUS/Viewer can make contour plots of FV#.



Accessing Solution Data at Material Points

- ABAQUS/Standard allows the f_i to be defined as functions of solution data, such as stress or strain, at the material points.
- The values of the solution data provided are from the beginning of the current increment.
- Subroutine **USDFLD** must use the ABAQUS utility routine **GETVRM** to access this material point data.



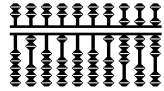
Explicit vs. Implicit Solution

- Since the **USDFLD** subroutine has access to material point quantities only at the start of the increment, the solution dependence introduced in this way is explicit.
 - The material properties for a given increment are not influenced by the results obtained during the increment.
 - Hence, the accuracy of the results depends on the size of the time increment.
 - Therefore, the user can control the time increment in the **USDFLD** subroutine by means of the variable **PNEWDT**.

- For most nonlinear material behavior (i.e., plasticity) ABAQUS/Standard uses an implicit integration method to calculate the material behavior at the end of the current increment.
 - Such an implicit integration method allows ABAQUS/Standard to use any size time increment and yet still have the solution remain bounded.

Using Solution-Dependent State Variables

- Solution-dependent state variables (SDVs) must be used in **USDFLD** if f_i have any history dependence.
 - ABAQUS/Standard does not store the values of f_i calculated in **USDFLD**.
- The SDVs updated in **USDFLD** are then passed into other user subroutines that can be called at this material point, such as those listed



in Defining Field-Variable-Dependent Material Properties (p. L4.5).

- The number of state variables is specified with the *DEPVAR option:

```
*ELASTIC, DEPENDENCIES=1
** Table of modulus values decreasing as a function
** of field variable 1.
2000., 0.3, 0., 0.00
1500., 0.3, 0., 0.01
1200., 0.3, 0., 0.02
1000., 0.3, 0., 0.04
*USER DEFINED FIELD
*DEPVAR
1
```

User Subroutine GETVRM

The subroutine **GETVRM** provides **USDFLD** with access to the solution data stored in databases during the analysis.

GETVRM Subroutine Interface

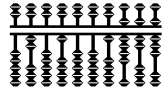
```
CALL GETVRM('VAR', ARRAY, JARRAY, FLGRAY, JRCD,  
1 JMAC, JMATYP, MATLAYO, LACCFLA)
```

Variables Provided to GETVRM

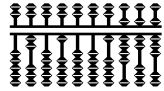
- The variables provided to **GETVRM** are the output variable key, **VAR**, for the desired solution data, and **JMAC**, **JMATYP**, **MATLAYO**, **LACCFLA** (these variables are not discussed further in these notes)
- The available output variable keys are listed in the output table in Section 4.2.1 of the ABAQUS/Standard User's Manual.
 - The variable must be available for results file output at the element integration points; e.g., S for stress.

Variables Returned by GETVRM

- An array containing individual floating-point components of the output variable (**ARRAY**).
- An array containing individual integer value components of the output variable (**JARRAY**).
- A character array (**FLGRAY**) containing flags corresponding to the individual components.
 - Flags will contain either **YES**, **NO**, or **N/A** (not applicable).
- A return code (**JRCD**). **JRCD=0** indicates that **GETVRM** encountered no errors, while a value of 1 indicates that there was an output request error or that all components of the output variable requested are zero.



- The components for a requested variable are written as follows.
 - Single index components (and requests without components) are returned in positions 1, 2, 3, etc.
 - Double index components (tensors) are returned in the order 11, 22, 33, 12, 13, 23 for symmetric tensors, followed by 21, 31, 32 for unsymmetric tensors, such as the deformation gradient.
 - Thus, the stresses for a plane stress element are returned as $\text{ARRAY}(1) = S11$, $\text{ARRAY}(2) = S22$, $\text{ARRAY}(3) = 0.0$, and $\text{ARRAY}(4) = S12$.
 - Three values are always returned for principal value requests, the minimum value first and maximum value third, regardless of the dimensionality of the analysis.



Elements Supported by GETVRM

- Since the **GETVRM** capability pertains to material point quantities, it cannot be used for most of the element types that do not require a ***MATERIAL** definition.
- The following element types are, therefore, not supported:

DASHPOT_x

SPRING_x

JOINTC

JOINT_xD

DRAG_xD

ITS_{xxx}

MASS

ROTARYI

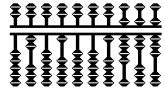
all acoustic elements

all contact elements

all gasket elements

all hydrostatic fluid elements

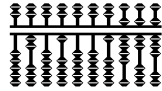
USA elements



USDFLD Subroutine Interface

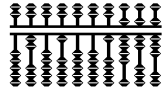
The interface to user subroutine USDFLD is:

```
      SUBROUTINE USDFLD(FIELD, STATEV, PNEWDT, DIRECT, T,  
1      CELENT, TIME, DTIME, CMNAME, ORNAME, NFIELD,  
2      NSTATV, NOEL, NPT, LAYER, KSPT, KSTEP, KINC, NDI,  
3      NSHR, COORD, JMAC, JMATYP, MATLAYO, LACCFLA)  
  
C  
      INCLUDE 'ABA_PARAM.INC'  
  
C  
      CHARACTER*80 CMNAME,ORNAME  
      CHARACTER*8  FLGRAY(15)  
      DIMENSION FIELD(NFIELD), STATEV(NSTATV), DIRECT(3, 3),  
1 T(3, 3), TIME(2), COORD(*), JMAC(*), JMATYP(*)  
      DIMENSION ARRAY(15), JARRAY(15)  
  
      user coding to define FIELD and,  
      if necessary, STATEV and PNEWDT
```



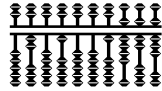
Variables to be Defined

- The array **FIELD** (**NFIELD**) contains the field variables (f_i) at the current material (integration) point.
 - These are passed in with the values interpolated from the nodes at the end of the current increment, as specified with the ***INITIAL CONDITIONS** option or the ***FIELD** option.
 - The updated f_i are used to calculate the values of material properties that are a function of field variables. The updated f_i are passed into other user subroutines (**CREEP**, **HETVAL**, **UEXPAN**, **UHARD**, **UHYPEL**, **UMAT**, **UMATHT**, and **UTRS**) that are called at this material point.



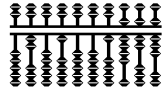
Variables that may be Defined

- The array containing the solution-dependent state variables, **STATEV** (**NSTATV**) , can be defined in **USDFLD**.
 - These are passed in as the values at the beginning of the increment.
 - In all cases **STATEV** can be updated in this subroutine, and the updated values are passed into other user subroutines (**CREEP**, **HETVAL**, **UEXPAN**, **UMAT**, **UMATHT**, and **UTRS**) that are called at this material point.
 - The number of state variables associated with this material point is defined with the ***DEPVAR** option.
- The ratio, **PNEWDT**, of suggested new time increment to the time increment being used (**DTIME**, see below) can be given.
 - This variable allows the user to provide input to the automatic time incrementation algorithms in ABAQUS.

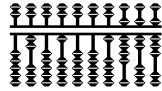


Variables for Information Only

- The number of field variables (**NFIELD**) that exist at this point.
- The direction cosines in the global coordinate system of the material directions associated with the current integration point (**DIRECT**).
 - **DIRECT**(#, 1) defines the first material direction.
- The direction cosines (**T**) of any rotations of the shell or membrane material direction about the element's normal.
- The characteristic element length in the model (**CELENT**).
- The name (**CNAME**) of the associated ***MATERIAL** option.
- The name (**ORNAME**) of the ***ORIENTATION** associated with this element.
- The number of direct stress components (**NDI**) and the number of shear stress components (**NSHR**).

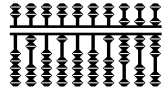


- The step (**KSTEP**) and increment number (**KINC**) in which the routine is being called
- The value of step time (**TIME (1)**) and the value of total time (**TIME (2)**) at the end of the current increment.
- The current time increment (**DTIME**).
- The element number (**NOEL**) and integration point number (**NPT**).
- The layer (**LAYER**) and section point numbers (**KSPT**), where appropriate.
- The coordinates (**COORD**) at the material point.
- Variables that must be passed into the GETVRM utility routine (**JMAC**, **JMATYP**, **MATLAYO**, and **LACCFLA**).



USDFLD and Automatic Time Incrementation

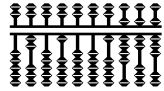
- ABAQUS/Standard uses an automatic time incrementation algorithm to control the size of the time increment used in an analysis.
 - This algorithm allows ABAQUS/Standard to reduce the time increment size when convergence is unlikely or the results are not accurate enough and to increase the time increment when convergence is easily obtained.
- Subroutines such as **USDFLD** can make it impossible for this algorithm to function properly.
 - Therefore, these subroutines are given the variable **PNEWDT** to provide information to the incrementation algorithm.



- **PNEWDT** is set to a large value before each call to **USDFLD**.
- If **PNEWDT** is redefined to be less than 1.0, ABAQUS must abandon the time increment and attempt it again with a smaller time increment.
 - The suggested new time increment provided to the automatic time integration algorithms is

PNEWDT*DTIME,

where the **PNEWDT** used is the minimum value for all calls to user subroutines that allow redefinition of **PNEWDT** for this iteration.



- If **PNEWDT** is given a value that is greater than 1.0 for all calls to user subroutines for this iteration and the increment converges in this iteration, ABAQUS may increase the time increment.
 - The suggested new time increment provided to the automatic time integration algorithms is
$$\mathbf{PNEWDT * DTIME},$$
 - where the **PNEWDT** used is the minimum value for all calls to user subroutines for this iteration.
- If the time increment size should be maintained, set **PNEWDT** = 1.0.
- If automatic time incrementation is not selected for the analysis procedure, values of **PNEWDT** that are greater than 1.0 will be ignored and values of **PNEWDT** that are less than 1.0 will cause the job to terminate.

Example: Laminated Composite Plate Failure

- This problem is described in detail in the ABAQUS Example Problems Manual, Section 1.1.14.
- This problem models the damage that occurs in a laminated composite plate with a hole in the center as it is subjected to in-plane compression.
 - The plate consists of graphite-epoxy plies with fiber directions that are in a $(-45/45)$ layup.
- A quarter-symmetry finite element model (shown in Figure 4–1) was used in this problem.
 - Rather than model the composite plate with shell elements, two layers of CPS4 elements were used—the thickness of the plate is large enough that out-of-plane displacements should be minimal.

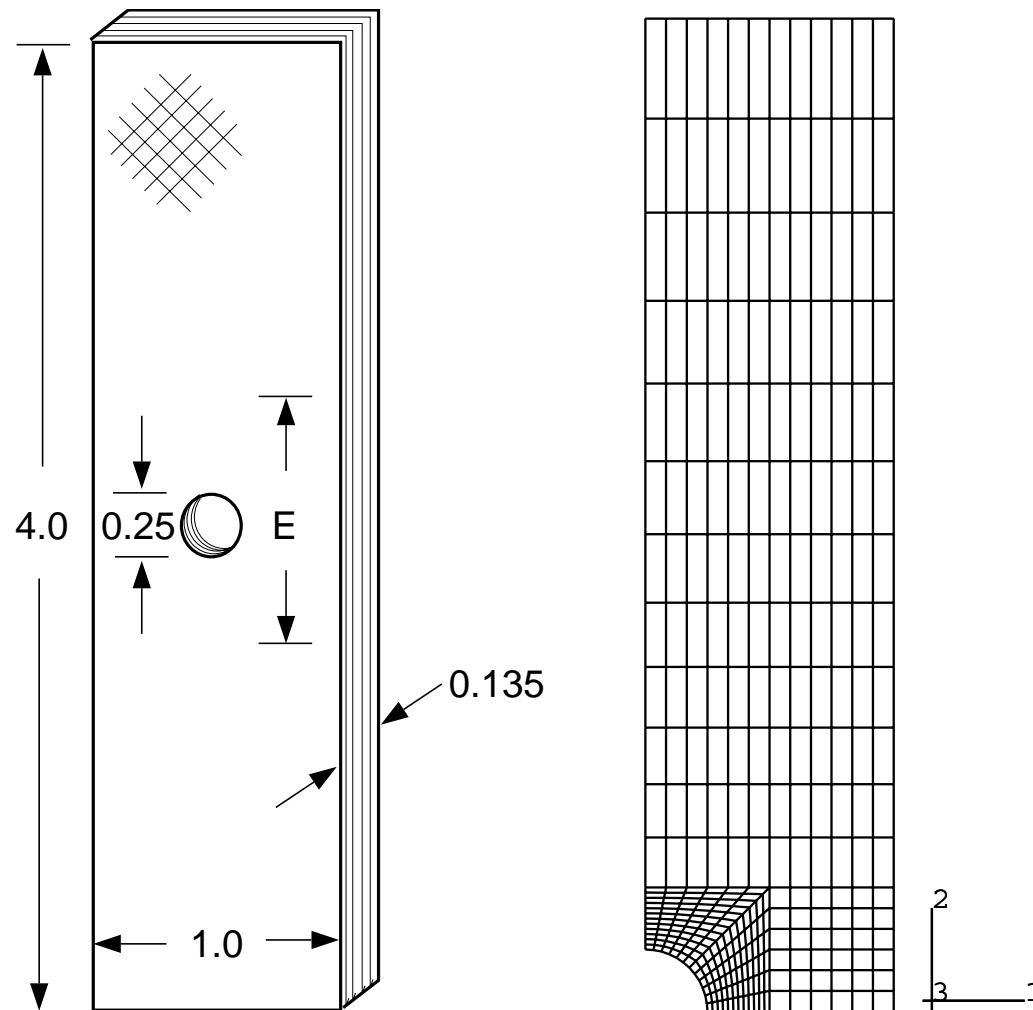
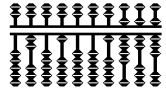
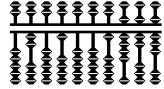


Figure 4–1. Problem Geometry and Finite Element Mesh



Material Model

- The material behavior of each ply is described in detail by Chang and Lessard.

Chang, F-K., and L. B. Lessard, “Damage Tolerance of Laminated Composites Containing an Open Hole and Subjected to Compressive Loadings: Part I—Analysis,” *Journal of Composite Materials*, vol. 25, pp. 2–43, 1991.

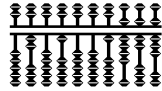
- The initial elastic ply properties are:

longitudinal modulus $E_{11} = 22700$ ksi,

transverse modulus $E_{22} = 1880$ ksi,

shear modulus $G = 1010$ ksi, and

Poisson's ratio $\nu = 0.23$.



- The material accumulates damage in shear, leading to a nonlinear stress-strain relation of the form

$$\gamma_{12} = G_{12}^{-1} \sigma_{12} + \alpha \sigma_{12}^3, \quad (\text{Eq. 4.1})$$

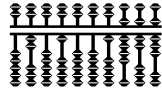
where G_{12} is the (initial) ply shear modulus and the nonlinearity is characterized by the factor $\alpha = 0.8 \times 10^{-14}$.

- To account for the nonlinearity, the nonlinear stress-strain relation (Equation 4.1) must be expressed in a different form:

The stress at the end of the increment must be given as a linear function of the strain.

- The most obvious way to do this is to linearize the nonlinear term, leading to the relation

$$\gamma_{12}^{(i+1)} = (G_{12}^{-1} + \alpha (\sigma_{12}^{(i)})^2) \sigma_{12}^{(i+1)}. \quad (\text{Eq. 4.2})$$



- Inverting the relation given in Equation 4.2 gives

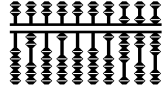
$$\sigma_{12}^{(i+1)} = \frac{G_{12}}{1 + \alpha G_{12} (\alpha_{12}^{(i)})^2} \gamma_{12}^{(i+1)}, \quad (\text{Eq. 4.3})$$

which provides an algorithm to define the effective shear modulus.

- However, this algorithm is not very suitable because it is unstable at higher strain levels, which is readily demonstrated by stability analysis (see the Example Problems Manual for details).
- To obtain a more stable algorithm, we write the nonlinear stress-strain law in the form

$$\gamma_{12} + \beta \sigma_{12}^3 = G_{12}^{-1} \sigma_{12} + (\alpha + \beta) \sigma_{12}^3, \quad (\text{Eq. 4.4})$$

where β is an as yet unknown coefficient.



- Using a stability analysis of the expression in Equation 4.4, the optimal stress-strain algorithm is found to be

$$\sigma_{12}^{(i+1)} = \frac{1 + (2\alpha(\sigma_{12}^{(i)})^3)/\gamma_{12}^{(i)}}{1 + 3\alpha G_{12}(\alpha_{12}^{(i)})^2} G_{12} \gamma_{12}^{(i+1)}. \quad (\text{Eq. 4.5})$$

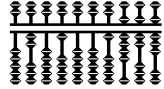
Writing this expression in terms of a damage parameter, d , gives

$$\sigma_{12}^{(i+1)} = (1 - d) G_{12} \gamma_{12}^{(i+1)},$$

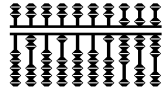
where

$$d = \frac{3\alpha G_{12}(\alpha_{12}^{(i)})^2 - 2\alpha(\sigma_{12}^{(i)})^3/\gamma_{12}^{(i)}}{1 + 3\alpha G_{12}(\alpha_{12}^{(i)})^2}. \quad (\text{Eq. 4.6})$$

- This relation is implemented in user subroutine **USDFLD**, and the value of the damage parameter is assigned directly to the third field variable (FV3) used for definition of the elastic properties.



- The following strength properties are used for the composite material:
 - transverse tensile strength $Y_t = 14.82$ ksi,
 - ply shear strength $S_c = 15.5$ ksi,
 - matrix compressive strength $Y_c = 36.7$ ksi, and
 - fiber buckling strength $X_c = 392.7$ ksi.



- The strength parameters can be combined into failure criteria for multiaxial loading. Three different failure modes are considered in the model analyzed.
 - Fiber buckling failure is not considered in this model because the primary mode of failure is fiber-matrix shear.

Matrix Tensile Cracking

The failure index for matrix tensile cracking with nonlinear shear behavior is

$$e_m^2 = \left(\frac{\sigma_{22}}{Y_t} \right)^2 + \frac{2\sigma_{12}^2 / G_{12} + 3\alpha\sigma_{12}^4}{2S_c^2 / G_{12} + 3\alpha S_c^4}.$$

When the composite fails in this mode, transverse stiffness (E_{22}) and Poisson's ratio become zero.

Matrix Compressive Cracking

The form of the failure index is identical to that for the tensile cracking mode. The same failure index (field variable) is used in **USDFLD** because these two modes will not occur simultaneously at the same point.

Fiber-Matrix Shearing Failure

The failure criterion has essentially the same form as the other two criteria:

$$e_{fs}^2 = \left(\frac{\sigma_{11}}{X_c} \right)^2 + \frac{2\sigma_{12}^2 / G_{12} + 3\alpha\sigma_{12}^4}{2S_c^2 / G_{12} + 3\alpha S_c^4}.$$

This failure mechanism can occur simultaneously with the other two criteria; therefore, a separate failure index is used in **USDFLD**.

- User subroutine **USDFLD** is used to calculate the values of the active failure indices:

The value for the matrix tensile cracking/matrix compressive failure index (e_m) is stored as SDV(1).

- When the index exceeds a value of 1.0, the value of f_1 is set to 1.0.

The value for the fiber-matrix shear failure index (e_{fs}) is stored as SDV(2).

- When the index exceeds a value of 1.0, the value of f_2 is set to 1.0.

- Subroutine **USDFLD** is also used to calculate the value of the damage parameter, d , in the nonlinear stress-strain relationship.
 - The value of d is stored as SDV(3) and f_3 .
- Table 4–1 shows the dependence of material properties on the f_i .

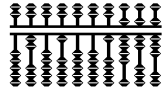
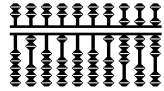


Table 4–1. Dependence of Elastic Material Properties on f_i

Material State	Elastic Properties				f_1	f_2	f_3
No failure or damage	E_{11}	E_{22}	ν_{12}	G_{12}	0	0	0
Matrix failure	E_{11}	0	0	G_{12}	1	0	0
Fiber-matrix failure	E_{11}	E_{22}	0	0	0	1	0
Shear damage	E_{11}	E_{22}	ν_{12}	0	0	0	1
Matrix & fiber-matrix	E_{11}	0	0	0	1	1	0
Matrix & damage	E_{11}	0	0	0	1	0	1
Fiber-matrix & damage	E_{11}	E_{22}	0	0	0	1	1
All failure modes	E_{11}	0	0	0	1	1	1



Partial Input Data

*HEADING

:

**NONLINEAR SHEAR WITH BUILT-IN EXPLICIT FAILURE

**

** FV1: MATRIX COMPRESSIVE/TENSILE FAILURE

** FV2: FIBER-MATRIX SHEAR FAILURE

** FV3: SHEAR NONLINEARITY (DAMAGE) PRIOR TO FAILURE

** TOTAL OF $2^3 = 8$ STATES

**

*MATERIAL, NAME=T300

*ELASTIC, TYPE=LAMINA, DEPENDENCIES=3

22.7E6, 1.88E6, 0.23, 1.01E6, 1.01E6, 1.01E6, 0., 0,
0, 0

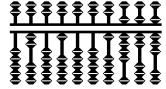
22.7E6, (1.00E0, 0.00), 1.01E6, 1.01E6, 1.01E6, 0., (1,
0, 0

22.7E6, 1.88E6, (0.00, 1.00E0), 1.01E6, 1.01E6, 0., 0,

(1, 0

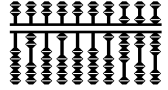
$e_m = 1.0$

$e_{fs} = 1.0$

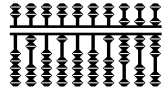


```
22.7E6, 1.00E0, 0.00, 1.00E0, 1.01E6, 1.01E6, 0., 1,
1, 0
22.7E6, 1.88E6, 0.23, 1.00E0, 1.01E6, 1.01E6, 0., 0,
0, 1
22.7E6, 1.00E0, 0.00, 1.00E0, 1.01E6, 1.01E6, 0., 1,
0, 1
22.7E6, 1.88E6, 0.00, 1.00E0, 1.01E6, 1.01E6, 0., 0,
1, 1
22.7E6, 1.00E0, 0.00, 1.00E0, 1.01E6, 1.01E6, 0., 1,
1, 1
*DEPVAR
3
*USER DEFINED FIELD
**
```

Shear damage



```
** ANALYSIS HISTORY
*STEP, INC=200, NLGEOM
*STATIC, DIRECT
0.05, 1.0
*BOUNDARY
XSYMMTRY, XSYMM
YSYMMTRY, YSYMM
1000, 2, , -0.027
:
*END STEP
```

User Subroutine

```

SUBROUTINE USDFLD (FIELD, STATEV, PNEWDT, DIRECT, T,
1  CELENT, TIME, DTIME, CMNAME, ORNAME, NFIELD, NSTATV,
2  NOEL, NPT, LAYER, KSPT, KSTEP, KINC, NDI, NSHR,
3  COORD, JMAC, JMATYP, MATLAYO, LACCFLA)

C

  INCLUDE 'ABA_PARAM.INC' C MATERIAL AND STRENGTH PARAMETERS

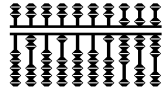
  PARAMETER (YT=14.86D3, XC=392.7D3, YC=36.7D3)
  PARAMETER (SC=15.5D3, G12=1.01D6, ALPHA=0.8D-14)

C

  CHARACTER*80 CMNAME, ORNAME
  CHARACTER*8  FLGRAY(15)
  DIMENSION FIELD(NFIELD), STATEV(NSTATV), DIRECT(3,3)
  DIMENSION T(3, 3), TIME(2), ARRAY(15), JARRAY(15)
  DIMENSION COORD(*), JMAC(*), JMATYP(*)

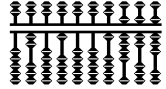
C
C INITIALIZE FAILURE FLAGS FROM STATEV.
  EM      = STATEV(1)
  EFS     = STATEV(2)
  DAMAGE  = STATEV(3)

```



```
C GET STRESSES FROM PREVIOUS INCREMENT
  CALL GETVRM('S', ARRAY, JARRAY, FLGRAY, JRCD,
1 JMAC, JMATYP, MATLAYO, LACCFLA)
  S11 = ARRAY(1)
  S22 = ARRAY(2)
  S12 = ARRAY(4)
  CALL GETVRM('E', ARRAY, JARRAY, FLGRAY, JRCD,
1 JMAC, JMATYP, MATLAYO, LACCFLA)
  E12 = ARRAY(4)

C
C DAMAGE INDEX: = 0 IF NO STRAIN TO PREVENT DIVIDE BY ZERO
C
  IF (E12.NE.0) THEN
    DAMAGE = (3.D0*ALPHA*G12*S12**2 -
&            2.D0*ALPHA*(S12**3)/E12) /
&            (1.D0 + 3.D0*ALPHA*G12*S12**2)
  ELSE
    DAMAGE = 0.D0
  ENDIF
```



C

$$F1 = S12^{**2} / (2.D0 * G12) + 0.75D0 * ALPHA * S12^{**4}$$

$$F2 = SC^{**2} / (2.D0 * G12) + 0.75D0 * ALPHA * SC^{**4}$$

C

C MATRIX TENSILE/COMPRESSIVE FAILURE

IF (EM .LT. 1.D0) THEN

IF (S22 .LT. 0.D0) THEN

$$EM = \text{SQRT}((S22/YC)^{**2} + F1/F2)$$

ELSE

$$EM = \text{SQRT}((S22/YT)^{**2} + F1/F2)$$

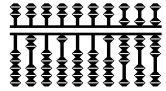
ENDIF

STATEV(1) = EM

ENDIF

C

Value of matrix failure modes is stored as
solution-dependent state variable

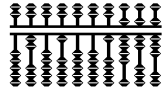


```
C FIBER-MATRIX SHEAR FAILURE
  IF (EFS .LT. 1.D0) THEN
    IF (S11 .LT. 0.D0) THEN
      EFS = SQRT((S11/XC)**2 + F1/F2)
    ELSE
      EFS = 0.D0
    ENDIF
    STATEV(2) = EFS
  ENDIF

C
C UPDATE FIELD VARIABLES
  FIELD(1) = 0.D0
  FIELD(2) = 0.D0
  IF (EM .GT. 1.D0) FIELD(1) = 1.D0
  IF (EFS .GT. 1.D0) FIELD(2) = 1.D0
  FIELD(3) = DAMAGE
  STATEV(3) = FIELD(3)

C

RETURN
END
```



Results

- The material model implemented with **USDFLD** in this example does a reasonably good job of modeling the experimentally observed behavior (see Figure 4–2).
- The extent of damage predicted in the composite plate (see Figure 4–3) during the analysis was very similar to the damage seen in the experimental specimens.

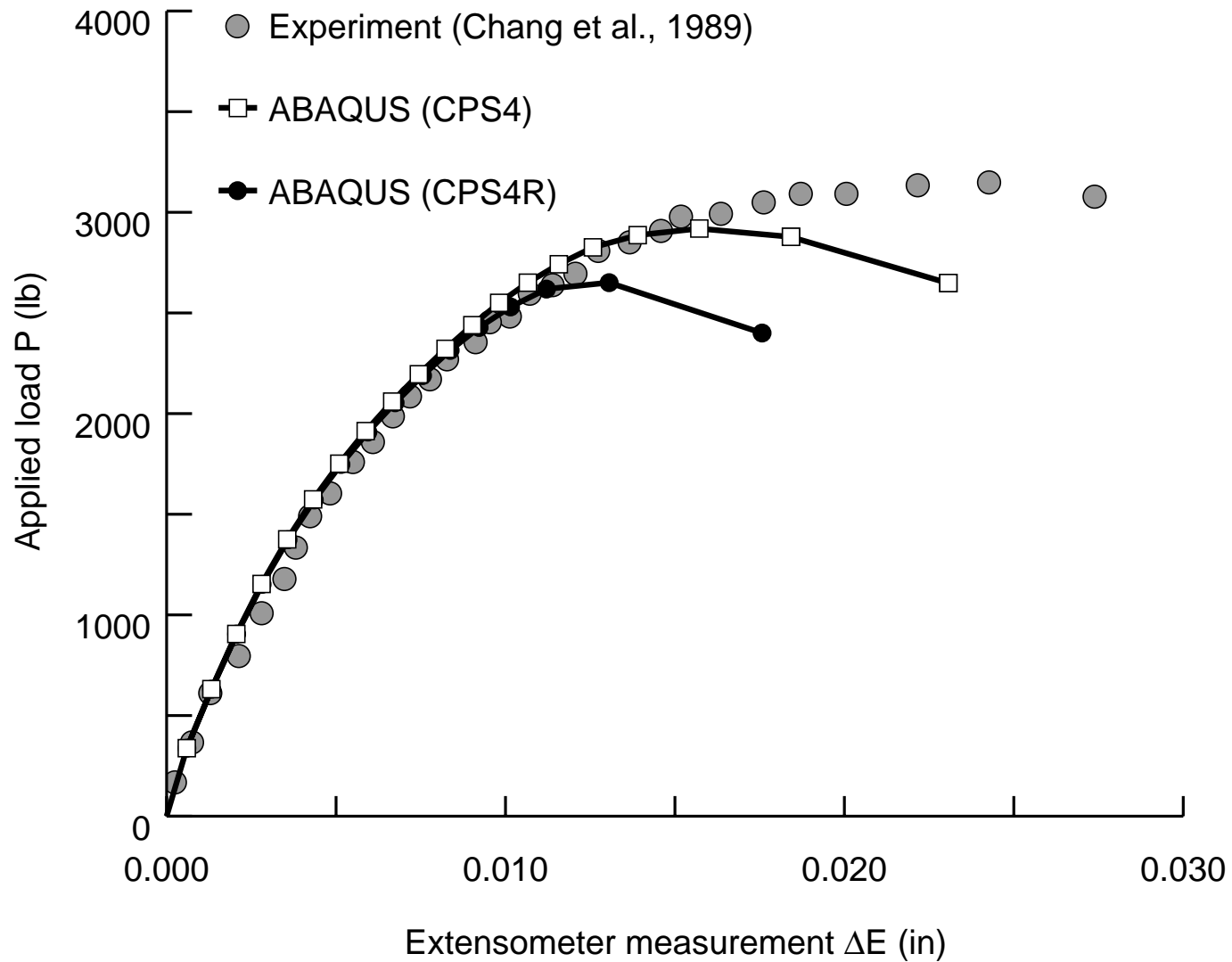


Figure 4–2. Experimental and Numerical Load-Deflection Curves

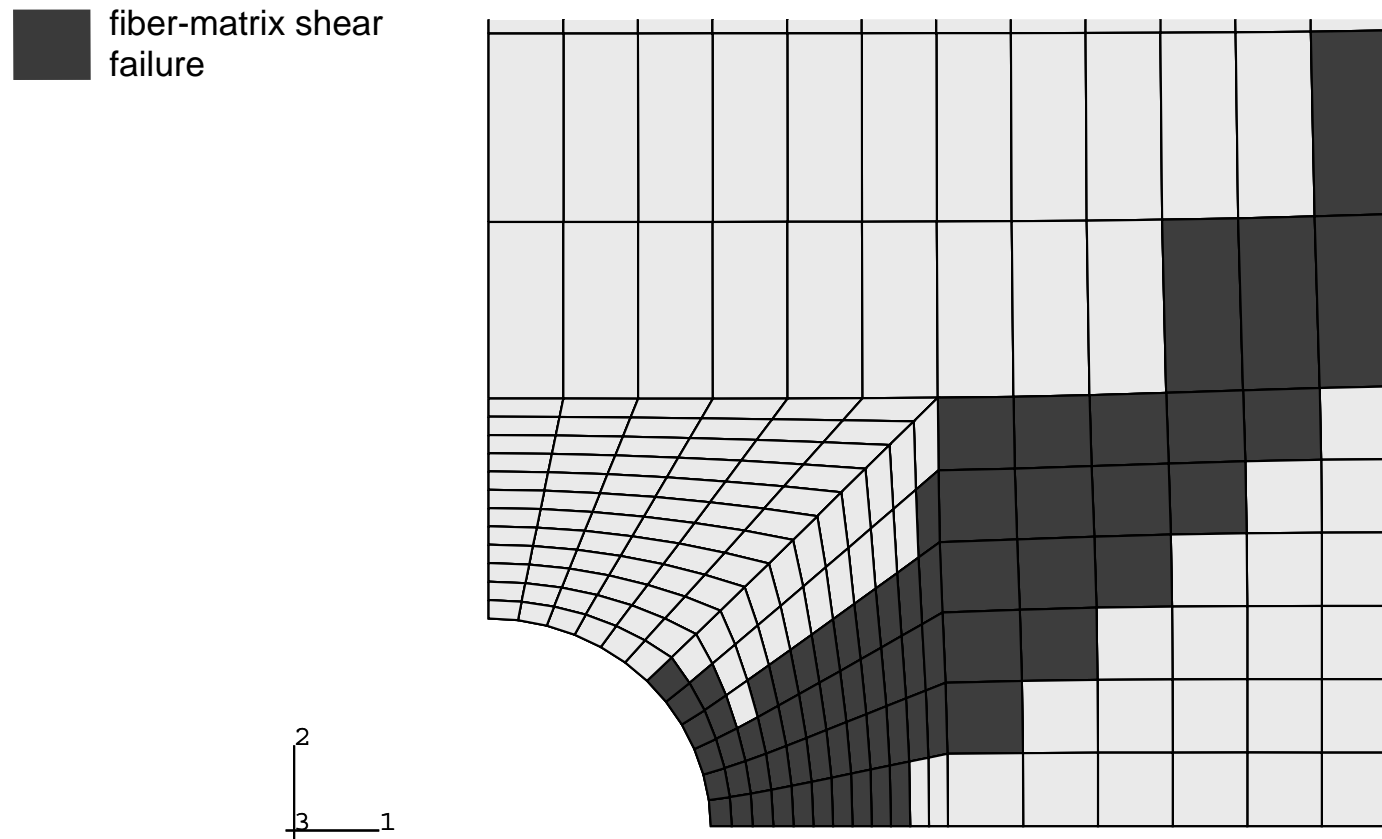
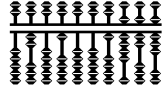
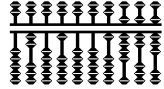
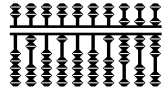


Figure 4–3. Material Damage in the Plate

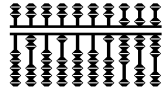


Remarks

- The values of the failure indices are not assigned directly to the f_i : instead, they are stored as solution-dependent state variables.
 - Only if the value of a failure index exceeds 1.0 is the corresponding user-defined field variable set equal to 1.0.
 - After the failure index has exceeded 1.0, the associated f_i continues to have the value 1.0 even though the stresses may reduce significantly, which ensures that the material does not “heal” after it has become damaged.



- The material model implemented with **USDFLD** assumes that after failure occurs the stresses in the failed directions drop to zero immediately, which corresponds to brittle failure with no energy absorption.
- This assumption is not very realistic: in reality, the stress-carrying capacity degrades gradually with increasing strain after failure occurs.
 - Hence, the behavior of the composite after onset of failure is not likely to be captured well by this model.
- Moreover, the instantaneous loss of stress-carrying capacity also makes the postfailure analysis results strongly dependent on the refinement of the finite element mesh and the finite element type used.



- In this example the only significant nonlinearity in the model is failure of the composite material. Hence, fixed time incrementation can be used effectively.
 - However, the results of this analysis are highly sensitive to the size of the time increment. The time increment used in the model shown, $\Delta t = 0.05$, is close to the largest allowable value (see Figure 4–4 and Figure 4–5).
- If other nonlinearities were present in the analysis, the automatic time incrementation algorithm would likely be needed.
 - In those types of analyses the variable **PNEWDT** would have to be used in user subroutine **USDFLD** to help control the size of the time increment.

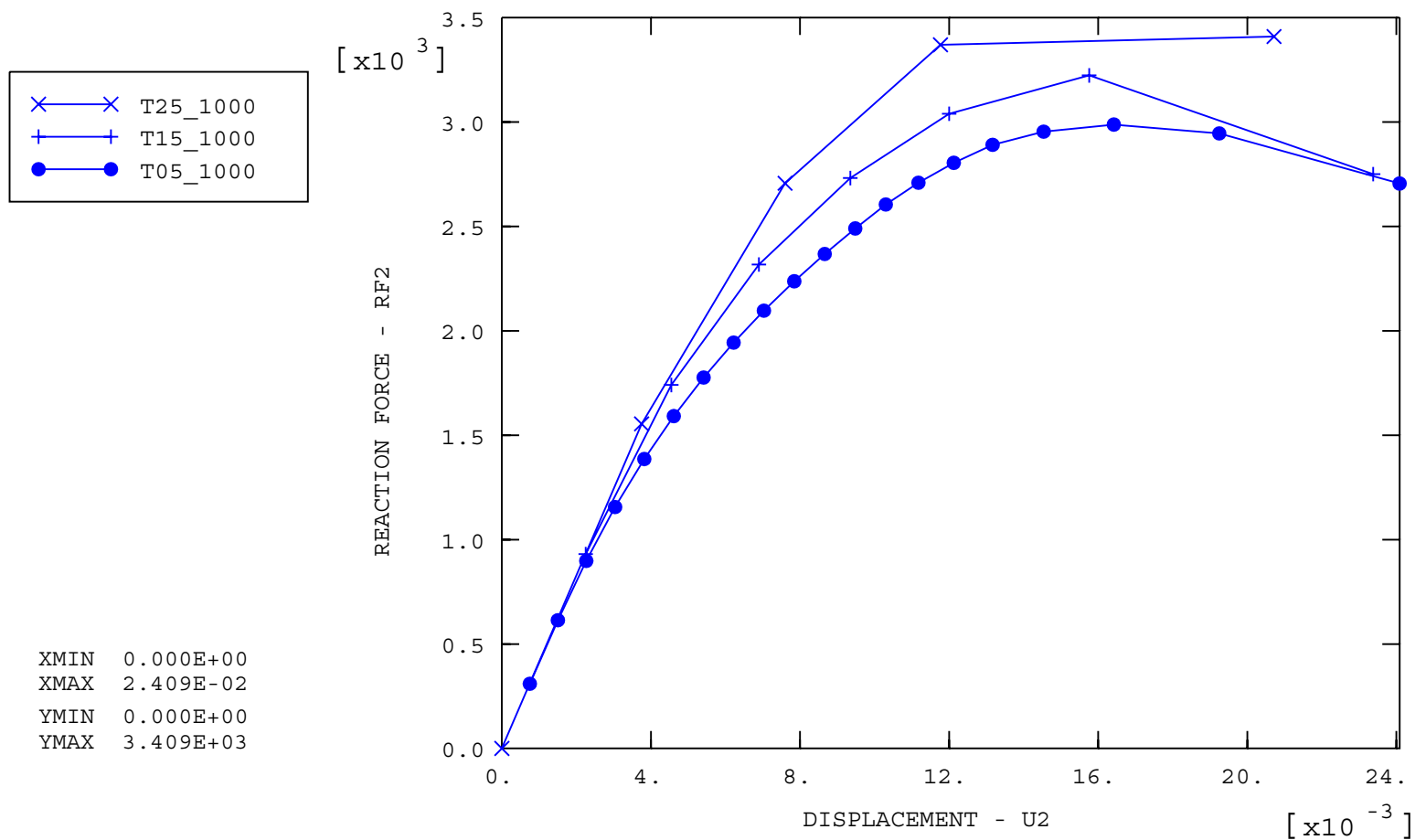
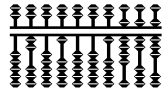


Figure 4–4. Force Deflection Curves for Analyses with $\Delta t = 0.05$, $\Delta t = 0.15$, and $\Delta t = 0.25$

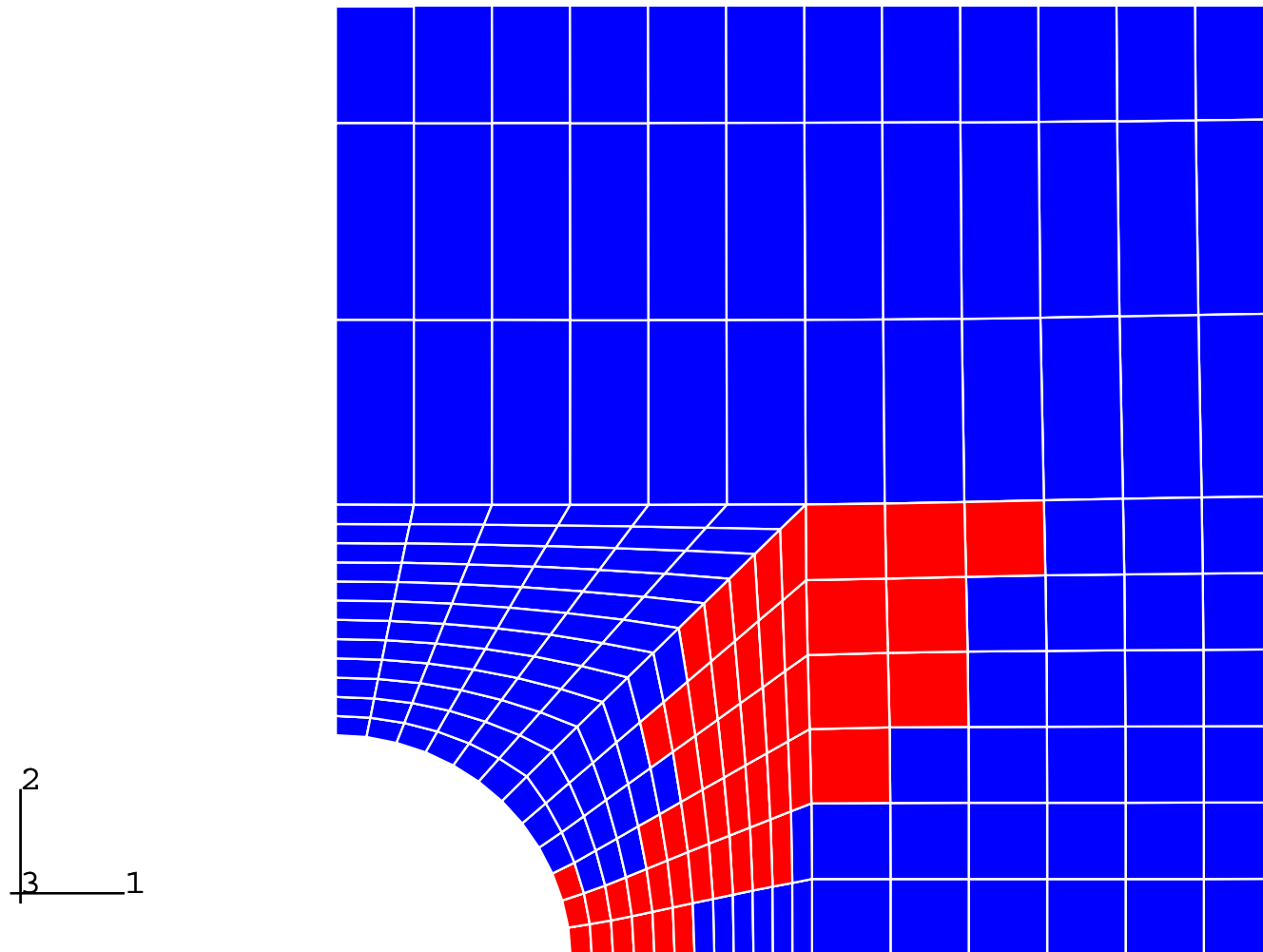
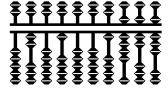
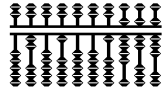


Figure 4–5. Extent of Material Damage in Analysis with $\Delta t = 0.25$

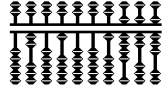


Lecture 5

User Subroutine: URDFIL

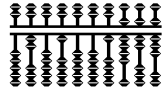
Overview

- Introduction
- ABAQUS Usage
- URDFIL Subroutine Interface
- Example: Using URDFIL to Terminate an Analysis

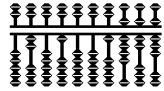


Introduction

- Subroutine **URDFIL** is used to read the results (**.fil**) file at the end of an increment.
 - Thus, the user can examine the results as the analysis is running.
 - This information can be used to make decisions such as whether to stop the analysis.
 - Results can also be extracted from the results file, stored in **COMMON** blocks, and passed into other subroutines.



- Subroutine **URDFIL** must call the utility routine **DBFILE** to read records from the results file.
 - A detailed discussion of this routine is provided in the ABAQUS/Standard User's Manual, Section 5.1.4.
- Subroutine **URDFIL** can call the utility routine **POSFIL** to begin reading the results file at a specified step and increment.
 - The default behavior is to begin reading the data from the beginning of the file.
 - A detailed discussion of this routine is provided in the ABAQUS/Standard User's Manual, Section 5.1.4.



ABAQUS Usage

- If an analysis requests that data be written to the results (**.fil**) file using the ***EL FILE**, ***NODE FILE**, ***CONTACT FILE**, or ***ENERGY FILE** options, subroutine **URDFIL** will be called at the end of any increment in which new information is written to the results file.
 - The coding for the subroutine must also be provided via the **user** parameter on the command line.

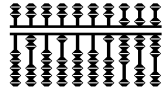
Utility Routine POSFIL

- The utility routine **POSFIL** is used to locate the position of a specific increment of results data stored on the results (**.fil**) file.
 - Once this position is found, the data for that increment can be read.
- The interface for this utility is:

```
CALL POSFIL(NSTEP, NINC, ARRAY, JRCD)
```

Variables to be provided to the utility routine:

- The desired step (**NSTEP**). If this variable is set to 0, the first available step will be read.
- The desired increment (**NINC**). If this variable is set to 0, the first available increment of the specified step will be read.



Variables returned from the utility routine:

- A real array (**ARRAY**) containing the values of record 2000 from the results file for the requested step and increment.
- A return code (**JRCD**). If it has a value of 0, the specified increment was found; if it has a value of 1, the specified increment was not found.
- If the step and increment requested are not found on the results file, **POSFIL** will return an error and leave the user positioned at the end of the results file.
- **POSFIL** cannot be used to move backward in the results file.
 - The user cannot use **POSFIL** to find a given increment in the file and then make a second call to **POSFIL** later to read an increment earlier than the first one found.

Utility Routine DBFILE

- The utility routine **DBFILE** is used to extract data from the results file.
- The interface for this utility is:

```
CALL DBFILE (LOP, ARRAY, JRCD)
```

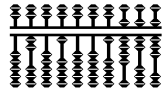
The only variable to be provided to the utility routine is:

- A flag, **LOP**, which must be set to 0 when this utility is used in **URDFIL**.

Variables returned from the utility routine are:

- The array, **ARRAY**, containing one record from the results file.
- The flag **JRCD** is returned as nonzero if an end-of-file marker is read when **DBFILE** is called with **LOP=0**.

- The formats of the data records for the results file are described in the ABAQUS/Standard User's Manual, Section 5.1.2.
 - **ARRAY** must be dimensioned adequately in the user's routines to contain the largest record on the file.
 - For almost all cases 500 words is sufficient.
 - The exceptions arise if the problem definition includes user elements or user materials that use more than this many state variables.
 - When the results file has been written on a system on which ABAQUS runs in double precision, **ARRAY** must be declared double precision in the user's routine.



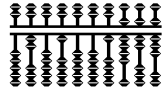
URDFIL Subroutine Interface

The interface to user subroutine `URDFIL` is:

```
      SUBROUTINE URDFIL(LSTOP, LOVRWRT, KSTEP, KINC,  
1          DTIME, TIME)  
  
C  
      INCLUDE 'ABA_PARAM.INC'  
  
C  
      DIMENSION ARRAY(513), JRRAY(NPRECD, 513), TIME(2)  
      EQUIVALENCE (ARRAY(1), JRRAY(1 ,1))  
  
      user coding to read the results file  
  
      RETURN  
      END
```

Variables to be Defined

- The flag (**LSTOP**) to indicate whether an analysis should continue.
 - The analysis will be terminated if **LSTOP** is set to 1.
 - Otherwise, the analysis will continue.
- The flag (**LOVRWRT**) to indicate that the information written to the results file for the current increment can be overwritten.
 - If **LOVRWRT** is set to 1, information for the current increment will be overwritten by information written to the results file in a subsequent increment unless the current increment is the final increment written to the results file.
 - The purpose of this flag is to reduce the size of the results file by allowing information for an increment to be overwritten by information for a subsequent increment.



- The variable **DTIME** allows the user to provide input to the automatic time incrementation algorithms in ABAQUS (if automatic time incrementation is chosen).
 - It is passed in as the size of the next time increment to be taken and can be updated to increase or reduce the time increment.
 - If automatic time incrementation is not selected in the analysis procedure, updated values of **DTIME** are ignored.

Variables for Information Only

- The step (**KSTEP**) and increment number (**KINC**) in which the routine is being called.
- The value of step time (**TIME (1)**) and the value of total time (**TIME (2)**) at the end of the current increment.

Example: Using URDFIL to Terminate an Analysis

- In this example the values of Mises stress and the displacement in the 2-direction of a specific node, 63, are monitored, and if they exceed a given value, the analysis is stopped.
- The structure is a cantilever beam subjected to a tip load in the 2-direction.

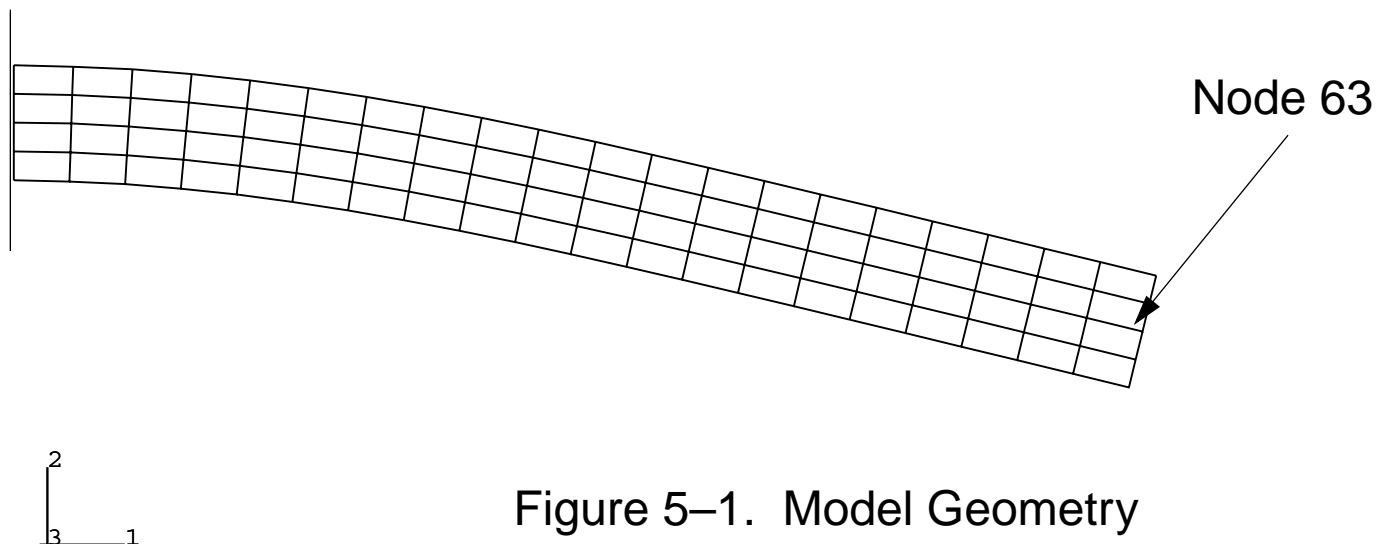
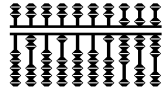


Figure 5–1. Model Geometry



Input Data

***HEADING**

Demonstration of URDFIL user subroutine

***RESTART,WRITE**

***NODE**

1, 0, 0

21, 1, 0

85, 0, .1

105 ,1, .1

***NGEN, NSET=BOTTOM**

1, 21, 1

***NGEN, NSET=TOP**

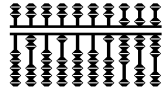
85, 105, 1

***NFILL, NSET=ALL**

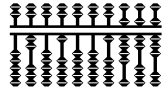
BOTTOM, TOP, 4, 21

***NSET, NSET=LEFT, GEN**

1, 85, 21

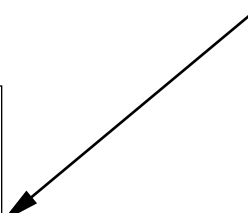


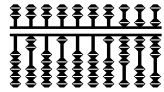
```
*ELEMENT, TYPE=CPS4R
1, 1, 2, 23, 22
*ELGEN, ELSET=ALL
1, 20, 1, 1, 4, 21, 20
*SOLID SECTION, ELSET=ALL, MATERIAL=STEEL, ORIENTATION=BEAM
0.1
*ORIENTATION, NAME=BEAM
1, 0, 0, 0, 1, 0
1, 0
*MATERIAL, NAME=STEEL
*ELASTIC
2E11, 0.3
*PLASTIC
2E8, 0
2E9, 0.1
*BOUNDARY
LEFT, 1, 2, 0
```



```
*STEP, NLGEOM, INC=30
*STATIC
0.1, 1
*CLOAD
63, 2, -1E6
*NODE PRINT, FREQ=0
*EL PRINT, FREQ=0
*EL FILE, FREQUENCY=1
SINV
*NODE FILE, FREQUENCY=1
U
*END STEP
```

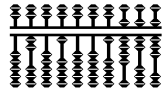
Write data to the results file frequently to ensure that you can monitor the progress of the analysis.





User Subroutine

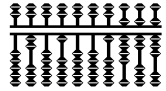
```
      SUBROUTINE URDFIL(LSTOP, LOVRWRT, KSTEP, KINC,  
1          DTIME, TIME)  
  
C  
      INCLUDE 'ABA_PARAM.INC'  
      DIMENSION ARRAY(513), JRRAY(NPRECD, 513), TIME(2)  
      EQUIVALENCE (ARRAY(1), JRRAY(1,1))  
      PARAMETER (TOL=5.0D8)  
      PARAMETER (DEFL=1.5D-1)  
      LMISES=0  
      LDEFL=0  
  
C  
C Assume that we do not mind if .fil file results are  
C overwritten.  
C  
      LOVRWRT=1  
  
C  
C Find current increment
```



```
C
      CALL POSFIL(KSTEP, KINC, ARRAY, JRCD)
C
C Loop over all of the records
C
      DO K1=1,999999
          CALL DBFILE(0,ARRAY,JRCD)
          IF (JRCD .NE. 0) GO TO 110
          KEY=JRRAY(1,2)
C
C Record 1 contains element information for subsequent
C records
C
          IF (KEY .EQ. 1) THEN
              IELM = JRRAY(1, 3)
              IMATPT = JRRAY(1, 4)
          END IF
C
```

Loop over large number to ensure that all records are read.

Check for end-of-file.



C Record 12 contains values for SIN V

C

```
      IF (KEY .EQ. 12) THEN
        IF (ARRAY(3) .GT. TOL) THEN
          LMISES=1
          GOTO 210
        END IF
      END IF
```

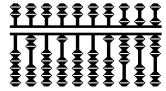
C

C Record 101 contains U

C

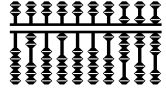
```
      IF (KEY .EQ. 101) THEN
        IF (JRRAY(1, 3) .EQ. 63) THEN
          IF (ABS(ARRAY(5)) .GT. DEFL) THEN
            LDEFL=1
            GOTO 210
          END IF
        END IF
```

Write why analysis is
terminated to message file



```
        END IF
      END DO
110 CONTINUE
C
210 IF (LMISES .EQ. 1) THEN
        WRITE(7, *)
        WRITE(7, 1023) TOL, IELEM, IMATPT
        WRITE(7, *)
        LSTOP=1
      END IF
      IF (LDEFL .EQ. 1) THEN
        WRITE(7, *)
        WRITE(7, 1024) DEFL
        WRITE(7, *)
        LSTOP=1
      END IF

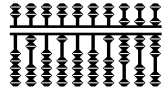
      RETURN
```



C

C

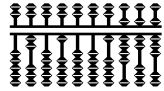
```
1023 FORMAT ('***NOTE: ANALYSIS TERMINATES MISES
           &STRESS EXCEEDS', 2X, E9.3, 1X, 'IN ELEMENT', 1X, I6,
           &1X, 'AT INT. PT.', 1X, I6)
1024 FORMAT ('***NOTE: ANALYSIS TERMINATES AS DEFLECTION
           &OF NODE 63 EXCEEDS', 2X, E9.3)
END
```

Remarks

- The analysis terminates when only 12.3% of the defined load is applied to the model because the tolerance for Mises stress was exceeded in element 1.

```
***NOTE: ANALYSIS TERMINATES MISES STRESS EXCEEDS  
0.500E+09 IN ELEMENT      1 AT INT. PT.      1
```



Lecture 6

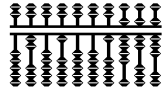
Writing a UMAT or VUMAT

Overview

- Motivation
- Steps Required in Writing a UMAT or VUMAT
- UMAT Interface
- Examples
- VUMAT Interface
- Examples

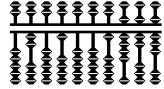
Overview

- ABAQUS/Standard and ABAQUS/Explicit have interfaces that allow the user to implement general constitutive equations.
 - In ABAQUS/Standard the user-defined material model is implemented in user subroutine **UMAT**.
 - In ABAQUS/Explicit the user-defined material model is implemented in user subroutine **VUMAT**.
- Use **UMAT** and **VUMAT** when none of the existing material models included in the ABAQUS material library accurately represents the behavior of the material to be modeled.



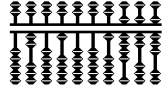
- These interfaces make it possible to define any (proprietary) constitutive model of arbitrary complexity.
- User-defined material models can be used with any ABAQUS structural element type.
- Multiple user materials can be implemented in a single **UMAT** or **VUMAT** routine and can be used together.

In this lecture the implementation of material models in **UMAT** or **VUMAT** will be discussed and illustrated with a number of examples.

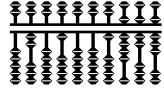


Motivation

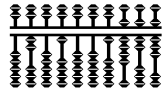
- Proper testing of advanced constitutive models to simulate experimental results often requires complex finite element models.
 - Advanced structural elements
 - Complex loading conditions
 - Thermomechanical loading
 - Contact and friction conditions
 - Static and dynamic analysis



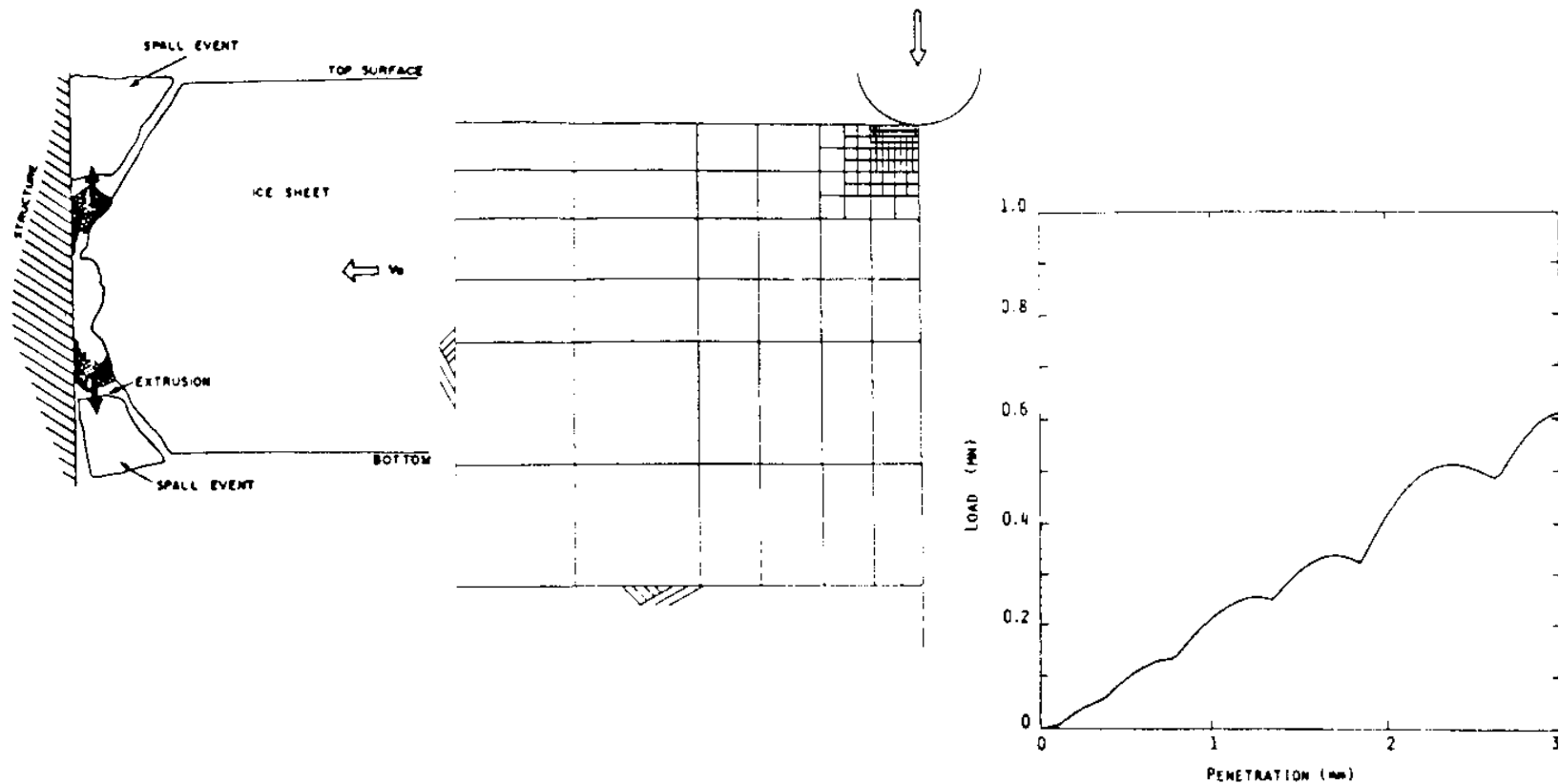
- Special analysis problems occur if the constitutive model simulates material instabilities and localization phenomena.
 - Special solution techniques are required for quasi-static analysis.
 - Robust element formulations should be available.
 - Explicit dynamic solution algorithms with robust, vectorized contact algorithms are desired.
- In addition, robust features are required to present and visualize the results.
 - Contour and path plots of state variables.
 - X – Y plots.
 - Tabulated results.

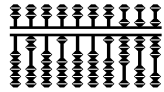


- The material model developer should be concerned only with the development of the material model and not the development and maintenance of the FE software.
 - Developments unrelated to material modeling
 - Porting problems with new systems
 - Long-term program maintenance of user-developed code

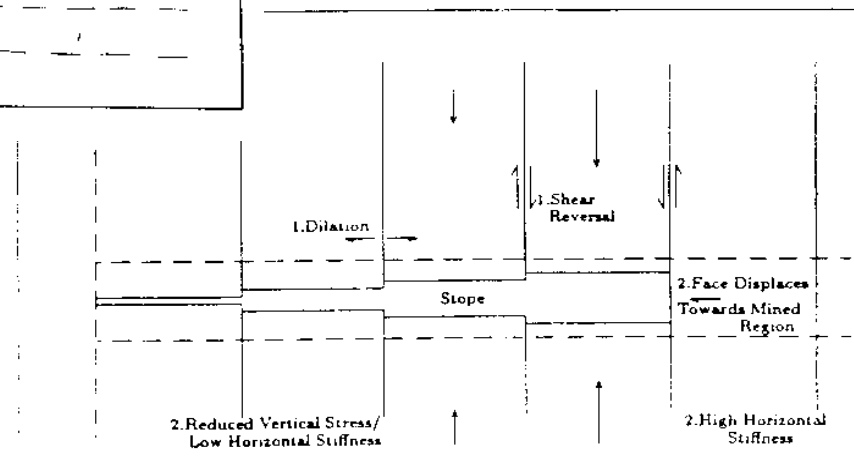
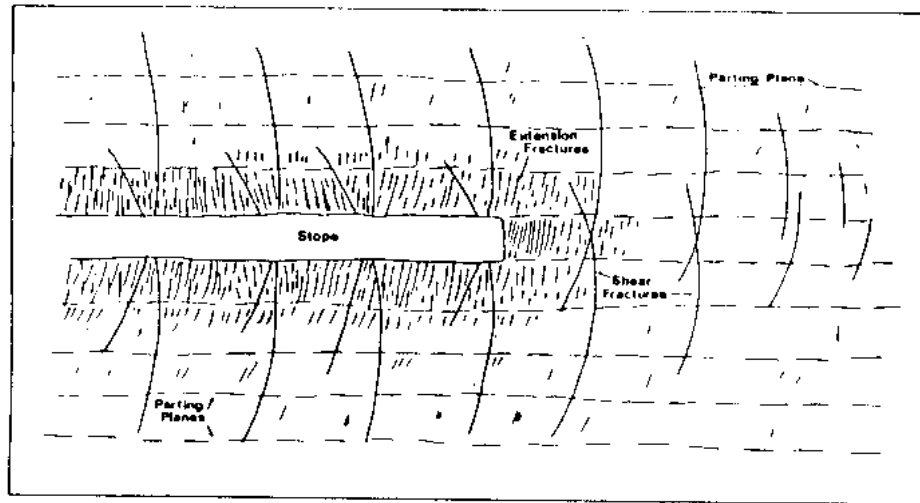


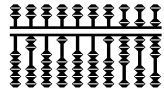
- “*Finite Element Modelling of the Damage Process in Ice,*”
R. F. McKenna, I. J. Jordaan, and J. Xiao, ABAQUS Users’ Conference Proceedings, 1990.



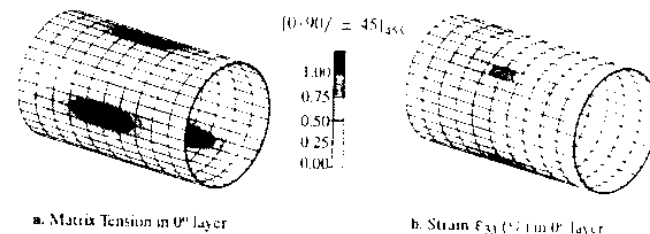
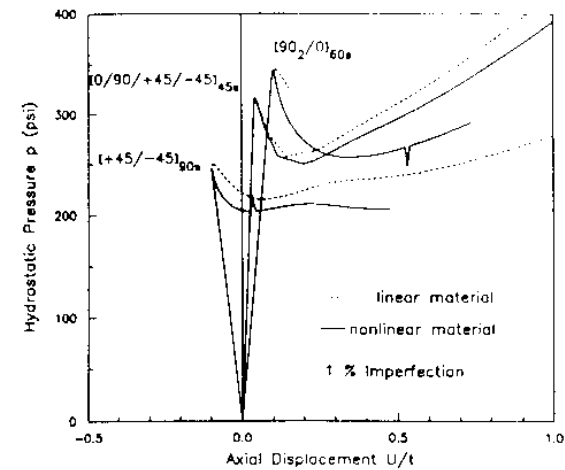
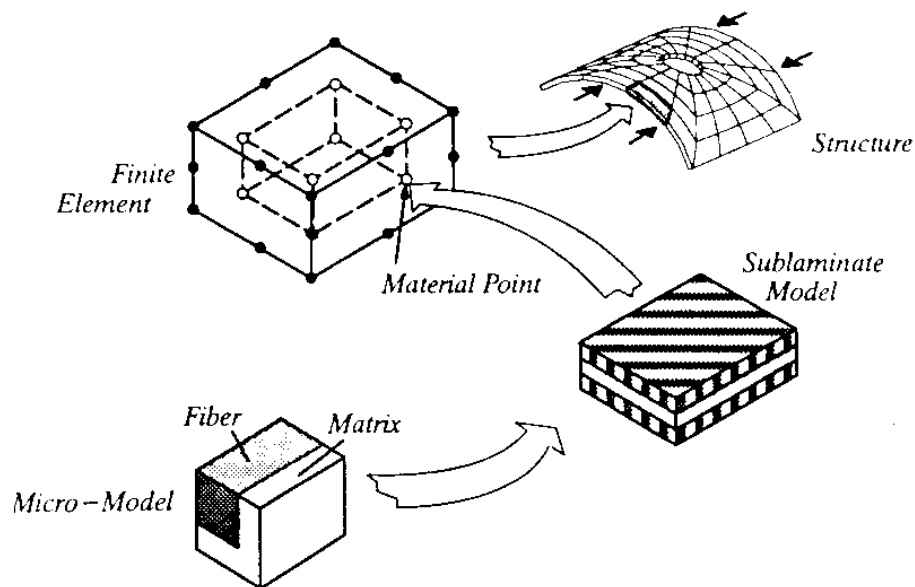


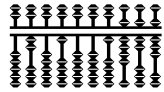
- “*The Numerical Simulation of Excavations in Deep Level Mining*,” M. F. Snyman, G. P. Mitchell, and J. B. Martin, ABAQUS Users’ Conference Proceedings, 1991.



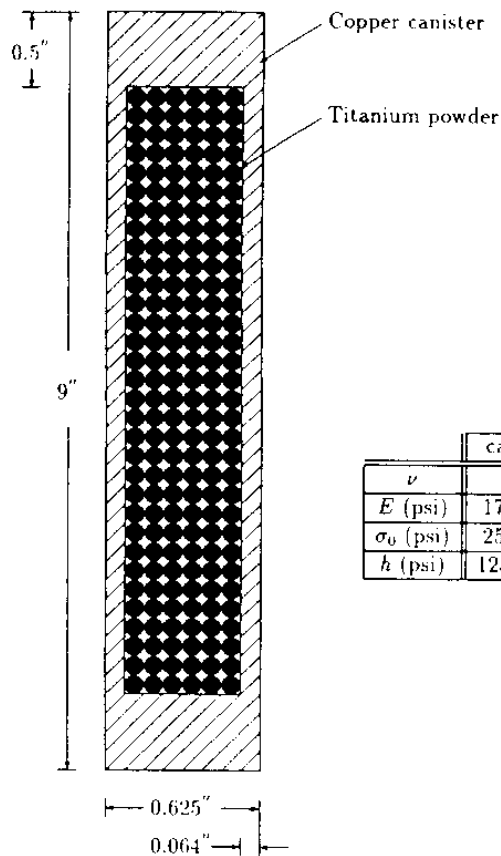


- “*Combined Micromechanical and Structural Finite Element Analysis of Laminated Composites*,” R. M. HajAli, D. A. Pecknold, and M. F. Ahmad, ABAQUS Users’ Conference Proceedings, 1993.

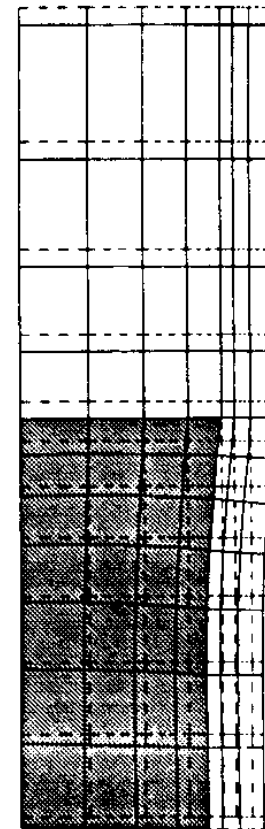


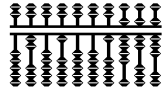


- “*Deformation Processing of Metal Powders: Cold and Hot Isostatic Pressing,*” R. M. Govindarajan and N. Aravas, private communication, 1993.

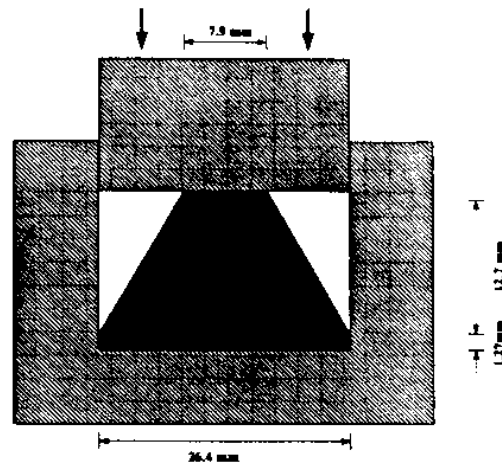


	canister	powder
ν	0.33	0.327
E (psi)	17×10^6	16.1×10^6
σ_0 (psi)	25×10^3	62×10^3
h (psi)	125×10^3	688×10^3

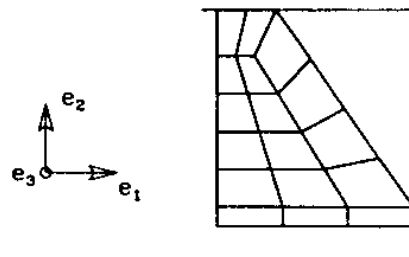




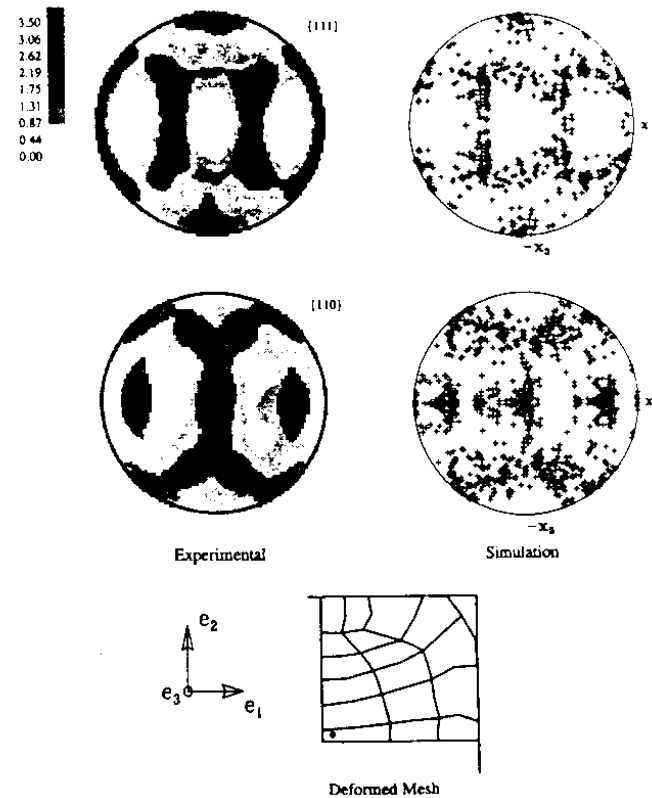
- “*Macroscopic Shape Change and Evolution of Crystallographic Texture in Pre-textured FCC Metals*,” S. R. Kalidindi and Anand, Acta Metallurgica, 1993.



(a)

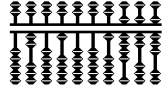


(b)

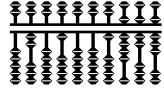


Steps Required in Writing a UMAT or VUMAT

- Proper definition of the constitutive equation, which requires one of the following:
 - Explicit definition of stress (Cauchy stress for large-strain applications)
 - Definition of the stress rate only (in corotational framework)
- Furthermore, it is likely to require:
 - Definition of dependence on time, temperature, or field variables
 - Definition of internal state variables, either explicitly or in rate form



- Transformation of the constitutive rate equation into an incremental equation using a suitable integration procedure:
 - Forward Euler (explicit integration)
 - Backward Euler (implicit integration)
 - Midpoint method

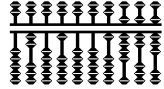


This is the hard part! Forward Euler (explicit) integration methods are simple but have a *stability limit*,

$$|\Delta\epsilon| < \Delta\epsilon_{stab},$$

where $\Delta\epsilon_{stab}$ is usually less than the elastic strain magnitude.

- For explicit integration the time increment must be controlled.
- For implicit or midpoint integration the algorithm is more complicated and often requires local iteration. However, there is usually no stability limit.
- An incremental expression for the internal state variables must also be obtained.

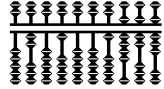


- Calculation of the (consistent) Jacobian (required for ABAQUS/Standard **UMAT** only).
- For small-deformation problems (e.g., linear elasticity) or large-deformation problems with small volume changes (e.g., metal plasticity), the consistent Jacobian is

$$C = \frac{\partial \Delta \sigma}{\partial \Delta \epsilon} ,$$

where $\Delta \sigma$ is the increment in (Cauchy) stress and $\Delta \epsilon$ is the increment in strain. (In finite-strain problems, ϵ is an approximation to the logarithmic strain.)

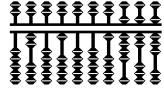
- This matrix may be nonsymmetric as a result of the constitutive equation or integration procedure.
- The Jacobian is often approximated such that a loss of quadratic convergence may occur.



- It is easily calculated for forward integration methods (usually the elasticity matrix).
- If large deformations with large volume changes are considered (e.g., pressure-dependent plasticity), the exact form of the consistent Jacobian

$$C = \frac{1}{J} \frac{\partial \Delta(J\sigma)}{\partial \Delta \epsilon}$$

should be used to ensure rapid convergence. Here, J is the determinant of the deformation gradient.

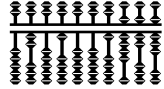


- Hyperelastic constitutive equations
 - Total-form constitutive equations relating the Cauchy stress σ and the deformation gradient F are commonly used to model, for example, rubber elasticity.
 - In this case, the consistent Jacobian is defined through:

$$\delta(J\sigma) = JC:\delta D,$$

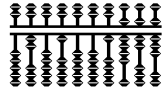
where $J = |F|$, C is the material Jacobian, and δD is the virtual rate of deformation, defined as

$$\delta D = \text{sym}(\delta F \cdot F^{-1}).$$



- Coding the **UMAT** or **VUMAT**:
 - Follow FORTRAN 77 or C conventions.
 - Make sure that the code can be vectorized (for **VUMAT** only, to be discussed later).
 - Make sure that all variables are defined and initialized properly.
 - Use ABAQUS utility routines as required.
 - Assign enough storage space for state variables with the *DEPVAR option.

- Verifying the **UMAT** or **VUMAT** with a small (one element) input file.
 1. Run tests with all displacements prescribed to verify the integration algorithm for stresses and state variables. Suggested tests include:
 - Uniaxial
 - Uniaxial in oblique direction
 - Uniaxial with finite rotation
 - Finite shear
 2. Run similar tests with load prescribed to verify the accuracy of the Jacobian.
 3. Compare test results with analytical solutions or standard ABAQUS material models, if possible. If the above verification is successful, apply to more complicated problems.



UMAT Interface

- These input lines act as the interface to a **UMAT** in which isotropic hardening plasticity is defined.

```
*MATERIAL, NAME=ISOPLAS
```

```
*USER MATERIAL, CONSTANTS=8, (UNSYMM)
```

```
30.E6, 0.3, 30.E3, 0., 40.E3, 0.1, 50.E3, 0.5
```

```
*DEPVAR
```

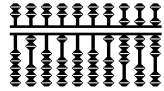
```
13
```

```
*INITIAL CONDITIONS, TYPE=SOLUTION
```

Data line to specify initial solution-dependent variables

```
*USER SUBROUTINES, (INPUT=file_name)
```

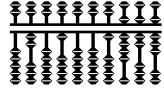
- The ***USER MATERIAL** option is used to input material constants for the **UMAT**. The unsymmetric equation solution technique will be used if the **UNSYMM** parameter is used.



- The *DEPVAR option is used to allocate space at each material point for solution-dependent state variables (SDVs).
- The *INITIAL CONDITIONS, TYPE=SOLUTION option is used to initialize SDVs if they are starting at a nonzero value.
- Coding for the **UMAT** is supplied in a separate file. The **UMAT** is invoked with the ABAQUS execution procedure, as follows:

```
abaqus job=... user=....
```

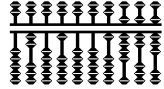
- The user subroutine must be invoked in a restarted analysis because user subroutines are not saved on the restart file.



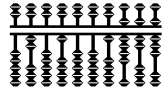
- Additional notes:
 - If a constant material Jacobian is used and no other nonlinearity is present, reassembly can be avoided by invoking the quasi-Newton method with the input line

***SOLUTION TECHNIQUE, REFORM KERNEL= n**

- n is the number of iterations done without reassembly.
- This does not offer advantages if other nonlinearities (such as contact changes) are present.



- Solution-dependent state variables can be output with identifiers SDV1, SDV2, etc. Contour, path, and X–Y plots of SDVs can be plotted in ABAQUS/Viewer.
- Include only a single **UMAT** subroutine in the analysis. If more than one material must be defined, test on the material name in **UMAT** and branch.



- The **UMAT** subroutine header is shown below:

```
SUBROUTINE UMAT(STRESS, STATEV, DDSDE, SSE, SPD, SCD, RPL,
1 DDSDDT, DRPLDE, DRPLDT, STRAN, DSTRAN, TIME, DTIME, TEMP, DTEMP,
2 PREDEF, DPRED, CMNAME, NDI, NSHR, NTENS, NSTATV, PROPS, NPROPS,
3 COORDS, DROT, PNEWDT, CELENT, DFGRD0, DFGRD1, NOEL, NPT, LAYER,
4 KSPT, KSTEP, KINC)
```

C

```
INCLUDE 'ABA_PARAM.INC'
```

C

```
CHARACTER*8 CMNAME
```

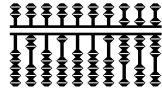
C

```
DIMENSION STRESS(NTENS), STATEV(NSTATV), DDSDE(NTENS, NTENS),
1 DDSDDT(NTENS), DRPLDE(NTENS), STRAN(NTENS), DSTRAN(NTENS),
2 PREDEF(1), DPRED(1), PROPS(NPROPS), COORDS(3), DROT(3, 3),
3 DFGRD0(3, 3), DFGRD1(3, 3)
```

- The include statement sets the proper precision for floating point variables (**REAL*8** on most machines).
- The material name, **CMNAME**, is an 8-byte character variable.

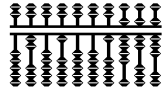
UMAT Variables

- The following quantities are available in **UMAT**:
 - Stress, strain, and SDVs at the start of the increment
 - Strain increment, rotation increment, and deformation gradient at the start and end of the increment
 - Total and incremental values of time, temperature, and user-defined field variables
 - Material constants, material point position, and a characteristic element length
 - Element, integration point, and composite layer number (for shells and layered solids)
 - Current step and increment numbers



- The following quantities must be defined:
 - Stress, SDVs, and material Jacobian
- The following variables may be defined:
 - Strain energy, plastic dissipation, and “creep” dissipation
 - Suggested new (reduced) time increment

Complete descriptions of all parameters are provided in the **UMAT** section in Chapter 24 of the ABAQUS/Standard User's Manual.



- The header is usually followed by dimensioning of local arrays. It is good practice to define constants via parameters and to include comments.

```

      DIMENSION EELAS(6), EPLAS(6), FLOW(6)
C
      PARAMETER(ZERO=0.D0, ONE=1.D0, TWO=2.D0, THREE=3.D0, SIX=6.D0,
1          ENUMAX=.4999D0, NEWTON=10, TOLER=1.0D-6)
C
C -----
C   UMAT FOR ISOTROPIC ELASTICITY AND ISOTROPIC MISES PLASTICITY
C   CANNOT BE USED FOR PLANE STRESS
C -----
C   PROPS(1) - E
C   PROPS(2) - NU
C   PROPS(3..) - YIELD AND HARDENING DATA
C   CALLS UHARD FOR CURVE OF YIELD STRESS VS. PLASTIC STRAIN
C -----

```

- The **PARAMETER** assignments yield accurate floating point constant definitions on any platform.

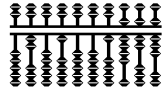
UMAT Utilities

- Utility routines **SINV**, **SPRINC**, **SPRIND**, and **ROTSIG** can be called to assist in coding **UMAT**.
 - **SINV** will return the first and second invariants of a tensor.
 - **SPRINC** will return the principal values of a tensor.
 - **SPRIND** will return the principal values and directions of a tensor.
 - **ROTSIG** will rotate a tensor with an orientation matrix.
 - **XIT** will terminate an analysis and close all files associated with the analysis properly.
- For details regarding the arguments required in making these calls, refer to the **UMAT** section in Chapter 24 of the ABAQUS/Standard User's Manual and the examples in this lecture.

UMAT Conventions

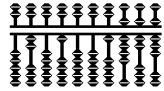
- Stresses and strains are stored as vectors.
 - For plane stress elements: σ_{11} , σ_{22} , σ_{12} .
 - For (generalized) plane strain and axisymmetric elements: σ_{11} , σ_{22} , σ_{33} , σ_{12} .
 - For three-dimensional elements: σ_{11} , σ_{22} , σ_{33} , σ_{12} , σ_{13} , σ_{23} .
- The shear strain is stored as engineering shear strain,

$$\gamma_{12} = 2\varepsilon_{12}.$$
- The deformation gradient, F_{ij} , is always stored as a three-dimensional matrix.

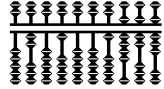


UMAT Formulation Aspects

- For geometrically nonlinear analysis the strain increment and incremental rotation passed into the routine are based on the Hughes-Winget formulae.
 - Linearized strain and rotation increments are calculated in the mid-increment configuration.
 - Approximations are made, particularly if rotation increments are large: more accurate measures can be obtained from the deformation gradient if desired.
- The user must define the Cauchy stress: when this stress reappears during the next increment, it will have been rotated with the incremental rotation, **DROT**, passed into the subroutine.
 - The stress tensor can be rotated back using the utility routine **ROTSIG** if this is not desired.



- If the *ORIENTATION option is used in conjunction with **UMAT**, stress and strain components will be in the local system (again, this basis system rotates with the material in finite-strain analysis).
- Tensor state variables must be rotated in the subroutine (use **ROTSIG**).
- If **UMAT** is used with reduced-integration elements or shear flexible shell or beam elements, the hourglass stiffness and the transverse shear stiffness must be specified with the *HOURGLASS STIFFNESS and *TRANSVERSE SHEAR STIFFNESS options, respectively.



Usage Hints

- At the start of a new increment, the strain increment is extrapolated from the previous increment.
 - This extrapolation, which may sometimes cause trouble, is turned off with `*STEP, EXTRAPOLATION=NO`.
- If the strain increment is too large, the variable **PNEWDT** can be used to suggest a reduced time increment.
 - The code will abandon the current time increment in favor of a time increment given by **PNEWDT*DTIME**.
- The characteristic element length can be used to define softening behavior based on fracture energy concepts.

Example 1: Isotropic Isothermal Elasticity

Governing Equations

- Isothermal elasticity equation (with Lamé's constants):

$$\sigma_{ij} = \lambda \delta_{ij} \epsilon_{kk} + 2\mu \epsilon_{ij},$$

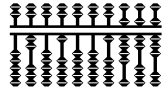
or in a Jaumann (corotational) rate form:

$$\dot{\sigma}_{ij}^J = \lambda \delta_{ij} \dot{\epsilon}_{kk} + 2\mu \dot{\epsilon}_{ij}.$$

- The Jaumann rate equation is integrated in a corotational framework:

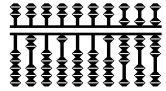
$$\Delta \sigma_{ij}^J = \lambda \delta_{ij} \Delta \epsilon_{kk} + 2\mu \Delta \epsilon_{ij}.$$

The appropriate coding is shown on the following pages.



Coding for Isotropic Isothermal Elasticity

```
C -----
C   UMAT FOR ISOTROPIC ELASTICITY
C   CANNOT BE USED FOR PLANE STRESS
C -----
C   PROPS(1) - E
C   PROPS(2) - NU
C -----
C
C       IF (NDI.NE.3) THEN
C           WRITE (7, *) 'THIS UMAT MAY ONLY BE USED FOR ELEMENTS
1       WITH THREE DIRECT STRESS COMPONENTS'
C           CALL XIT
C       ENDIF
C
C   ELASTIC PROPERTIES
C       EMOD=PROPS(1)
C       ENU=PROPS(2)
C       EBULK3=EMOD/(ONE-TWO*ENU)
C       EG2=EMOD/(ONE+ENU)
C       EG=EG2/TWO
C       EG3=THREE*EG
C       ELAM=(EBULK3-EG2)/THREE
```



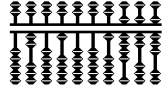
```
C
C   ELASTIC STIFFNESS
C
      DO K1=1, NDI
        DO K2=1, NDI
          DDSDDDE(K2, K1)=ELAM
        END DO
        DDSDDDE(K1, K1)=EG2+ELAM
      END DO
      DO K1=NDI+1, NTENS
        DDSDDDE(K1, K1)=EG
      END DO

C
C   CALCULATE STRESS
C
      DO K1=1, NTENS
        DO K2=1, NTENS
          STRESS(K2)=STRESS(K2)+DDSDDDE(K2, K1)*DSTRAN(K1)
        END DO
      END DO

C
      RETURN
      END
```

Remarks

- This very simple **UMAT** yields exactly the same results as the ABAQUS ***ELASTIC** option.
 - This is true even for large-strain calculations: all necessary large-strain contributions are generated by ABAQUS.
- The routine can be used with and without the ***ORIENTATION** option.
- It is usually straightforward to write a single routine that handles (generalized) plane strain, axisymmetric, and three-dimensional geometries.
 - Generally, plane stress must be treated as a separate case because the stiffness coefficients are different.
- The routine is written in incremental form as a preparation for subsequent elastic-plastic examples.



- Even for linear analysis, **UMAT** is called twice for the first iteration of each increment: once for assembly and once for recovery. Subsequently, it is called once per iteration: assembly and recovery are combined.
- A check is performed on the number of direct stress components, and the analysis is terminated by calling the subroutine, **XIT**.
 - A message is written to the message file (unit=7).

Example 2: Non-Isothermal Elasticity

Governing Equations

- Non-isothermal elasticity equation:

$$\sigma_{ij} = \lambda(T)\delta_{ij}\epsilon_{kk}^{el} + 2\mu(T)\epsilon_{ij}^{el}, \quad \epsilon_{ij}^{el} = \epsilon_{ij} - \alpha T\delta_{ij},$$

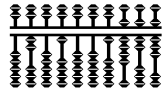
or in a Jaumann (corotational) rate form:

$$\dot{\sigma}_{ij}^J = \lambda\delta_{ij}\dot{\epsilon}_{kk}^{el} + 2\mu\dot{\epsilon}_{ij}^{el} + \dot{\lambda}\delta_{ij}\epsilon_{kk}^{el} + 2\dot{\mu}\epsilon_{ij}^{el}, \quad \dot{\epsilon}_{ij}^{el} = \dot{\epsilon}_{ij} - \alpha\dot{T}\delta_{ij}.$$

- The Jaumann rate equation is integrated in a corotational framework:

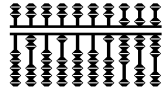
$$\Delta\sigma_{ij}^J = \lambda\delta_{ij}\Delta\epsilon_{kk}^{el} + 2\mu\Delta\epsilon_{ij}^{el} + \Delta\lambda\delta_{ij}\epsilon_{kk}^{el} + 2\Delta\mu\epsilon_{ij}^{el}, \quad \Delta\epsilon_{ij}^{el} = \Delta\epsilon_{ij} - \alpha\Delta T\delta_{ij}.$$

The appropriate coding is shown on the following pages.



Coding for Non-Isothermal Elasticity

```
C      LOCAL ARRAYS
C      -----
C      EELAS   - ELASTIC STRAINS
C      ETHERM  - THERMAL STRAINS
C      DTHERM  - INCREMENTAL THERMAL STRAINS
C      DELDSE  - CHANGE IN STIFFNESS DUE TO TEMPERATURE CHANGE
C      -----
C      DIMENSION EELAS(6), ETHERM(6), DTHERM(6), DELDSE(6,6)
C
C      PARAMETER(ZERO=0.D0, ONE=1.D0, TWO=2.D0, THREE=3.D0, SIX=6.D0)
C      -----
C      UMAT FOR ISOTROPIC THERMO-ELASTICITY WITH LINEARLY VARYING
C      MODULI - CANNOT BE USED FOR PLANE STRESS
C      -----
C      PROPS(1) - E(T0)
C      PROPS(2) - NU(T0)
C      PROPS(3) - T0
C      PROPS(4) - E(T1)
C      PROPS(5) - NU(T1)
C      PROPS(6) - T1
C      PROPS(7) - ALPHA
C      PROPS(8) - T_INITIAL
```

C ELASTIC PROPERTIES AT START OF INCREMENT

C

FAC1=(TEMP-PROPS(3))/(PROPS(6)-PROPS(3))

IF (FAC1 .LT. ZERO) FAC1=ZERO

IF (FAC1 .GT. ONE) FAC1=ONE

FAC0=ONE-FAC1

EMOD=FAC0*PROPS(1)+FAC1*PROPS(4)

ENU=FAC0*PROPS(2)+FAC1*PROPS(5)

EBULK3=EMOD/(ONE-TWO*ENU)

EG20=EMOD/(ONE+ENU)

EG0=EG20/TWO

ELAM0=(EBULK3-EG20)/THREE

C

C ELASTIC PROPERTIES AT END OF INCREMENT

C

FAC1=(TEMP+DTEMP-PROPS(3))/(PROPS(6)-PROPS(3))

IF (FAC1 .LT. ZERO) FAC1=ZERO

IF (FAC1 .GT. ONE) FAC1=ONE

FAC0=ONE-FAC1

EMOD=FAC0*PROPS(1)+FAC1*PROPS(4)

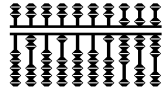
ENU=FAC0*PROPS(2)+FAC1*PROPS(5)

EBULK3=EMOD/(ONE-TWO*ENU)

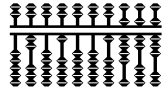
EG2=EMOD/(ONE+ENU)

EG=EG2/TWO

ELAM=(EBULK3-EG2)/THREE



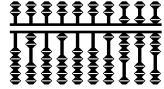
```
C
C   ELASTIC STIFFNESS AT END OF INCREMENT AND STIFFNESS CHANGE
C
      DO K1=1,NDI
        DO K2=1,NDI
          DDSDE(K2,K1)=ELAM
          DELDSE(K2,K1)=ELAM-ELAM0
        END DO
        DDSDE(K1,K1)=EG2+ELAM
        DELDSE(K1,K1)=EG2+ELAM-EG20-ELAM0
      END DO
      DO K1=NDI+1,NTENS
        DDSDE(K1,K1)=EG
        DELDSE(K1,K1)=EG-EG0
      END DO
C
C   CALCULATE THERMAL EXPANSION
C
      DO K1=1,NDI
        ETHERM(K1)=PROPS(7)*(TEMP-PROPS(8))
        DTHERM(K1)=PROPS(7)*DTEMP
      END DO
```



```
      DO K1=NDI+1,NTENS
        ETHERM(K1)=ZERO
        DTHERM(K1)=ZERO
      END DO

C
C   CALCULATE STRESS, ELASTIC STRAIN AND THERMAL STRAIN
C
      DO K1=1, NTENS
        DO K2=1, NTENS
          STRESS(K2)=STRESS(K2)+DDSDDE(K2,K1)*(DSTRAN(K1)-DTHERM(K1))
1          +DELDSE(K2,K1)*(STRAN(K1)-ETHERM(K1))
        END DO
        ETHERM(K1)=ETHERM(K1)+DTHERM(K1)
        EELAS(K1)=STRAN(K1)+DSTRAN(K1)-ETHERM(K1)
      END DO

C
C   STORE ELASTIC AND THERMAL STRAINS IN STATE VARIABLE ARRAY
C
      DO K1=1, NTENS
        STATEV(K1)=EELAS(K1)
        STATEV(K1+NTENS)=ETHERM(K1)
      END DO
      RETURN
      END
```



Remarks

- This **UMAT** yields exactly the same results as the ***ELASTIC** option with temperature dependence.
- The routine is written in incremental form, which allows generalization to more complex temperature dependence.

Example 3: Neo-Hookean Hyperelasticity

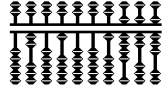
Governing Equations

- The *ELASTIC option does not work well for finite elastic strains because a proper finite-strain energy function is not defined.
- Hence, we define a proper strain energy density function:

$$U = U(I_1, I_2, J) = C_{10}(I_1 - 3) + \frac{1}{D_1}(J - 1)^2 .$$

- Here I_1 , I_2 , and J are the three strain invariants, expressed in terms of the left Cauchy-Green tensor, \underline{B} :

$$I_1 = tr(\underline{B}), \quad I_2 = \frac{1}{2}(I_1^2 - tr(\underline{B} \cdot \underline{B})), \quad \underline{B} = \underline{F} \cdot \underline{F}^T, \quad J = det(\underline{F}).$$



- In actuality, we use the deviatoric invariants \bar{I}_1 and \bar{I}_2 (see Section 4.6.1 of the ABAQUS Theory Manual for more information).
 - The constitutive equation can be written directly in terms of the deformation gradient:

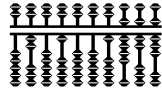
$$\sigma_{ij} = \frac{2}{J} C_{10} \left(\bar{B}_{ij} - \frac{1}{3} \delta_{ij} \bar{B}_{kk} \right) + \frac{2}{D_1} (J - 1) \delta_{ij}, \quad \bar{B}_{ij} = B_{ij} / J^{2/3}.$$

- We define the virtual rate of deformation as

$$\delta D_{ij} = \frac{1}{2} (\delta F_{im} F_{mj}^{-1} + F_{mi}^{-1} \delta F_{jm}).$$

- The Kirchhoff stress is defined through

$$\tau_{ij} = J \sigma_{ij}.$$



- The material Jacobian derives from the variation in Kirchhoff stress:

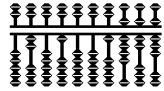
$$\delta \tau_{ij} = J C_{ijkl} \delta D_{kl} \quad ,$$

where C_{ijkl} are the components of the Jacobian. Using the Neo-Hookean model,

$$C_{ijkl} = \frac{2}{J} C_{10} \left(\frac{1}{2} (\delta_{ik} \bar{B}_{jl} + \bar{B}_{ik} \delta_{jl} + \delta_{il} \bar{B}_{jk} + \bar{B}_{il} \delta_{jk}) \right. \\ \left. - \frac{2}{3} \delta_{ij} \bar{B}_{kl} - \frac{2}{3} \bar{B}_{ij} \delta_{kl} + \frac{2}{9} \delta_{ij} \delta_{kl} \bar{B}_{mm} \right) + \frac{2}{D_1} (2J - 1) \delta_{ij} \delta_{kl} \quad .$$

- The expression is fairly complex, but it is straightforward to implement.
- For details of the derivation see Section 4.6.1 of the ABAQUS Theory Manual.

The appropriate coding is shown on the following pages.

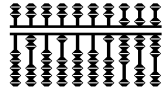


Coding for Neo-Hookean Hyperelasticity

```

C      LOCAL ARRAYS
C  -----
C      EELAS   - LOGARITHMIC ELASTIC STRAINS
C      EELASP  - PRINCIPAL ELASTIC STRAINS
C      BBAR    - DEVIATORIC RIGHT CAUCHY-GREEN TENSOR
C      BBARP   - PRINCIPAL VALUES OF BBAR
C      BBARN   - PRINCIPAL DIRECTION OF BBAR (AND EELAS)
C      DISTGR  - DEVIATORIC DEFORMATION GRADIENT (DISTORTION TENSOR)
C  -----
C
C      DIMENSION EELAS(6), EELASP(3), BBAR(6), BBARP(3), BBARN(3, 3),
1          DISTGR(3,3)
C
C      PARAMETER(ZERO=0.D0, ONE=1.D0, TWO=2.D0, THREE=3.D0, FOUR=4.D0,
1          SIX=6.D0)
C
C  -----
C      UMAT FOR COMPRESSIBLE NEO-HOOKEAN HYPERELASTICITY
C      CANNOT BE USED FOR PLANE STRESS
C  -----
C      PROPS(1) - E
C      PROPS(2) - NU

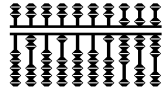
```

```

C -----
C
C   ELASTIC PROPERTIES
C
      EMOD=PROPS(1)
      ENU=PROPS(2)
      C10=EMOD/(FOUR*(ONE+ENU))
      D1=SIX*(ONE-TWO*ENU)/EMOD
C
C   JACOBIAN AND DISTORTION TENSOR
C
      DET=DFGRD1(1, 1)*DFGRD1(2, 2)*DFGRD1(3, 3)
1      -DFGRD1(1, 2)*DFGRD1(2, 1)*DFGRD1(3, 3)
      IF(NSHR.EQ.3) THEN
          DET=DET+DFGRD1(1, 2)*DFGRD1(2, 3)*DFGRD1(3, 1)
1          +DFGRD1(1, 3)*DFGRD1(3, 2)*DFGRD1(2, 1)
2          -DFGRD1(1, 3)*DFGRD1(3, 1)*DFGRD1(2, 2)
3          -DFGRD1(2, 3)*DFGRD1(3, 2)*DFGRD1(1, 1)
      END IF
      SCALE=DET**(-ONE/THREE)
      DO K1=1, 3
          DO K2=1, 3
              DISTGR(K2, K1)=SCALE*DFGRD1(K2, K1)
          END DO
      END DO

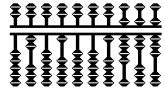
```



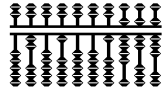
```

      END DO
C      CALCULATE DEVIATORIC LEFT CAUCHY-GREEN DEFORMATION TENSOR
C
      BBAR(1)=DISTGR(1, 1)**2+DISTGR(1, 2)**2+DISTGR(1, 3)**2
      BBAR(2)=DISTGR(2, 1)**2+DISTGR(2, 2)**2+DISTGR(2, 3)**2
      BBAR(3)=DISTGR(3, 3)**2+DISTGR(3, 1)**2+DISTGR(3, 2)**2
      BBAR(4)=DISTGR(1, 1)*DISTGR(2, 1)+DISTGR(1, 2)*DISTGR(2, 2)
1      +DISTGR(1, 3)*DISTGR(2, 3)
      IF(NSHR.EQ.3) THEN
          BBAR(5)=DISTGR(1, 1)*DISTGR(3, 1)+DISTGR(1, 2)*DISTGR(3, 2)
1      +DISTGR(1, 3)*DISTGR(3, 3)
          BBAR(6)=DISTGR(2, 1)*DISTGR(3, 1)+DISTGR(2, 2)*DISTGR(3, 2)
1      +DISTGR(2, 3)*DISTGR(3, 3)
      END IF
C
C      CALCULATE THE STRESS
C
      TRBBAR=(BBAR(1)+BBAR(2)+BBAR(3))/THREE
      EG=TWO*C10/DET
      EK=TWO/D1*(TWO*DET-ONE)
      PR=TWO/D1*(DET-ONE)
      DO K1=1,NDI
          STRESS(K1)=EG*(BBAR(K1)-TRBBAR)+PR
      END DO

```



```
      DO K1=NDI+1,NDI+NSHR
        STRESS(K1)=EG*BBAR(K1)
      END DO
C      CALCULATE THE STIFFNESS
C
      EG23=EG*TWO/THREE
      DDSDE(1, 1)= EG23*(BBAR(1)+TRBBAR)+EK
      DDSDE(2, 2)= EG23*(BBAR(2)+TRBBAR)+EK
      DDSDE(3, 3)= EG23*(BBAR(3)+TRBBAR)+EK
      DDSDE(1, 2)=-EG23*(BBAR(1)+BBAR(2)-TRBBAR)+EK
      DDSDE(1, 3)=-EG23*(BBAR(1)+BBAR(3)-TRBBAR)+EK
      DDSDE(2, 3)=-EG23*(BBAR(2)+BBAR(3)-TRBBAR)+EK
      DDSDE(1, 4)= EG23*BBAR(4)/TWO
      DDSDE(2, 4)= EG23*BBAR(4)/TWO
      DDSDE(3, 4)=-EG23*BBAR(4)
      DDSDE(4, 4)= EG*(BBAR(1)+BBAR(2))/TWO
      IF(NSHR.EQ.3) THEN
        DDSDE(1, 5)= EG23*BBAR(5)/TWO
        DDSDE(2, 5)=-EG23*BBAR(5)
        DDSDE(3, 5)= EG23*BBAR(5)/TWO
        DDSDE(1, 6)=-EG23*BBAR(6)
        DDSDE(2, 6)= EG23*BBAR(6)/TWO
        DDSDE(3, 6)= EG23*BBAR(6)/TWO
        DDSDE(5, 5)= EG*(BBAR(1)+BBAR(3))/TWO
```



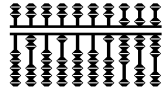
```

      DDSDE(6, 6) = EG*(BBAR(2)+BBAR(3))/TWO
      DDSDE(4, 5) = EG*BBAR(6)/TWO
      DDSDE(4, 6) = EG*BBAR(5)/TWO
      DDSDE(5, 6) = EG*BBAR(4)/TWO
    END IF
    DO K1=1, NTENS
      DO K2=1, K1-1
        DDSDE(K1, K2)=DDSDE(K2, K1)
      END DO
    END DO

C
C  CALCULATE LOGARITHMIC ELASTIC STRAINS (OPTIONAL)
C
      CALL SPRIND(BBAR, BBARP, BBARN, 1, NDI, NSHR)
      EELASP(1)=LOG(SQRT(BBARP(1))/SCALE)
      EELASP(2)=LOG(SQRT(BBARP(2))/SCALE)
      EELASP(3)=LOG(SQRT(BBARP(3))/SCALE)
      EELAS(1)=EELASP(1)*BBARN(1,1)**2+EELASP(2)*BBARN(2, 1)**2
1          +EELASP(3)*BBARN(3, 1)**2
      EELAS(2)=EELASP(1)*BBARN(1, 2)**2+EELASP(2)*BBARN(2, 2)**2
1          +EELASP(3)*BBARN(3, 2)**2
      EELAS(3)=EELASP(1)*BBARN(1, 3)**2+EELASP(2)*BBARN(2, 3)**2
1          +EELASP(3)*BBARN(3, 3)**2
      EELAS(4)=TWO*(EELASP(1)*BBARN(1, 1)*BBARN(1, 2)

```

Call to SPRIND



```

1          +EELASP(2)*BBARN(2, 1)*BBARN(2, 2)
2          +EELASP(3)*BBARN(3, 1)*BBARN(3, 2))
  IF(NSHR.EQ.3) THEN
    EELAS(5)=TWO*(EELASP(1)*BBARN(1, 1)*BBARN(1, 3)
1          +EELASP(2)*BBARN(2, 1)*BBARN(2, 3)
2          +EELASP(3)*BBARN(3, 1)*BBARN(3, 3))
    EELAS(6)=TWO*(EELASP(1)*BBARN(1, 2)*BBARN(1, 3)
1          +EELASP(2)*BBARN(2, 2)*BBARN(2, 3)
2          +EELASP(3)*BBARN(3, 2)*BBARN(3, 3))
  END IF
C
C  STORE ELASTIC STRAINS IN STATE VARIABLE ARRAY
C
  DO K1=1, NTENS
    STATEV(K1)=EELAS(K1)
  END DO
C
  RETURN
END

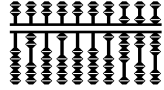
```

Remarks

- This **UMAT** yields exactly the same results as the ***HYPERELASTIC** option with $N = 1$ and $C_{01} = 0$.
- Note the use of the utility **SPRIND**.

```
CALL SPRIND(BBAR, BBARP, BBARN, 1, NDI, NSHR)
```

- Tensor **BBAR** consists of **NDI** direct components and **NSHR** shear components.
- **SPRIND** returns the principal values and direction cosines of the principal directions of **BBAR** in **BBARP** and **BBARN**, respectively.
- A value of 1 is used as the fourth argument to indicate that **BBAR** contains stresses. (A value of 2 is used for strains.)
- Hyperelastic materials are often implemented more easily in user subroutine **UHYPER**.



Example 4: Kinematic Hardening Plasticity

Governing Equations

- Elasticity:

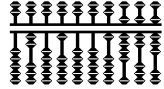
$$\sigma_{ij} = \lambda \delta_{ij} \epsilon_{kk}^{el} + 2\mu \epsilon_{ij}^{el},$$

or in a Jaumann (corotational) rate form:

$$\dot{\sigma}_{ij}^J = \lambda \delta_{ij} \dot{\epsilon}_{kk}^{el} + 2\mu \dot{\epsilon}_{ij}^{el}.$$

- The Jaumann rate equation is integrated in a corotational framework:

$$\Delta \sigma_{ij}^J = \lambda \delta_{ij} \Delta \epsilon_{kk}^{el} + 2\mu \Delta \epsilon_{ij}^{el}.$$



- Plasticity:
 - Yield function:

$$\sqrt{\frac{3}{2}}(S_{ij} - \alpha_{ij})(S_{ij} - \alpha_{ij}) - \sigma_y = 0 .$$

- Equivalent plastic strain rate:

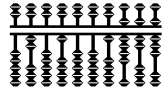
$$\dot{\epsilon}^{pl} = \sqrt{\frac{2}{3}} \dot{\epsilon}_{ij}^{pl} \dot{\epsilon}_{ij}^{pl} .$$

- Plastic flow law:

$$\dot{\epsilon}_{ij}^{pl} = \frac{3}{2}(S_{ij} - \alpha_{ij}) \dot{\epsilon}^{pl} / \sigma_y .$$

- Prager-Ziegler (linear) kinematic hardening:

$$\dot{\alpha}_{ij} = \frac{2}{3} h \dot{\epsilon}_{ij}^{pl} .$$



Integration Procedure

- We first calculate the equivalent stress based on purely elastic behavior (elastic predictor):

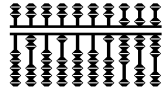
$$\bar{\sigma}^{pr} = \sqrt{\frac{3}{2}(S_{ij}^{pr} - \alpha_{ij}^o)(S_{ij}^{pr} - \alpha_{ij}^o)} , \quad S_{ij}^{pr} = S_{ij}^o + 2\mu\Delta e_{ij}.$$

- Plastic flow occurs if the elastic predictor is larger than the yield stress. The backward Euler method is used to integrate the equations:

$$\Delta \epsilon_{ij}^{pl} = \frac{3}{2}(S_{ij}^{pr} - \alpha_{ij}^o)\Delta \bar{\epsilon}^{pl} / \bar{\sigma}^{pr}.$$

- After some manipulation we obtain a closed form expression for the equivalent plastic strain increment:

$$\Delta \bar{\epsilon}^{pl} = (\bar{\sigma}^{pr} - \sigma_y) / (h + 3\mu) .$$



- This leads to the following update equations for the shift tensor, the stress, and the plastic strain:

$$\Delta\alpha_{ij} = \eta_{ij}h\Delta\bar{\epsilon}^{pl}, \quad \Delta\epsilon_{ij}^{pl} = \frac{3}{2}\eta_{ij}\Delta\bar{\epsilon}^{pl}$$

$$\sigma_{ij} = \alpha_{ij}^o + \Delta\alpha_{ij} + \eta_{ij}\sigma_y + \frac{1}{3}\delta_{ij}\sigma_{kk}^{pr}, \quad \eta_{ij} = (S_{ij}^{pr} - \alpha_{ij}^o)/\bar{\sigma}^{pr}.$$

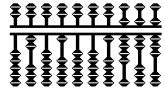
- In addition, you can readily obtain the consistent Jacobian:

$$\Delta\dot{\sigma}_{ij} = \lambda^*\delta_{ij}\Delta\dot{\epsilon}_{kk} + 2\mu^*\Delta\dot{\epsilon}_{ij} + \left(\frac{h}{1 + h/3\mu} - 3\mu^*\right)\eta_{ij}\eta_{kl}\Delta\dot{\epsilon}_{kl}$$

$$\mu^* = \mu(\sigma_y + h\Delta\bar{\epsilon}^{pl})/\bar{\sigma}^{pr}, \quad \lambda^* = k - \frac{2}{3}\mu^*.$$

- The integration procedure for kinematic hardening is described in Section 21 of the ABAQUS/Explicit User's Manual.

The appropriate coding is shown on the following pages.

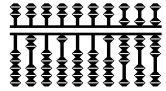


Coding for Kinematic Hardening Plasticity

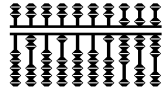
```

C      LOCAL ARRAYS
C      -----
C      EELAS   - ELASTIC STRAINS
C      EPLAS   - PLASTIC STRAINS
C      ALPHA   - SHIFT TENSOR
C      FLOW    - PLASTIC FLOW DIRECTIONS
C      OLDS    - STRESS AT START OF INCREMENT
C      OLDPL   - PLASTIC STRAINS AT START OF INCREMENT
C
C      DIMENSION EELAS(6), EPLAS(6), ALPHA(6), FLOW(6), OLDS(6), OLDPL(6)
C
C      PARAMETER(ZERO=0.D0, ONE=1.D0, TWO=2.D0, THREE=3.D0, SIX=6.D0,
1          ENUMAX=.4999D0, TOLER=1.0D-6)
C
C      -----
C      UMAT FOR ISOTROPIC ELASTICITY AND MISES PLASTICITY
C      WITH KINEMATIC HARDENING - CANNOT BE USED FOR PLANE STRESS
C      -----
C      PROPS(1) - E
C      PROPS(2) - NU
C      PROPS(3) - SYIELD
C      PROPS(4) - HARD

```



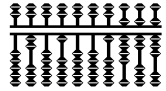
```
C -----
C
C   ELASTIC PROPERTIES
C
      EMOD=PROPS(1)
      ENU=MIN(PROPS(2), ENUMAX)
      EBULK3=EMOD/(ONE-TWO*ENU)
      EG2=EMOD/(ONE+ENU)
      EG=EG2/TWO
      EG3=THREE*EG
      ELAM=(EBULK3-EG2)/THREE
C
C   ELASTIC STIFFNESS
C
      DO K1=1, NDI
        DO K2=1, NDI
          DDSDE(K2, K1)=ELAM
        END DO
        DDSDE(K1, K1)=EG2+ELAM
      END DO
      DO K1=NDI+1, NTENS
        DDSDE(K1, K1)=EG
      END DO
```



```
C
C   RECOVER ELASTIC STRAIN, PLASTIC STRAIN AND SHIFT TENSOR AND ROTATE
C   NOTE: USE CODE 1 FOR (TENSOR) STRESS, CODE 2 FOR (ENGINEERING) STRAIN
C
      CALL ROTSIG(STATEV(          1), DROT, EELAS, 2, NDI, NSHR)
      CALL ROTSIG(STATEV( NTENS+1), DROT, EPLAS, 2, NDI, NSHR)
      CALL ROTSIG(STATEV(2*NTENS+1), DROT, ALPHA, 1, NDI, NSHR)

C
C   SAVE STRESS AND PLASTIC STRAINS AND
C   CALCULATE PREDICTOR STRESS AND ELASTIC STRAIN
C
      DO K1=1, NTENS
        OLDS(K1)=STRESS(K1)
        OLDPL(K1)=EPLAS(K1)
        EELAS(K1)=EELAS(K1)+DSTRAN(K1)
        DO K2=1, NTENS
          STRESS(K2)=STRESS(K2)+DDSDDE(K2, K1)*DSTRAN(K1)
        END DO
      END DO
```

Calls to ROTSIG

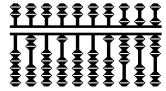


```
C
C      CALCULATE EQUIVALENT VON MISES STRESS
C
      SMISES=(STRESS(1)-ALPHA(1)-STRESS(2)+ALPHA(2))**2
1      + (STRESS(2)-ALPHA(2)-STRESS(3)+ALPHA(3))**2
2      + (STRESS(3)-ALPHA(3)-STRESS(1)+ALPHA(1))**2
      DO K1=NDI+1,NTENS
          SMISES=SMISES+SIX*(STRESS(K1)-ALPHA(K1))**2
      END DO
      SMISES=SQRT(SMISES/TWO)

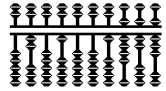
C
C      GET YIELD STRESS AND HARDENING MODULUS
C
      SYIELD=PROPS(3)
      HARD=PROPS(4)

C
C      DETERMINE IF ACTIVELY YIELDING
C
      IF(SMISES.GT.(ONE+TOLER)*SYIELD) THEN

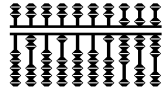
C
C      ACTIVELY YIELDING
```



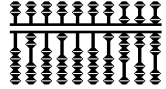
```
C      SEPARATE THE HYDROSTATIC FROM THE DEVIATORIC STRESS
C      CALCULATE THE FLOW DIRECTION
C
      SHYDRO= (STRESS (1) +STRESS (2) +STRESS (3) ) /THREE
      DO K1=1,NDI
        FLOW(K1) = (STRESS (K1) -ALPHA (K1) -SHYDRO) /SMISES
      END DO
      DO K1=NDI+1,NTENS
        FLOW(K1) = (STRESS (K1) -ALPHA (K1) ) /SMISES
      END DO
C
C      SOLVE FOR EQUIVALENT PLASTIC STRAIN INCREMENT
C
      DEQPL= (SMISES-SYIELD) / (EG3+HARD)
```



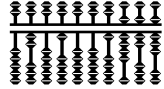
```
C
C      UPDATE SHIFT TENSOR, ELASTIC AND PLASTIC STRAINS AND STRESS
C
      DO K1=1,NDI
        ALPHA(K1)=ALPHA(K1)+HARD*FLOW(K1)*DEQPL
        EPLAS(K1)=EPLAS(K1)+THREE/TWO*FLOW(K1)*DEQPL
        EELAS(K1)=EELAS(K1)-THREE/TWO*FLOW(K1)*DEQPL
        STRESS(K1)=ALPHA(K1)+FLOW(K1)*SYIELD+SHYDRO
      END DO
      DO K1=NDI+1,NTENS
        ALPHA(K1)=ALPHA(K1)+HARD*FLOW(K1)*DEQPL
        EPLAS(K1)=EPLAS(K1)+THREE*FLOW(K1)*DEQPL
        EELAS(K1)=EELAS(K1)-THREE*FLOW(K1)*DEQPL
        STRESS(K1)=ALPHA(K1)+FLOW(K1)*SYIELD
      END DO
C      CALCULATE PLASTIC DISSIPATION
C
      SPD=ZERO
      DO K1=1,NTENS
        SPD=SPD+(STRESS(K1)+OLDS(K1))*(EPLAS(K1)-OLDPL(K1))/TWO
      END DO
```

```
C
C   FORMULATE THE JACOBIAN (MATERIAL TANGENT)
C   FIRST CALCULATE EFFECTIVE MODULI
C
      EFFG=EG*(SYIELD+HARD*DEQPL)/SMISES
      EFFG2=TWO*EFFG
      EFFG3=THREE*EFFG
      EFFLAM=(EBULK3-EFFG2)/THREE
      EFFHRD=EG3*HARD/(EG3+HARD)-EFFG3
      DO K1=1, NDI
        DO K2=1, NDI
          DDSDE(K2, K1)=EFFLAM
        END DO
        DDSDE(K1, K1)=EFFG2+EFFLAM
      END DO
      DO K1=NDI+1, NTENS
        DDSDE(K1, K1)=EFFG
      END DO
      DO K1=1, NTENS
        DO K2=1, NTENS
          DDSDE(K2, K1)=DDSDE(K2, K1)+EFFHRD*FLOW(K2)*FLOW(K1)
        END DO
      END DO
    ENDIF
```

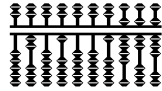


```
C
C   STORE ELASTIC STRAINS, PLASTIC STRAINS AND SHIFT TENSOR
C   IN STATE VARIABLE ARRAY
C
      DO K1=1,NTENS
        STATEV(K1)=EELAS(K1)
        STATEV(K1+NTENS)=EPLAS(K1)
        STATEV(K1+2*NTENS)=ALPHA(K1)
      END DO
C
      RETURN
      END
```



Remarks

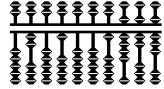
- This **UMAT** yields exactly the same results as the ***PLASTIC** option with **KINEMATIC** hardening.
 - This is also true for large-strain calculations. The necessary rotations of stress and strain are taken care of by **ABAQUS**.



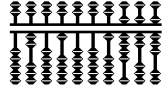
- Rotation of the shift tensor and the elastic and plastic strains is accomplished by the calls to **ROTSIG**. The call

```
CALL ROTSIG(STATEV(1), DROT, EELAS, 2, NDI, NSHR)
```

applies the incremental rotation, **DROT**, to **STATEV** and stores the result in **ELAS**.
 - **STATEV** consists of **NDI** direct components and **NSHR** shear components.
 - A value of 1 is used as the fourth argument to indicate that the transformed array contains tensor shear components such as α_{ij} . A value of 2 indicates that the array contains engineering shear components, such as ϵ_{ij}^{pl} .
- The rotation should be applied prior to the integration procedure.



- The routine is written for linear hardening because the classical Prager-Ziegler theory is limited to this case.
 - More complex nonlinear kinematic hardening models are much more difficult to integrate.
 - However, once a suitable integration procedure is obtained, the implementation in **UMAT** is straightforward and follows the examples discussed here.



Example 5: Isotropic Hardening Plasticity

Governing Equations

- Elasticity:

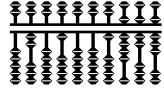
$$\sigma_{ij} = \lambda \delta_{ij} \epsilon_{kk}^{el} + 2\mu \epsilon_{ij}^{el},$$

or in a Jaumann (corotational) rate form:

$$\dot{\sigma}_{ij}^J = \lambda \delta_{ij} \dot{\epsilon}_{kk}^{el} + 2\mu \dot{\epsilon}_{ij}^{el}.$$

- The Jaumann rate equation is integrated in a corotational framework:

$$\Delta \sigma_{ij}^J = \lambda \delta_{ij} \Delta \epsilon_{kk}^{el} + 2\mu \Delta \epsilon_{ij}^{el}.$$



- Plasticity:
 - Yield function:

$$\sqrt{\frac{3}{2}} S_{ij} S_{ij} - \sigma_y(\bar{\epsilon}^{pl}) = 0, \quad S_{ij} = \sigma_{ij} - \frac{1}{3} \delta_{ij} \sigma_{kk}.$$

- Equivalent plastic strain:

$$\bar{\epsilon}^{pl} = \int_0^t \dot{\bar{\epsilon}}^{pl} dt, \quad \dot{\bar{\epsilon}}^{pl} = \sqrt{\frac{2}{3}} \dot{\epsilon}_{ij}^{pl} \dot{\epsilon}_{ij}^{pl}.$$

- Plastic flow law:

$$\dot{\epsilon}_{ij}^{pl} = \frac{3 S_{ij}}{2 \sigma_y} \dot{\bar{\epsilon}}^{pl}.$$

Integration Procedure

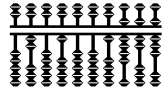
- We first calculate the von Mises stress based on purely elastic behavior (elastic predictor):

$$\bar{\sigma}^{pr} = \sqrt{\frac{3}{2} S_{ij}^{pr} S_{ij}^{pr}}, \quad S_{ij}^{pr} = S_{ij}^o + 2\mu \Delta e_{ij}.$$

- If the elastic predictor is larger than the current yield stress, plastic flow occurs. The backward Euler method is used to integrate the equations.
 - After some manipulation we can reduce the problem to a single equation in terms of the incremental equivalent plastic strain:

$$\bar{\sigma}^{pr} - 3\mu \Delta \bar{\epsilon}^{pl} = \sigma_y(\bar{\epsilon}^{pl}).$$

- This equation is solved with Newton's method.



- After the equation is solved, the following update equations for the stress and the plastic strain can be used:

$$\sigma_{ij} = \eta_{ij}\sigma_y + \frac{1}{3}\delta_{ij}\sigma_{kk}^{pr}, \quad \Delta\epsilon_{ij}^{pl} = \frac{3}{2}\eta_{ij}\Delta\bar{\epsilon}^{pl}$$

$$\eta_{ij} = S_{ij}^{pr} / \bar{\sigma}^{pr}.$$

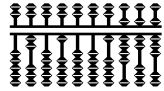
- In addition, you can readily obtain the consistent Jacobian:

$$\Delta\dot{\sigma}_{ij} = \lambda^*\delta_{ij}\Delta\dot{\epsilon}_{kk} + 2\mu^*\Delta\dot{\epsilon}_{ij} + \left(\frac{h}{1 + h/3\mu} - 3\mu^*\right)\eta_{ij}\eta_{kl}\Delta\dot{\epsilon}_{kl}$$

$$\mu^* = \mu\sigma_y/\bar{\sigma}^{pr}, \quad \lambda^* = k - \frac{2}{3}\mu^*, \quad h = d\sigma_y/d\bar{\epsilon}^{pl}.$$

- A detailed discussion about the isotropic plasticity integration algorithm can be found in Section 4.2.2 of the ABAQUS Theory Manual.

The appropriate coding is shown on the following pages.

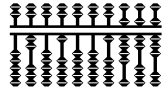


Coding for Isotropic Mises Plasticity

```

C      LOCAL ARRAYS
C  -----
C      EELAS   - ELASTIC STRAINS
C      EPLAS   - PLASTIC STRAINS
C      FLOW    - DIRECTION OF PLASTIC FLOW
C  -----
C
C      DIMENSION EELAS(6),EPLAS(6),FLOW(6), HARD(3)
C
C      PARAMETER(ZERO=0.D0, ONE=1.D0, TWO=2.D0, THREE=3.D0, SIX=6.D0,
1          ENUMAX=.4999D0, NEWTON=10, TOLER=1.0D-6)
C
C  -----
C      UMAT FOR ISOTROPIC ELASTICITY AND ISOTROPIC MISES PLASTICITY
C      CANNOT BE USED FOR PLANE STRESS
C  -----
C      PROPS(1) - E
C      PROPS(2) - NU
C      PROPS(3..) - SYIELD AN HARDENING DATA
C      CALLS UHARD FOR CURVE OF YIELD STRESS VS. PLASTIC STRAIN
C  -----

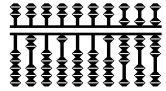
```



```
C
C      ELASTIC PROPERTIES
C
      EMOD=PROPS(1)
      ENU=MIN(PROPS(2), ENUMAX)
      EBULK3=EMOD/(ONE-TWO*ENU)
      EG2=EMOD/(ONE+ENU)
      EG=EG2/TWO
      EG3=THREE*EG
      ELAM=(EBULK3-EG2)/THREE

C
C      ELASTIC STIFFNESS
C

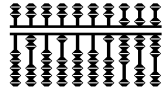
      DO K1=1, NDI
        DO K2=1, NDI
          DDSDE(K2, K1)=ELAM
        END DO
        DDSDE(K1, K1)=EG2+ELAM
      END DO
      DO K1=NDI+1, NTENS
        DDSDE(K1, K1)=EG
      END DO
```



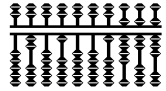
```

C   RECOVER ELASTIC AND PLASTIC STRAINS AND ROTATE FORWARD
C   ALSO RECOVER EQUIVALENT PLASTIC STRAIN
C
      CALL ROTSIG(STATEV(      1), DROT, EELAS, 2, NDI, NSHR)
      CALL ROTSIG(STATEV(NTENS+1), DROT, EPLAS, 2, NDI, NSHR)
      EQPLAS=STATEV(1+2*NTENS)
C
C   CALCULATE PREDICTOR STRESS AND ELASTIC STRAIN
C
      DO K1=1, NTENS
        DO K2=1, NTENS
          STRESS(K2)=STRESS(K2)+DDSDDE(K2, K1)*DSTRAN(K1)
        END DO
        EELAS(K1)=EELAS(K1)+DSTRAN(K1)
      END DO
C
C   CALCULATE EQUIVALENT VON MISES STRESS
C
      SMISES=(STRESS(1)-STRESS(2))**2+(STRESS(2)-STRESS(3))**2
1          + (STRESS(3)-STRESS(1))**2
      DO K1=NDI+1,NTENS
        SMISES=SMISES+SIX*STRESS(K1)**2
      END DO
      SMISES=SQRT(SMISES/TWO)

```



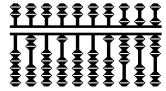
```
C
C   GET YIELD STRESS FROM THE SPECIFIED HARDENING CURVE
C
      NVALUE=NPROPS/2-1
      CALL UHARD(SYIEL0, HARD, EQPLAS, EQPLASRT, TIME, DTIME, TEMP,
1         DTEMP, NOEL, NPT, LAYER, KSPT, KSTEP, KINC, CMNAME, NSTATV,
2         STATEV, NUMFIELDV, PREDEF, DPRED, NVALUE, PROPS(3))
C
C   DETERMINE IF ACTIVELY YIELDING
C
      IF (SMISES.GT.(ONE+TOLER)*SYIEL0) THEN
C
C   ACTIVELY YIELDING
C   SEPARATE THE HYDROSTATIC FROM THE DEVIATORIC STRESS
C   CALCULATE THE FLOW DIRECTION
C
      SHYDRO=(STRESS(1)+STRESS(2)+STRESS(3))/THREE
      DO K1=1,NDI
          FLOW(K1)=(STRESS(K1)-SHYDRO)/SMISES
      END DO
      DO K1=NDI+1, NTENS
          FLOW(K1)=STRESS(K1)/SMISES
      END DO
```



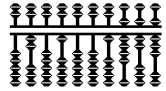
```

C
C      SOLVE FOR EQUIVALENT VON MISES STRESS
C      AND EQUIVALENT PLASTIC STRAIN INCREMENT USING NEWTON ITERATION
C
      SYIELD=SYIEL0
      DEQPL=ZERO
      DO KEWTON=1, NEWTON
        RHS=SMISES-EG3*DEQPL-SYIELD
        DEQPL=DEQPL+RHS/(EG3+HARD(1))
        CALL UHARD(SYIELD,HARD,EQPLAS+DEQPL,EQPLASRT,TIME,DTIME,TEMP,
1        DTEMP,NOEL,NPT,LAYER,KSPT,KSTEP,KINC,CMNAME,NSTATV,
2        STATEV,NUMFIELDV,PREDEF,DPRED,NVALUE,PROPS(3))
        IF(ABS(RHS).LT.TOLER*SYIEL0) GOTO 10
      END DO
C
C      WRITE WARNING MESSAGE TO THE .MSG FILE
C
      WRITE(7,2) NEWTON
2      FORMAT(/,30X,'***WARNING - PLASTICITY ALGORITHM DID NOT ',
1      'CONVERGE AFTER ',I3,' ITERATIONS')
10  CONTINUE

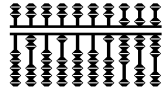
```



```
C
C      UPDATE STRESS, ELASTIC AND PLASTIC STRAINS AND
C      EQUIVALENT PLASTIC STRAIN
C
      DO K1=1,NDI
        STRESS(K1)=FLOW(K1)*SYIELD+SHYDRO
        EPLAS(K1)=EPLAS(K1)+THREE/TWO*FLOW(K1)*DEQPL
        EELAS(K1)=EELAS(K1)-THREE/TWO*FLOW(K1)*DEQPL
      END DO
      DO K1=NDI+1,NTENS
        STRESS(K1)=FLOW(K1)*SYIELD
        EPLAS(K1)=EPLAS(K1)+THREE*FLOW(K1)*DEQPL
        EELAS(K1)=EELAS(K1)-THREE*FLOW(K1)*DEQPL
      END DO
      EQPLAS=EQPLAS+DEQPL
C
C      CALCULATE PLASTIC DISSIPATION
C
      SPD=DEQPL*(SYIEL0+SYIELD)/TWO
```



```
C
C   FORMULATE THE JACOBIAN (MATERIAL TANGENT)
C   FIRST CALCULATE EFFECTIVE MODULI
C
      EFFG=EG*SYIELD/SMISES
      EFFG2=TWO*EFFG
      EFFG3=THREE/TWO*EFFG2
      EFFLAM=(EBULK3-EFFG2)/THREE
      EFFHRD=EG3*HARD(1)/(EG3+HARD(1))-EFFG3
      DO K1=1, NDI
        DO K2=1, NDI
          DDSDE(K2, K1)=EFFLAM
        END DO
        DDSDE(K1, K1)=EFFG2+EFFLAM
      END DO
      DO K1=NDI+1, NTENS
        DDSDE(K1, K1)=EFFG
      END DO
      DO K1=1, NTENS
        DO K2=1, NTENS
          DDSDE(K2, K1)=DDSDE(K2, K1)+EFFHRD*FLOW(K2)*FLOW(K1)
        END DO
      END DO
    ENDIF
```

```

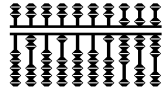
C
C   STORE ELASTIC AND (EQUIVALENT) PLASTIC STRAINS
C   IN STATE VARIABLE ARRAY
C
      DO K1=1, NTENS
        STATEV(K1)=EELAS(K1)
        STATEV(K1+NTENS)=EPLAS(K1)
      END DO
      STATEV(1+2*NTENS)=EQPLAS
C
      RETURN
      END

      SUBROUTINE UHARD(SYIELD,HARD,EQPLAS,EQPLASRT,TIME,DTIME,TEMP,
1      DTEMP,NOEL,NPT,LAYER,KSPT,KSTEP,KINC,
2      CMNAME,NSTATV,STATEV,NUMFIELDV,
3      PREDEF,DPRED,NVALUE,TABLE)

      INCLUDE 'ABA_PARAM.INC'

      CHARACTER*80 CMNAME
      DIMENSION HARD(3),STATEV(NSTATV),TIME(*),
1      PREDEF(NUMFIELDV),DPRED(*)

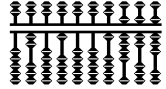
```



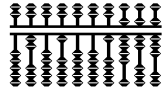
```

C
    DIMENSION TABLE(2, NVALUE)
C
    PARAMETER(ZERO=0.D0)
C
    SET YIELD STRESS TO LAST VALUE OF TABLE, HARDENING TO ZERO
C
    SYIELD=TABLE(1, NVALUE)
    HARD(1)=ZERO
C
    IF MORE THAN ONE ENTRY, SEARCH TABLE
C
    IF(NVALUE.GT.1) THEN
        DO K1=1, NVALUE-1
            EQPL1=TABLE(2,K1+1)
            IF(EQPLAS.LT.EQPL1) THEN
                EQPL0=TABLE(2, K1)
                IF(EQPL1.LE.EQPL0) THEN
                    WRITE(7, 1)
1          FORMAT(//, 30X, '***ERROR - PLASTIC STRAIN MUST BE ` ,
1          'ENTERED IN ASCENDING ORDER')
                    CALL XIT
                ENDIF
            ENDIF
        END DO
    END IF

```

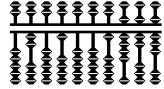


```
C
C      CURRENT YIELD STRESS AND HARDENING
C
      DEQPL=EQPL1-EQPL0
      SYIEL0=TABLE(1, K1)
      SYIEL1=TABLE(1, K1+1)
      DSYIEL=SYIEL1-SYIEL0
      HARD(1)=DSYIEL/DEQPL
      SYIELD=SYIEL0+(EQPLAS-EQPL0)*HARD(1)
      GOTO 10
    ENDIF
  END DO
10  CONTINUE
    ENDIF
    RETURN
  END
```

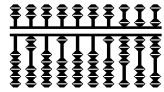


Remarks

- This **UMAT** yields exactly the same results as the ***PLASTIC** option with **ISOTROPIC** hardening.
 - This result is also true for large-strain calculations. The necessary rotations of stress and strain are taken care of by **ABAQUS**.
 - The rotation of elastic and plastic strain, prior to integration, is accomplished by the calls to **ROTSIG**.



- The routine calls user subroutine **UHARD** to recover a piecewise linear hardening curve.
 - It is straightforward to replace the piecewise linear curve by an analytic description.
 - A local Newton iteration is used to determine the current yield stress and hardening modulus.
 - If the data are not given in ascending order of strain, the routine **XIT** is called, which closes all files and terminates execution.



VUMAT Interface

- These input lines act as the interface to a **VUMAT** in which kinematic hardening plasticity is defined.

```
*MATERIAL, NAME=KINPLAS
```

```
*USER MATERIAL, CONSTANTS=4
```

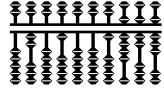
```
30.E6, 0.3, 30.E3, 40.E3
```

```
*DEPVAR
```

```
5
```

```
*INITIAL CONDITIONS, TYPE=SOLUTION
```

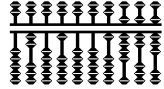
Data line to specify initial solution-dependent variables



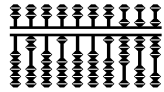
- The input lines are identical to those for the **UMAT** interface.
 - The user subroutine must be kept in a separate file, and is invoked with the ABAQUS execution procedure, as follows:

abaqus job=... user=....

- The user subroutine must be invoked in a restarted analysis because user subroutines are not saved in the restart file.

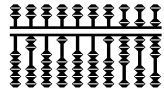


- Additional notes:
 - Solution-dependent state variables can be output with identifiers SDV1, SDV2, etc. Contour, path, and X–Y plots of SDVs can be plotted in ABAQUS/Viewer.
 - Include only a single **VUMAT** subroutine in the analysis. If more than one material must be defined, test on the material name in the **VUMAT** routine and branch.



- The **VUMAT** subroutine header is shown below:

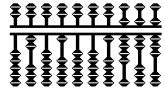
```
      SUBROUTINE VUMAT(  
C Read only -  
      1  NBLOCK, NDIR, NSHR, NSTATEV, NFIELDV, NPROPS, LANEAL,  
      2  STEPTIME, TOTALTIME, DT, CMNAME, COORDMP, CHARLENGTH,  
      3  PROPS, DENSITY, STRAININC, RELSPININC,  
      4  TEMPOLD, STRETCHOLD, DEFGRADOLD, FIELDOLD,  
      5  STRESSOLD, STATEOLD, ENERINTERNOLD, ENERINELASOLD,  
      6  TEMPNEW, STRETCHNEW, DEFGRADNEW, FIELDNEW,  
C Write only -  
      7  STRESSNEW, STATENew, ENERINTERNNEW, ENERINELASNEW)  
C  
      INCLUDE 'VABA_PARAM.INC'  
C
```



```
DIMENSION PROPS(NPROPS), DENSITY(NBLOCK), COORDMP(NBLOCK),  
1  CHARLENGTH(NBLOCK), STRAININC(NBLOCK, NDIR+NSHR),  
2  RELSPININC(NBLOCK, NSHR), TEMPOLD(NBLOCK),  
3  STRETCHOLD(NBLOCK, NDIR+NSHR), DEFGRADOLD(NBLOCK, NDIR+NSHR+NSHR),  
4  FIELDOLD(NBLOCK, NFIELDV), STRESSOLD(NBLOCK, NDIR+NSHR),  
5  STATEOLD(NBLOCK, NSTATEV), ENERINTERNOLD(NBLOCK),  
6  ENERINELASOLD(NBLOCK), TEMPNEW(NBLOCK),  
7  STRETCHNEW(NBLOCK, NDIR+NSHR), DEFGRADNEW(NBLOCK, NDIR+NSHR+NSHR),  
8  FIELDNEW(NBLOCK, NFIELDV), STRESSNEW(NBLOCK, NDIR+NSHR),  
9  STATENEW(NBLOCK, NSTATEV), ENERINTERNNEW(NBLOCK),  
1  ENERINELASNEW(NBLOCK)
```

C

```
CHARACTER*8 CMNAME
```



VUMAT Variables

- The following quantities are available in **VUMAT**, but they cannot be redefined:
 - Stress, stretch, and SDVs at the start of the increment
 - Relative rotation vector and deformation gradient at the start and end of an increment and strain increment
 - Total and incremental values of time, temperature, and user-defined field variables at the start and end of an increment
 - Material constants, density, material point position, and a characteristic element length
 - Internal and dissipated energies at the beginning of the increment
 - Number of material points to be processed in a call to the routine (**NBLOCK**)

- A flag indicating whether the routine is being called during an annealing process
- The following quantities must be defined:
 - Stress and SDVs at the end of an increment
- The following variables may be defined:
 - Internal and dissipated energies at the end of the increment

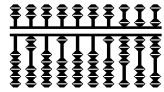
Many of these variables are equivalent or similar to those in **UMAT**.

Complete descriptions of all parameters are provided in the **VUMAT** section in Chapter 21 of the ABAQUS/Explicit User's Manual.

Comparison of VUMAT and UMAT Interfaces

There are a number of significant differences between the **UMAT** and **VUMAT** interfaces.

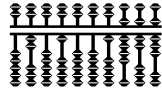
- **VUMAT** uses a two-state architecture: the initial values are in the **OLD** arrays, the new values must be put in the **NEW** arrays.
- The **VUMAT** interface is written to take advantage of vector processing.
- The material Jacobian does not need to be defined.
- No information is provided about element numbers.
- The time increment cannot be redefined.
- Utility routines are not available because they would prevent vectorization.



- The header is usually followed by dimensioning of local arrays. It is good practice to define constants via parameters and to include comments.

```
      PARAMETER( ZERO = 0.D0, ONE = 1.D0, TWO = 2.D0, THREE = 3.D0,  
1      THIRD = 1.D0/3.D0, HALF = .5D0, TWO_THIRDS = 2.D0/3.D0,  
2      THREE_HALFS = 1.5D0 )  
C J2 Mises Plasticity with kinematic hardening for plane strain case.  
C The state variables are stored as:  
C      STATE(*, 1) = back stress component 11  
C      STATE(*, 2) = back stress component 22  
C      STATE(*, 3) = back stress component 33  
C      STATE(*, 4) = back stress component 12  
C      STATE(*, 5) = equivalent plastic strain
```

- The **PARAMETER** assignments yield accurate floating point constant definitions on any platform.



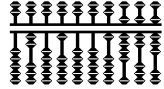
VUMAT Conventions

- Stresses and strains are stored as vectors.
 - For plane stress elements: σ_{11} , σ_{22} , σ_{12} .
 - For plane strain and axisymmetric elements: σ_{11} , σ_{22} , σ_{33} , σ_{12} .
 - For three-dimensional elements: σ_{11} , σ_{22} , σ_{33} , σ_{12} , σ_{23} , σ_{31} .

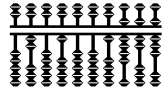
For three-dimensional elements, this storage scheme is inconsistent with that for ABAQUS/Standard.

- The shear strain is stored as tensor shear strains:

$$\epsilon_{12} = \frac{1}{2}\gamma_{12}.$$



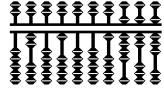
- The deformation gradient is stored similar to the way in which symmetric tensors are stored.
 - For plane stress elements: $F_{11}, F_{22}, F_{12}, F_{21}$.
 - For plane strain and axisymmetric elements: $F_{11}, F_{22}, F_{33}, F_{12}, F_{21}$.
 - For three-dimensional elements:
 $F_{11}, F_{22}, F_{33}, F_{12}, F_{23}, F_{31}, F_{21}, F_{32}, F_{13}$.



VUMAT Formulation Aspects

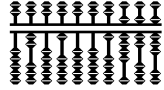
Vectorized Interface

- In **VUMAT** the data are passed in and out in large blocks (dimension **NBLOCK**). **NBLOCK** typically is equal to 64 or 128.
 - Each entry in an array of length **NBLOCK** corresponds to a single material point. All material points in the same block have the same material name and belong to the same element type.
- This structure allows vectorization of the routine.
 - A vectorized **VUMAT** should make sure that all operations are done in vector mode with **NBLOCK** the vector length.
- In vectorized code, branching inside loops should be avoided.
 - Element type-based branching should be outside the **NBLOCK** loop.



Corotational Formulation

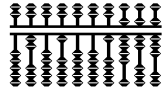
- The constitutive equation is formulated in a corotational framework, based on the Jaumann stress rate.
 - The strain increment is obtained with Hughes-Winget.
 - Other measures can be obtained from the deformation gradient.
- The user must define the Cauchy stress: this stress reappears during the next increment as the “old” stress.
- There is no need to rotate tensor state variables.



Example 6: VUMAT for Kinematic Hardening

The governing equations and integration procedure are the same as in **Example 4: Kinematic Hardening Plasticity** (p. L6.54).

The Jacobian is not required.



Coding for Kinematic Hardening Plasticity VUMAT

C

E = PROPS(1)

XNU = PROPS(2)

YIELD = PROPS(3)

HARD = PROPS(4)

C

C ELASTIC CONSTANTS

C

TWOMU = E / (ONE + XNU)

THREMU = THREE_HALFS * TWOMU

SIXMU = THREE * TWOMU

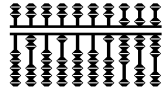
ALAMDA = TWOMU * (E - TWOMU) / (SIXMU - TWO * E)

TERM = ONE / (TWOMU * (ONE + HARD/THREMU))

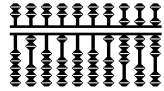
CON1 = SQRT(TWO_THIRDS)

C

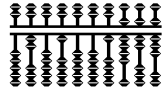
C



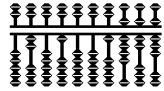
```
C
C If stepTime equals to zero, assume the material pure elastic and use
C initial elastic modulus
C
      IF( STEPTIME .EQ. ZERO ) THEN
C
      DO I = 1,NBLOCK
C
C   Trial Stress
          TRACE = STRAININC (I, 1) + STRAININC (I, 2) + STRAININC (I, 3)
          STRESSNEW(I, 1)=STRESSOLD(I, 1) + ALAMDA*TRACE
1      +          TWOMU*STRAININC(I,1)
          STRESSNEW(I, 2)=STRESSOLD(I, 2) + ALAMDA*TRACE
1      +          TWOMU*STRAININC(I, 2)
          STRESSNEW(I, 3)=STRESSOLD(I, 3) + ALAMDA*TRACE
1      +          TWOMU*STRAININC(I,3)
          STRESSNEW(I, 4)=STRESSOLD(I, 4)
1      +          TWOMU*STRAININC(I, 4)
          END DO
C
      ELSE
```



```
C
C   PLASTICITY CALCULATIONS IN BLOCK FORM
C
      DO I = 1, NBLOCK
C   Elastic predictor stress
          TRACE = STRAININC(I, 1) + STRAININC(I, 2) + STRAININC(I, 3)
          SIG1= STRESSOLD(I, 1) + ALAMDA*TRACE + TWOMU*STRAININC(I, 1)
          SIG2= STRESSOLD(I, 2) + ALAMDA*TRACE + TWOMU*STRAININC(I, 2)
          SIG3= STRESSOLD(I, 3) + ALAMDA*TRACE + TWOMU*STRAININC(I, 3)
          SIG4= STRESSOLD(I, 4)                + TWOMU*STRAININC(I, 4)
C   Elastic predictor stress measured from the back stress
          S1 = SIG1 - STATEOLD(I, 1)
          S2 = SIG2 - STATEOLD(I, 2)
          S3 = SIG3 - STATEOLD(I, 3)
          S4 = SIG4 - STATEOLD(I, 4)
C   Deviatoric part of predictor stress measured from the back stress
          SMEAN = THIRD * ( S1 + S2 + S3 )
          DS1 = S1 - SMEAN
          DS2 = S2 - SMEAN
          DS3 = S3 - SMEAN
C   Magnitude of the deviatoric predictor stress difference
          DSMAG = SQRT( DS1**2 + DS2**2 + DS3**2 + TWO*S4**2 )
```



```
C Check for yield by determining the factor for plasticity, zero for
C elastic, one for yield
      RADIUS = CON1 * YIELD
      FACYLD = ZERO
      IF( DSMAG - RADIUS .GE. ZERO ) FACYLD = ONE
C Add a protective addition factor to prevent a divide by zero when DSMAG
C is zero. If DSMAG is zero, we will not have exceeded the yield stress
C and FACYLD will be zero.
      DSMAG = DSMAG + ( ONE - FACYLD )
C Calculated increment in gamma ( this explicitly includes the time step)
      DIFF   = DSMAG - RADIUS
      DGAMMA = FACYLD * TERM * DIFF
C Update equivalent plastic strain
      DEQPS  = CON1 * DGAMMA
      STATENEW(I, 5) = STATEOLD(I, 5) + DEQPS
C Divide DGAMMA by DSMAG so that the deviatoric stresses are explicitly
C converted to tensors of unit magnitude in the following calculations
      DGAMMA = DGAMMA / DSMAG
C Update back stress
      FACTOR  = HARD * DGAMMA * TWO_THIRDS
      STATENEW(I, 1) = STATEOLD(I, 1) + FACTOR * DS1
      STATENEW(I, 2) = STATEOLD(I, 2) + FACTOR * DS2
      STATENEW(I, 3) = STATEOLD(I, 3) + FACTOR * DS3
      STATENEW(I, 4) = STATEOLD(I, 4) + FACTOR * S4
```



C Update stress

 FACTOR = TWOMU * DGAMMA

 STRESSNEW(I, 1) = SIG1 - FACTOR * DS1

 STRESSNEW(I, 2) = SIG2 - FACTOR * DS2

 STRESSNEW(I, 3) = SIG3 - FACTOR * DS3

 STRESSNEW(I, 4) = SIG4 - FACTOR * S4

C Update the specific internal energy -

 STRESS_POWER = HALF * (

 1 (STRESSOLD(I, 1)+STRESSNEW(I, 1)) * STRAININC(I, 1)

 2 + (STRESSOLD(I, 2)+STRESSNEW(I, 2)) * STRAININC(I, 2)

 3 + (STRESSOLD(I, 3)+STRESSNEW(I, 3)) * STRAININC(I, 3)

 4 + TWO * (STRESSOLD(I, 4)+STRESSNEW(I, 4)) * STRAININC(I, 4))

 ENERINTERNNEW(I) = ENERINTERNOLD(I)

 1 + STRESS_POWER/DENSITY(I)

C Update the dissipated inelastic specific energy -

 SMEAN = THIRD * (STRESSNEW(I, 1)+STRESSNEW(I, 2)

 1 + STRESSNEW(I, 3))

 EQUIV_STRESS = SQRT(THREE_HALFS

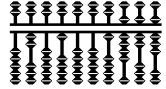
 1 * ((STRESSNEW(I, 1)-SMEAN)**2

 2 + (STRESSNEW(I, 2)-SMEAN)**2

 3 + (STRESSNEW(I, 3)-SMEAN)**2

 4 + TWO * STRESSNEW(I, 4)**2))

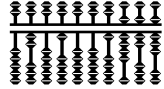
C



```
        PLASTIC_WORK_INC = EQUIV_STRESS * DEQPS  
        ENERINELASNEW(I) = ENERINELASOLD(I)  
1      +          PLASTIC_WORK_INC / DENSITY(I)  
C  
      END DO  
C  
      END IF  
      RETURN  
      END
```

Remarks

- In the **datacheck** phase, **VUMAT** is called with a set of fictitious strains and a **TOTALTIME** and **STEPTIME** both equal to 0.0.
 - A check is done on the user's constitutive relation, and an initial stable time increment is determined based on calculated equivalent initial material properties.
 - Ensure that elastic properties are used in this call to **VUMAT**; otherwise, too large an initial time increment may be used, leading to instability.
 - A warning message is printed to the status (**.sta**) file informing the user that this check is being performed.



- Special coding techniques are used to obtain vectorized coding.
 - All small loops inside the material routine are “unrolled.”
 - The same code is executed regardless of whether the behavior is purely elastic or elastic plastic.
- Special care must be taken to avoid divides by zero.
 - No external subroutines are called inside the loop.
 - The use of *local* scalar variables inside the loop is allowed.
 - The compiler will automatically expand these local scalar variables to local vectors.
 - Iterations should be avoided.
- If iterations cannot be avoided, use a fixed number of iterations and do not test on convergence.

Example 7: VUMAT for Isotropic Hardening

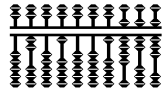
The governing equations and integration procedure are the same as in **Example 5: Isotropic Hardening Plasticity** (p. L6.69).

The increment of equivalent plastic strain is obtained explicitly through

$$\Delta \bar{\epsilon}^{pl} = \frac{\bar{\sigma}^{pr} - \sigma_y}{3\mu + h},$$

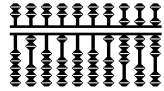
where σ_y is the yield stress and $h = d\sigma_y/d\bar{\epsilon}^{pl}$ is the plastic hardening at the beginning of the increment.

The Jacobian is not required.



Coding for Isotropic Hardening Plasticity VUMAT

```
C
C
      parameter ( zero = 0.d0, one = 1.d0, two = 2.d0,
*              third = 1.d0 / 3.d0, half = 0.5d0, op5 = 1.5d0)
C
C For plane strain, axisymmetric, and 3D cases using
C the J2 Mises Plasticity with piecewise-linear isotropic hardening.
C
C The state variable is stored as:
C
C              STATE(*,1) = equivalent plastic strain
C
C User needs to input
C      props(1)      Young's modulus
C      props(2)      Poisson's ratio
C      props(3..)    syield and hardening data
C      calls vuhard for curve of yield stress vs. plastic strain
C
```

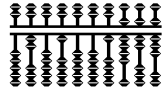


```

e      = props(1)
xnu    = props(2)
twomu  = e / ( one + xnu )
alamda = xnu * twomu / ( one - two * xnu )
thremu = op5 * twomu
nvalue = nprops/2-1

C
if ( stepTime .eq. zero ) then
  do k = 1, nblock
    trace = strainInc(k,1) + strainInc(k,2) + strainInc(k,3)
    stressNew(k,1) = stressOld(k,1)
*      + twomu * strainInc(k,1) + alamda * trace
    stressNew(k,2) = stressOld(k,2)
*      + twomu * strainInc(k,2) + alamda * trace
    stressNew(k,3) = stressOld(k,3)
*      + twomu * strainInc(k,3) + alamda * trace
    stressNew(k,4)=stressOld(k,4) + twomu * strainInc(k,4)
    if ( nshr .gt. 1 ) then
      stressNew(k,5)=stressOld(k,5) + twomu * strainInc(k,5)
      stressNew(k,6)=stressOld(k,6) + twomu * strainInc(k,6)
    end if
  end do
else

```



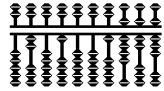
```
do k = 1, nblock

    peeqOld=stateOld(k,1)
    call vuhard(yieldOld, hard, peeqOld, props(3), nvalue)

    trace = strainInc(k,1) + strainInc(k,2) + strainInc(k,3)

    s11 = stressOld(k,1) + twomu * strainInc(k,1) + alamda * trace
    s22 = stressOld(k,2) + twomu * strainInc(k,2) + alamda * trace
    s33 = stressOld(k,3) + twomu * strainInc(k,3) + alamda * trace
    s12 = stressOld(k,4) + twomu * strainInc(k,4)

    if ( nshr .gt. 1 ) then
        s13 = stressOld(k,5) + twomu * strainInc(k,5)
        s23 = stressOld(k,6) + twomu * strainInc(k,6)
    end if
```



C

```
smean = third * ( s11 + s22 + s33 )
```

```
s11 = s11 - smean
```

```
s22 = s22 - smean
```

```
s33 = s33 - smean
```

```
if ( nshr .eq. 1 ) then
```

```
    vmises = sqrt( op5*(s11*s11+s22*s22+s33*s33+two*s12*s12) )
```

```
else
```

```
    vmises = sqrt( op5 * ( s11 * s11 + s22 * s22 + s33 * s33 +  
*                          two * s12 * s12 + two * s13 * s13 + two * s23 * s23 ) )
```

```
end if
```

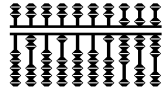
C

```
sigdif = vmises - yieldOld
```

```
facyld = zero
```

```
if ( sigdif .gt. zero ) facyld = one
```

```
deqps = facyld * sigdif / ( thremu + hard )
```

```
C
C Update the stress
C
      yieldNew = yieldOld + hard * deqps
      factor = yieldNew / ( yieldNew + thremu * deqps )

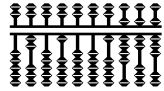
      stressNew(k,1) = s11 * factor + smean
      stressNew(k,2) = s22 * factor + smean
      stressNew(k,3) = s33 * factor + smean
      stressNew(k,4) = s12 * factor

      if ( nshr .gt. 1 ) then

          stressNew(k,5) = s13 * factor
          stressNew(k,6) = s23 * factor

      end if

C
C Update the state variables
C
      stateNew(k,1) = stateOld(k,1) + deqps
```



```
C
C Update the specific internal energy -
C
      if ( nshr .eq. 1 ) then

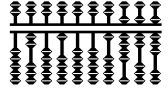
          stressPower = half * (
*           ( stressOld(k,1) + stressNew(k,1) ) * strainInc(k,1) +
*           ( stressOld(k,2) + stressNew(k,2) ) * strainInc(k,2) +
*           ( stressOld(k,3) + stressNew(k,3) ) * strainInc(k,3) ) +
*           ( stressOld(k,4) + stressNew(k,4) ) * strainInc(k,4)

          else

          stressPower = half * (
*           ( stressOld(k,1) + stressNew(k,1) ) * strainInc(k,1) +
*           ( stressOld(k,2) + stressNew(k,2) ) * strainInc(k,2) +
*           ( stressOld(k,3) + stressNew(k,3) ) * strainInc(k,3) ) +
*           ( stressOld(k,4) + stressNew(k,4) ) * strainInc(k,4) +
*           ( stressOld(k,5) + stressNew(k,5) ) * strainInc(k,5) +
*           ( stressOld(k,6) + stressNew(k,6) ) * strainInc(k,6)

          end if

          enerInternNew(k) = enerInternOld(k) + stressPower / density(k)
```

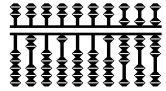


```
C
C Update the dissipated inelastic specific energy -
C
      plasticWorkInc = half * ( yieldOld + yieldNew ) * deqps
      enerInelasNew(k) = enerInelasOld(k)
      *
        + plasticWorkInc / density(k)

      end do

    end if

C
  return
end
```



```
      subroutine vuhard(syield, hard, eqplas, table, nvalue)
      include 'vaba_param.inc'

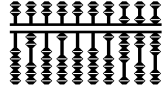
c
      dimension table(2, nvalue)

c
      parameter(zero=0.d0)

c
c      set yield stress to last value of table, hardening to zero
c
      syield=table(1, nvalue)
      hard=zero

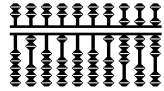
c
c      if more than one entry, search table
c
      if(nvalue.gt.1) then
        do k1=1, nvalue-1
          eqp11=table(2,k1+1)
          if(eqplas.lt.eqp11) then
            eqp10=table(2, k1)

c
c            yield stress and hardening
c
            deqp1=eqp11-eqp10
            syiel0=table(1, k1)
```



```
        syiel1=table(1, k1+1)
        dsyiel=syiel1-syiel0
        hard=dsyiel/deqpl
        syield=syiel0+(eqplas-eqpl0)*hard
        goto 10
    endif
end do
10    continue
endif

return
end
```



Remarks

- This **VUMAT** yields the same results as the ***PLASTIC** option with **ISOTROPIC** hardening.
 - This result is also true for large-strain calculations. The necessary rotations of stress and strain are taken care of by **ABAQUS**.
- The routine calls user subroutine **VUHARD** to recover a piecewise linear hardening curve.
 - It is straightforward to replace the piecewise linear curve by an analytic description.

Lecture 7

Creating a Nonlinear User Element

Overview

- Motivation
- Defining a User Element
- UEL Interface
- Example 1: Planar Beam Element with Nonlinear Section Behavior
- Example 2: Force Control Element
- Using Nonlinear User Elements in Various Analysis Procedures

Overview

ABAQUS/Standard has an interface that allows users to implement linear and nonlinear finite elements.

- A nonlinear finite element is implemented in user subroutine **UEL**.

The interface makes it possible to define any (proprietary) element of arbitrary complexity.

- If coded properly, user elements can be utilized with most analysis procedures in ABAQUS/Standard.
- Multiple user elements can be implemented in a single **UEL** routine and can be utilized together.

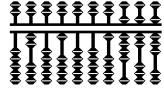
In this lecture the implementation of nonlinear finite elements *only* will be discussed and illustrated with examples.

Motivation

ABAQUS/Standard is a versatile analysis tool with a large element library that allows analysis of the most complex structural problems.

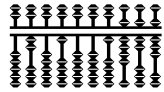
However, situations arise in which augmenting the ABAQUS library with user-defined elements is useful:

- Modeling nonstructural physical processes that are coupled to structural behavior
- Applying solution-dependent loads
- Modeling active control mechanisms



The advantages of implementing user elements in an analysis code such as ABAQUS, instead of writing a complete analysis code, are obvious:

- ABAQUS offers a large selection of structural elements, analysis procedures, and modeling tools.
- ABAQUS offers pre- and postprocessing.
 - Many third-party vendors offer pre- and postprocessors with interfaces to ABAQUS.
- Maintaining and porting subroutines is much easier than maintaining and porting a complete finite element program.



- “*Finite Element Simulations in Mechanics of Materials and Deformation Processing Research*,” Mary C. Boyce, ABAQUS Users’ Conference Proceedings, 1992.

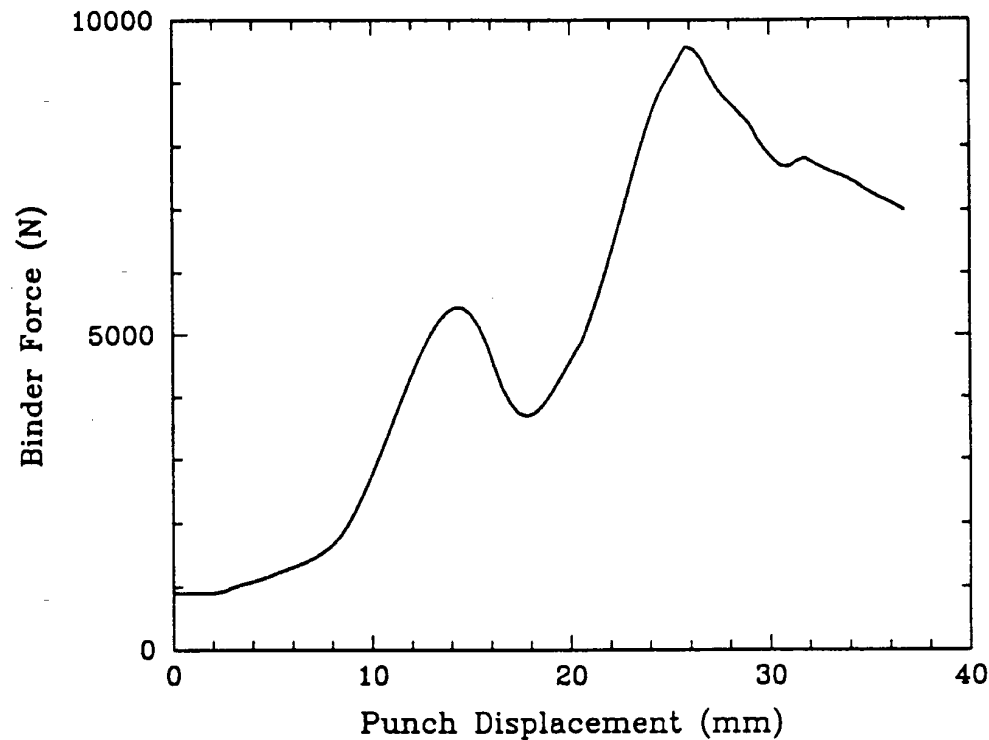
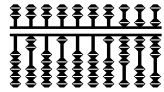


Figure 7–1. Calculated Binder Force Trajectory Using Active Global Binder Displacement and Local Strain Control



- “*User Elements Developed for the Nonlinear Dynamic Analysis of Reinforced Concrete Structures*,” Thomas Wenk, Peter Linde, and Hugo Bachmann, ABAQUS Users’ Conference Proceedings, 1993.

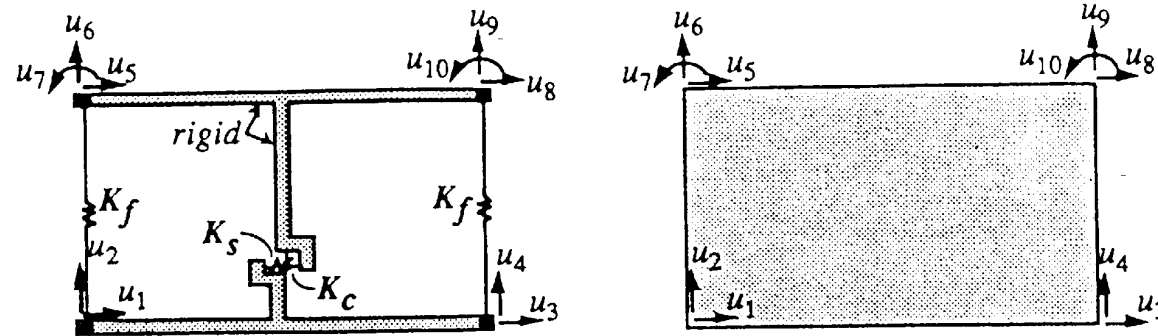
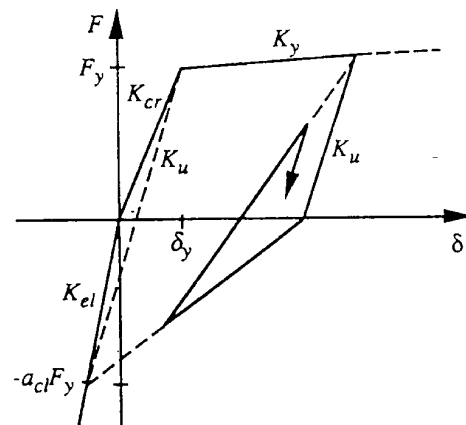


Figure 7–2. Macro Model Simulating Structural Wall Behavior (left), Corresponding User Element U3 (right)



Hysteretic Rules For
Flexural Springs K_f in
Macro Model (above)

- “*User Element for Crack Propagation in Concrete-Like Materials*,” R. Vitali and G.L. Zanutelli, ABAQUS Users’ Conference Proceedings, 1994.

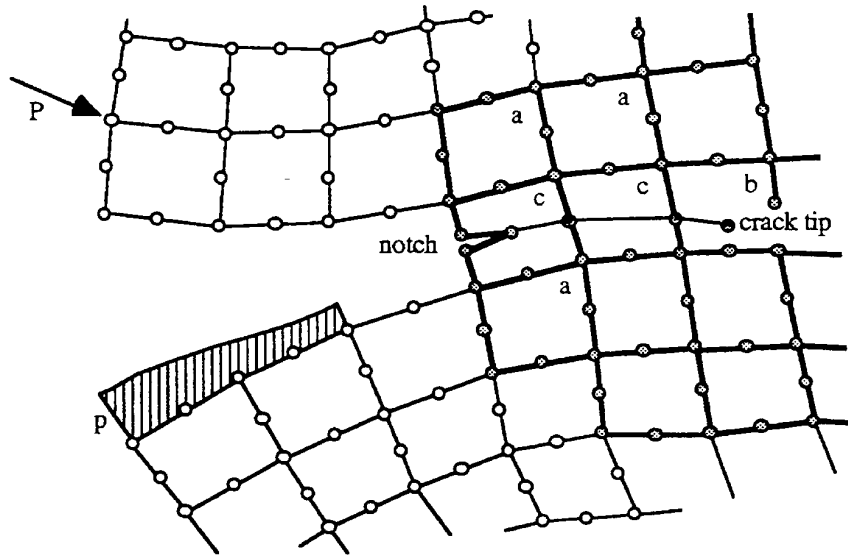
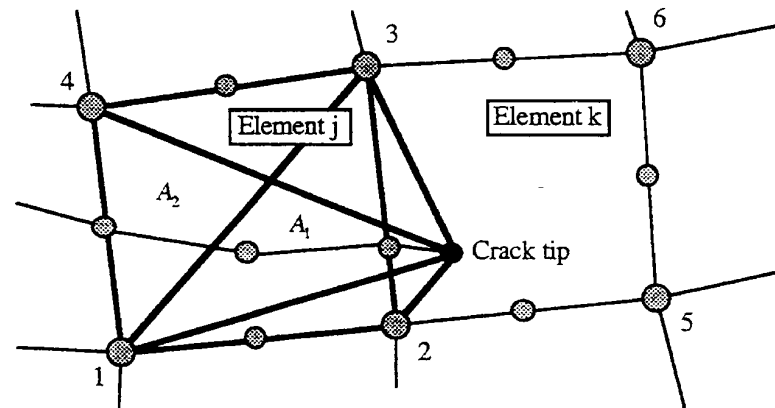
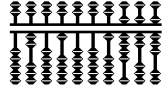


Figure 7–3. Configuration at a General Increment i .

Definition of the User Element



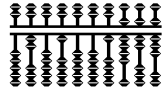


Defining a User Element

Key Characteristics of a User Element

Before a **UEL** subroutine can be written, the following key characteristics of the element must be defined:

- The number of nodes on the element
- The number of coordinates present at each node
- The degrees of freedom active at each node



Other Important Element Properties

In addition, the following properties must be determined:

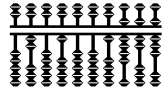
- The number of element properties to be defined external to the **UEL**
- The number of solution-dependent state variables (SDVs) to be stored per element
- The number of (distributed) load types available for the element

These items need not be determined immediately: they can be added easily after the basic **UEL** subroutine is completed.

Defining the User Element Behavior

The element's main contribution to the model during general analysis steps is to provide “fluxes” F^N at the nodes that depend on the values of the degrees of freedom u^N at the nodes.

- F^N is defined as a residual quantity: $F^N = F_{ext}^N - F_{int}^N$.
 - F_{ext}^N is the external flux (due to applied distributed loads) and F_{int}^N is the internal flux (due to stresses, e.g.) at node N .
- If the degrees of freedom are displacements, the associated fluxes are the nodal forces. Similarly, rotations correspond to moments and temperatures to heat fluxes.
- In nonlinear user elements the fluxes/forces will often depend on the increments in the degrees of freedom Δu^N and the internal state variables H^α .
 - State variables must be updated in the user subroutine.



The solution of the (nonlinear) system of equations in general steps requires that you define the element Jacobian (stiffness matrix):

$$K^{NM} = -\frac{dF^N}{du^M}.$$

- The Jacobian should include all direct and indirect dependencies of F^N on u^N , which includes terms of the form

$$-\frac{\partial F^N}{\partial H^\alpha} \frac{\partial H^\alpha}{\partial u^M}.$$

- A more accurately defined Jacobian improves convergence in general steps.
- The Jacobian (stiffness) determines the solution for linear perturbation steps, so it must be exact.
 - The Jacobian can be symmetric or nonsymmetric.

The complexity of the formulation of a user element can vary greatly.

- Simple elements can be developed to function as “control” and “feedback” mechanisms in an analysis that consists of regular elements.
- Complex nonlinear structural elements often require significant effort in their development.

If the element is built out of a nonlinear material, you should create a separate subroutine (or series of subroutines) to describe the material behavior.

- If the material model is implemented in user subroutine **UMAT**, a call to **UMAT** can be included in **UEL**.
- The integration issues discussed for **UMAT** also apply to the material models used in **UEL**.

UEL Interface

ABAQUS Options

A user element is defined with the *USER ELEMENT option. This option must appear in the input file before the user element is invoked with the *ELEMENT option.

The syntax for interfacing to UEL is as follows:

```
*USER ELEMENT, TYPE=Un, NODES=, COORDINATES=,
PROPERTIES=, I PROPERTIES=, VARIABLES=, UNSYMM
```

Data line(s)

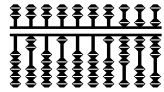
```
*ELEMENT, TYPE=Un, ELSET=UEL
```

Data line(s)

```
*UEL PROPERTY, ELSET=UEL
```

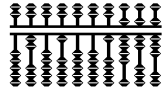
Data line(s)

```
*USER SUBROUTINES, (INPUT=file_name)
```



Parameter Definition

Parameter	Definition
TYPE	(User-defined) element type of the form Un , where n is a number
NODES	Number of nodes on the element
COORDINATES	Maximum number of coordinates at any node
PROPERTIES	Number of floating point properties
I PROPERTIES	Number of integer properties
VARIABLES	Number of SDVs
UNSYMM	Flag to indicate that the Jacobian is unsymmetric



Data Lines

A data line of the form

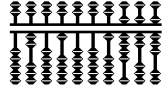
$dof_1, dof_2, \dots,$

where

dof_1 is the first degree of freedom active at the node and

dof_2 is the second degree of freedom active at the node, etc.,

follows the *USER ELEMENT option. If all nodes of the user element have the same active degrees of freedom, no further data are needed.



However, if some nodes have different active degrees of freedom, enter subsequent data lines of the form

position, dof_1, dof_2, ... ,

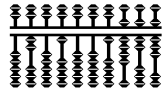
where

position is the (local) node number (position) on the element,

dof_1 is the first degree of freedom active at this and following nodes,
and

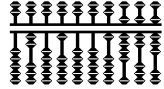
dof_2 is the second degree of freedom active at this and following nodes, etc.

The active degrees of freedom can be changed at any node in the element.



The dimensional units of a degree of freedom for a user element are the same as those for regular elements in ABAQUS (1–3 are displacements, 4–6 are rotations, etc.).

- This correspondence is important for convergence controls in nonlinear analysis.
- It is also relevant for three-dimensional rotations in geometric nonlinear analysis because of the nonlinear nature of finite rotations.



User elements can have “internal” degrees of freedom in the sense that they belong to nodes that are not connected to other elements.

- Convergence will be checked for the internal degrees of freedom, so it is important to choose the internal degrees of freedom appropriately (i.e., an internal degree of freedom 1 should have the dimension of displacement).
- For efficiency reasons you should choose internal degree of freedom numbers that are present at external nodes on the element or elsewhere in the model.

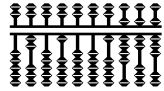
More on Keywords and Parameters

The maximum number of coordinates at any node of the element is specified with the COORDINATES parameter.

- The value of COORDINATES may be increased to match the highest displacement degree of freedom active on the element.

The total number of SDVs per element is set with the VARIABLES parameter.

- If the element is integrated numerically, VARIABLES should be set equal to the number of integration points times the number of SDVs per point.
- Solution-dependent state variables can be output with the identifiers SDV1, SDV2, etc. SDVs for any element can be printed *only* to the data (**.dat**), results (**.fil**), or output database (**.odb**) files and plotted as X–Y plots in ABAQUS/Viewer.

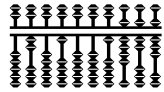


The number of user element properties is given with the **PROPERTIES** and **I PROPERTIES** parameters.

- **PROPERTIES** determines the number of floating point property values.
- **I PROPERTIES** determines the number of integer property values.

Property values are given with the ***UEL PROPERTY** option.

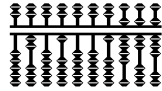
- The properties are assigned on an element set basis; hence, the same **UEL** subroutine can be used for user elements with different properties.
 - With this approach “hard-coding” the property values in the user subroutine is not necessary.



Coding for the **UEL** is supplied in a separate file and invoked with the ABAQUS execution procedure as follows:

```
abaqus job=... user=....
```

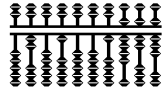
- The user subroutine must be invoked in a restarted analysis because user subroutines are not saved on the restart file.



User Element Loads

Distributed load and flux types can be applied with the *DLOAD and *DFLUX options by using load type keys Un and $UnNU$.

- In either case the equivalent nodal load vector for the distributed load type must be defined in user subroutine **UEL**.
 - If load type key Un is used, the load magnitude is defined on the data line and can be varied in time with the *AMPLITUDE option.
 - If load type key $UnNU$ is used, all of the load definition is applied in user subroutine **UEL**: a time-dependent load magnitude vector must be coded.
- If the load depends on the solution variables, the corresponding “load stiffness” contributions matrix to the Jacobian should be included for best performance.



UEL Interface

The interface to user subroutine **UEL** is:

```

SUBROUTINE UEL(RHS, AMATRX, SVARS, ENERGY, NDOFEL, NRHS, NSVARS,
1  PROPS, NPROPS, COORDS, MCRD, NNODE, U, DU, V, A, JTYPE, TIME,
2  DTIME, KSTEP, KINC, JELEM, PARAMS, NDLOAD, JDLTYPE, ADLMAG,
3  PREDEF, NPREDF, LFLAGS, MLVARX, DDLMAG, MDLOAD, PNEWDT, JPROPS,
4  NJPRO, PERIOD)

```

C

```

INCLUDE 'ABA_PARAM.INC'

```

C

```

DIMENSION RHS(MLVARX,*), AMATRX(NDOFEL, NDOFEL), PROPS(*),
1  SVARS(*), ENERGY(*), COORDS(MCRD, NNODE), U(NDOFEL),
2  DU(MLVARX,*), V(NDOFEL), A(NDOFEL), TIME(2), PARAMS(*),
3  JDLTYP(MDLOAD, *), ADLMAG(MDLOAD, *), DDLMAG(MDLOAD, *),
4  PREDEF(2, NPREDF, NNODE), LFLAGS(*), JPROPS(*)

```

The “include” statement sets the proper precision for floating point variables (**REAL*8** on most machines).

UEL Variables

The following quantities are available in **UEL**:

- Coordinates; displacements; incremental displacements; and, for dynamics, velocities and accelerations
- SDVs at the start of the increment
- Total and incremental values of time, temperature, and user-defined field variables
- User element properties
- Load types as well as total and incremental load magnitudes
- Element type and user-defined element number
- Procedure type flag and, for dynamics, integration operator values
- Current step and increment numbers

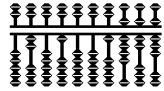
The following quantities *must* be defined:

- Right-hand-side vector (residual nodal fluxes or forces)
- Jacobian (stiffness) matrix
- Solution-dependent state variables

The following variables may be defined:

- Energies associated with the element (strain energy, plastic dissipation, kinetic energy, etc.)
- Suggested new (reduced) time increment

A complete description of all parameters is provided in Section 24.2.19 of the ABAQUS/Standard User's Manual.



The header is usually followed by dimensioning of local arrays. It is good practice to define constants via parameters and to include comments.

```
        dimension b(2, 7), gauss(2)
c
        parameter(zero=0.d0, one=1.d0, two=2.d0, three=3.d0, four=4.d0,
1          six=6.d0, eight=8.d0, twelve=12.d0)
        data gauss/.211324865d0, .788675135d0/
c
c simple 2-d linear beam element with generalized section properties
c
c get length and direction and cross section dimensions
c
```

- The PARAMETER assignments yield accurate floating point constant definitions on any platform.
- Arrays can be initialized with a DATA statement.

UEL Conventions

The solution variables (displacement, velocity, etc.) are arranged on a node/degree of freedom basis.

- The degrees of freedom of the first node are first, followed by the degrees of freedom of the second node, etc.
 - Consider a planar beam that uses degrees of freedom 1, 2, and 6 (u_x , u_y , ϕ_z) at its first and second node and degrees of freedom 1 and 2 at its third (middle) node. The ordering is:

Element variable	1	2	3	4	5	6	7	8
Node	1	1	1	2	2	2	3	3
Degree of freedom	1	2	6	1	2	6	1	2

- The flux vector and Jacobian matrix must be ordered in the same way.

UEL Formulation Aspects and Usage Hints

The displacement, velocities, etc. passed into the **UEL** are in the global system, regardless whether the *TRANSFORM option is used at any of the nodes.

- The flux vector and Jacobian matrix must also be formulated in the global system.

The Jacobian must be formulated as a *full* matrix, even if it is symmetric.

- If the UNSYMM parameter is not used, ABAQUS will symmetrize the Jacobian defined by the user.

For transient heat transfer and dynamic analysis, heat capacity and inertia contributions must be included in the flux vector.

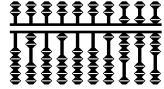
- **UELS** for these procedures will be discussed later in this lecture.

At the start of a new increment, the increment in solution variable(s) is extrapolated from the previous increment.

- The flux vector and the Jacobian must be based on these extrapolated values.
- If extrapolation is not desired, it can be switched off with `*STEP, EXTRAPOLATION=NO`.

If the increment in solution variable(s) is too large, the variable **PNEWDT** can be used to suggest a new time increment.

- ABAQUS will abandon the current time increment and will attempt the increment again with one that is a factor **PNEWDT** smaller.



Coding and Testing the UEL

Follow the basic rules for writing ABAQUS user subroutines.

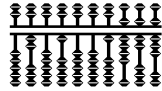
- Follow FORTRAN 77 or C conventions.
- Make sure that all variables are defined and initialized.
- Assign enough storage space for state variables.

Complex **UEL**s may have many potential problem areas. Do not use a large model when trying to debug a **UEL**.

Verify the **UEL** with a one-element input file.

1. Run tests using *general* steps in which all solution variables are prescribed to verify the resultant fluxes.
2. Run tests using *linear perturbation* steps in which all loads are prescribed to verify the element Jacobian (stiffness).
3. Run tests using *general* steps in which all loads are prescribed to verify the consistency of the Jacobian and the flux vector.

Gradually increase the complexity of the test problems. Compare the results with standard ABAQUS elements, if possible.

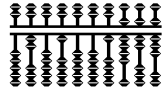


Example 1: Planar Beam Element with Nonlinear Section Behavior

Objective

Analyze a planar concrete frame structure.

- The frame is loaded to an extent where significant nonlinearity occurs in the concrete but the displacements are still small enough that geometric nonlinearity may be neglected.
- Develop a model that describes the nonlinear section behavior directly in terms of axial force and bending moment.
 - This is similar to the
*BEAM GENERAL SECTION, SECTION=NONLINEAR
GENERAL option, but allows coupling between the axial and bending terms.



- The transverse shear deformation can be neglected.

Coding Requirements

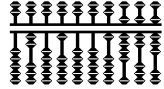
The element is integrated numerically; hence, the following quantities require definition in the **UEL**:

- The element $[B]$ matrix, which relates the axial strain, ϵ , and curvature, κ , to the element displacements, $\{u_e\}$:

$$\begin{Bmatrix} \epsilon \\ \kappa \end{Bmatrix} = [B] \{u_e\} .$$

- A constitutive law $[D]$ relating axial force, F , and moment, M , to axial strain and curvature:

$$\begin{Bmatrix} F \\ M \end{Bmatrix} = [D] \begin{Bmatrix} \epsilon \\ \kappa \end{Bmatrix} .$$



- The element stiffness matrix:

$$[K_e] = \int_0^l [B]^T [D] [B] dl \ .$$

- The element internal force vector:

$$\{F_e\} = \int_0^l [B]^T \begin{Bmatrix} F \\ M \end{Bmatrix} dl \ .$$

- The integration is done numerically:

$$\int_0^l A dl = \sum_{i=1}^n A_i l_i,$$

where n is the number of integration points and l_i is the length associated with integration point i .

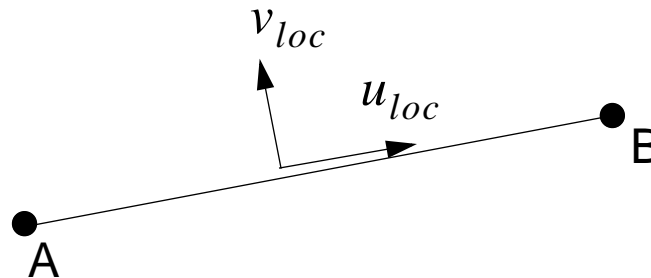
Element Formulation

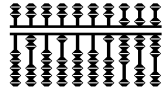
The element formulation is based on Euler-Bernoulli beam theory.

- The interpolation is described purely in terms of the displacements, which are C_1 continuous at the nodes.
- The curvature is obtained as the second derivative of the displacement normal to the beam.

The simplest two-dimensional beam element has two nodes, with two displacements and one rotation (u_x, u_y, ϕ_z) at each node.

- The active degrees of freedom are 1, 2, and 6.



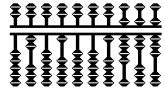


In its basic form linear interpolation is used for the tangential displacement, u_{loc} , and cubic interpolation for the normal displacement, v_{loc} .

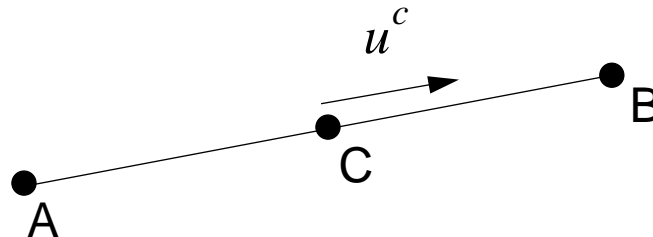
- The cubic interpolation for the normal displacement yields a linear variation for the curvature.
- The linear interpolation for the tangential displacement yields a constant axial strain.

The constant axial strain and linear curvature variation are inconsistent and may lead to excessive local axial forces if the axial and bending behavior are coupled.

- Considering that the intent is to analyze nonlinear concrete behavior, such coupling will be present.
- The excessive axial forces may lead to overly stiff behavior.



To prevent this problem, an extra “internal” node is added to the element. The internal node has one degree of freedom: the tangential displacement.

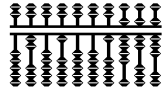


Both the axial strain and the curvature now vary linearly. The interpolation functions are:

$$u_{loc} = u_{loc}^A(1 - 3\xi + 2\xi^2) + u_{loc}^B(-\xi + 2\xi^2) + u^C(4\xi - 4\xi^2)$$

$$v_{loc} = v_{loc}^A(1 - 3\xi^2 + 2\xi^3) + v_{loc}^B(3\xi^2 - 2\xi^3) \\ + \phi^A l(\xi - 2\xi^2 + \xi^3) + \phi^B l(-\xi^2 + \xi^3)$$

where l is the element length and $\xi = s/l$ is the dimensionless position along the beam.

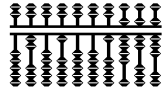


- This yields the following expressions for the axial strain and the curvature:

$$\varepsilon = \frac{1}{l} [u_{loc}^A(-3 + 4\xi) + u_{loc}^B(-1 + 4\xi) + u^C(4 - 8\xi)]$$

$$\kappa = \frac{1}{l^2} [v_{loc}^A(-6 + 12\xi) + v_{loc}^B(6 - 12\xi) + \phi^A l(-4 + 6\xi) + \phi^B l(-2 + 6\xi)]$$

- These linear relations are implemented in the B-matrix of the element.
 - The B-matrix also handles the transformation from local to global displacements at the nodes.
- The element is integrated numerically with a two-point Gauss scheme.

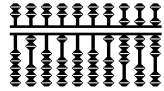


Element Definition in the Input File

The following data lines define the user element in the input file:

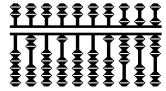
```
*user element, type=u1, nodes=3, coordinates=2, properties=3, variables=8
1, 2, 6
3, 1
*element, type=u1, elset=one
1, 1, 2, 3
*uel property, elset=one
2., 1., 1000.
```

- The user element name is **U1**, which is used in the *ELEMENT option.
- Eight state variables are allocated, so four variables can be defined at each integration point.
- Three element properties are allocated: the section height, the section width, and Young's modulus.
- The element has three nodes: the third “internal” node is unique to each element.

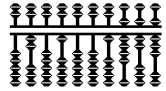


Coding for Planar Beam Example

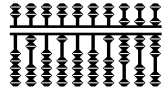
```
c
c  simple 2-d linear beam element with generalized section properties
c
      subroutine uel(rhs, amatrix, svars, energy, ndofel, nrhs, nsvars,
1  props, nprops, coords, mcrd, nnode, u, du, v, a, jtype, time, dtime,
2  kstep, kinc, jelem, params, ndload, jdltyp, adlmag, predef, npredf,
3  lflags, mlvarx, ddlmag, mdload, pnewdt, jprops, njprop, period)
c
      include 'aba_param.inc'
c
      dimension rhs(mlvarx, *), amatrix(ndofel, ndofel), svars(*), props(*),
1  energy(7), coords(mcrd,nnode), u(ndofel), du(mlvarx, *), v(ndofel),
2  a(ndofel), time(2), params(*), jdltyp(mdload, *), adlmag(mdload, *),
3  ddlmag(mdload, *), predef(2, npredf, nnode), lflags(4), jprops(*)
c
      dimension b(2, 7), gauss(2)
c
      parameter(zero=0.d0, one=1.d0, two=2.d0, three=3.d0, four=4.d0,
1  six=6.d0, eight=8.d0, twelve=12.d0)
      data gauss/.211324865d0, .788675135d0/
c
```



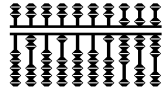
```
c  calculate length and direction cosines
c
      dx=coords(1, 2)-coords(1, 1)
      dy=coords(2, 2)-coords(2, 1)
      dl2=dx**2+dy**2
      dl=sqrt(dl2)
      hdl=dl/two
      acos=dx/dl
      asin=dy/dl
c
c  initialize rhs and lhs
c
      do k1=1, 7
        rhs(k1, 1)= zero
        do k2=1, 7
          amatrix(k1, k2)= zero
        end do
      end do
c
      nsvint=nsvars/2
```



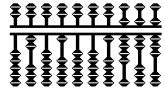
```
c
c  loop over integration points
c
      do kintk=1, 2
        g=gauss(kintk)
c
c  make b-matrix
c
        b(1, 1)=(-three+four*g)*acos/dl
        b(1, 2)=(-three+four*g)*asin/dl
        b(1, 3)=zero
        b(1, 4)=(-one+four*g)*acos/dl
        b(1, 5)=(-one+four*g)*asin/dl
        b(1, 6)=zero
        b(1, 7)=(four-eight*g)/dl
        b(2, 1)=(-six+twelve*g)*-asin/dl2
        b(2, 2)=(-six+twelve*g)* acos/dl2
        b(2, 3)=(-four+six*g)/dl
        b(2, 4)= (six-twelve*g)*-asin/dl2
        b(2, 5)= (six-twelve*g)* acos/dl2
        b(2, 6)= (-two+six*g)/dl
        b(2, 7)=zero
c
```

```
c  calculate (incremental) strains and curvatures
c
      eps=zero
      deps=zero
      cap=zero
      dcap=zero
      do k=1, 7
        eps=eps+b(1, k)*u(k)
        deps=deps+b(1, k)*du(k, 1)
        cap=cap+b(2, k)*u(k)
        dcap=dcap+b(2, k)*du(k, 1)
      end do
c
c  call constitutive routine ugenb
c
      isvint=1+(kintk-1)*nsvint
      bn=zero
      bm=zero
      daxial=zero
      dbend=zero
      dcoupl=zero
      call ugenb(bn, bm, daxial, dbend, dcoupl, eps, deps, cap, dcap,
1          svars(isvint), nsvint, props, nprops)
```

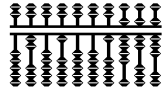


```
c
c  assemble rhs and lhs
c
      do k1=1, 7
        rhs(k1, 1)=rhs(k1, 1)-hdl*(bn*b(1, k1)+bm*b(2, k1))
        bd1=hdl*(daxial*b(1, k1)+dcoupl*b(2, k1))
        bd2=hdl*(dcoupl*b(1, k1)+dbend *b(2, k1))
        do k2=1, 7
          amatrix(k1, k2)=amatrix(k1, k2)+bd1*b(1, k2)+bd2*b(2, k2)
        end do
      end do
    end do
c
      return
    end
```



Remarks

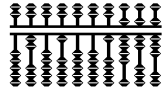
- This **UEL** uses essentially the same formulation as the simple B23 element for geometrically linear analysis.
- The routine can be used with and without the ***TRANSFORM** option.
- It would be relatively straightforward to generalize this routine for three-dimensional analyses.
 - It is much more complicated to extend the routine to geometrically nonlinear analysis.
- Even for linear analysis this routine is called twice (for each element) during the first iteration of an increment: once for assembly and once for recovery. Subsequently, it is called once per iteration: assembly and recovery are combined.



Generalized Constitutive Behavior

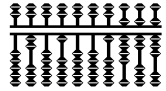
At each integration point the generalized constitutive behavior is processed in the user-created subroutine **UGENB**.

- This subroutine is patterned after user subroutine **UGENS**, which allows you to model the behavior of a shell.
- The following quantities are passed into **UGENB**:
 - Total and incremental axial strain and curvature
 - State variables at the start of the increment
 - User element properties
- You must define:
 - The axial force and bending moment, as well as the linearized force/moment-strain/curvature relations
 - The solution-dependent state variables

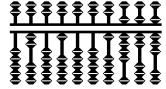


A simple linear elastic subroutine **UGENB** follows:

```
      subroutine ugenb(bn,bm,daxial,dbend,dcoupl,eps,deps,cap,dcap,
1          svint,nsvint,props,nprops)
c
c      include 'aba_param.inc'
c
c      parameter(zero=0.d0,twelve=12.d0)
c
c      dimension svint(*),props(*)
c
c  variables to be defined by the user
c
c  bn      - axial force
c  bm      - bending moment
c  daxial  - current tangent axial stiffness
c  dbend   - current tangent bending stiffness
c  dcoupl  - tangent coupling term
c
c  variables that may be updated
c
c  svint   - state variables for this integration point
```



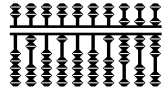
```
c
c  variables passed in for information
c
c  eps      - axial strain
c  deps     - incremental axial strain
c  cap      - curvature change
c  dcap     - incremental curvature change
c  props    - element properties
c  nprops   - # element properties
c  nsvint   - # state variables
c
c  current assumption
c
c  props(1) - section height
c  props(2) - section width
c  props(3) - Young's modulus
c
      h=props(1)
      w=props(2)
      E=props(3)
```



```
c
c  formulate linear stiffness
c
      daxial=E*h*w
      dbend=E*w*h**3/twelve
      dcoupl=zero
c
c  calculate axial force and moment
c
      bn=svint(1)+daxial*deps
      bm=svint(2)+dbend*dcap
c
c  store internal variables
c
      svint(1)=bn
      svint(2)=bm
      svint(3)=eps
      svint(4)=cap
c
      return
      end
```

Remarks

- The coding in this routine is very similar in nature to what would be coded in the subroutines **UMAT** and **UGENS**.
- The routine stores the axial strain, curvature, axial force, and bending moment at each Gauss point.
 - For nonlinear material behavior additional quantities would be stored.
- The constitutive relation is written in incremental form for easy generalization to nonlinear section behavior.
- **UGENB** and **UEL** must be combined in *one* external file.

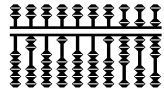


Example 2: Force Control Element

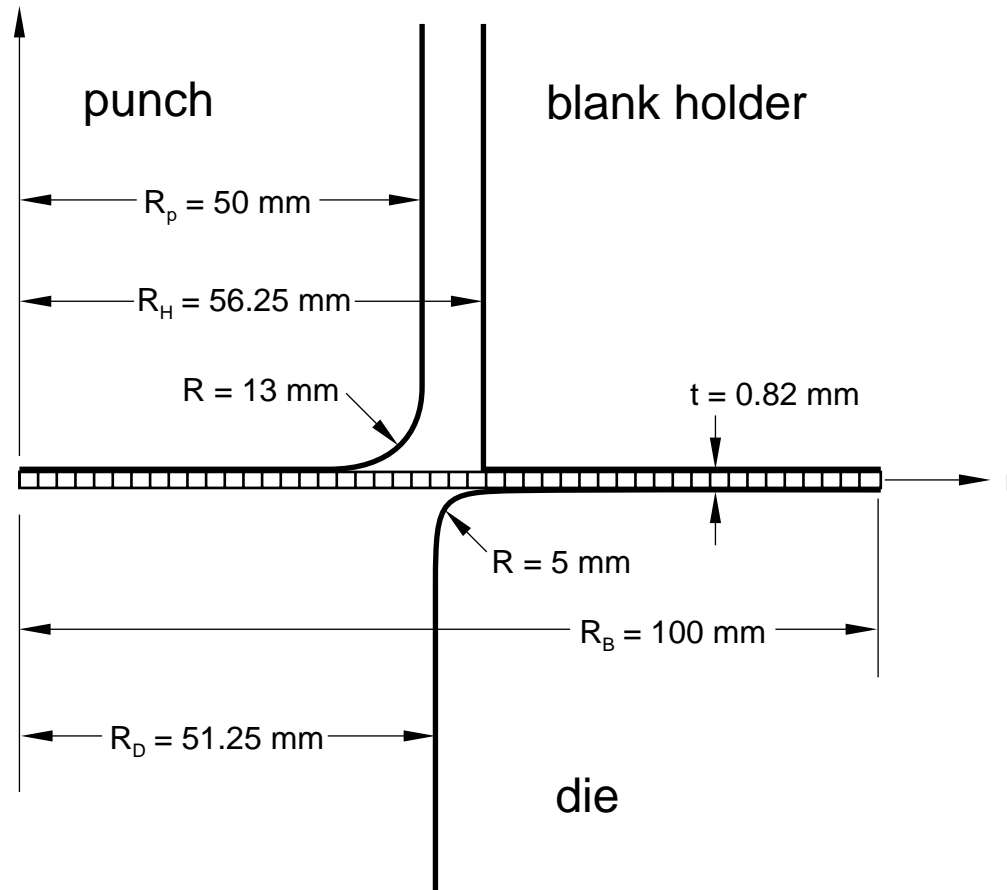
Objective

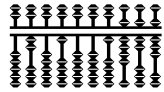
Implement an active control mechanism in a finite element model.

- The specific objective is to model a deep drawing operation in which the blank holder force is dynamically adapted during the process.
 - The punch force is measured during the deep drawing process.
 - If the punch force approaches a value that might cause tearing of the sheet, the blank holder force is decreased.
 - If the punch force decreases significantly below the critical value for tearing, the blank holder force is increased to minimize the danger of wrinkling.

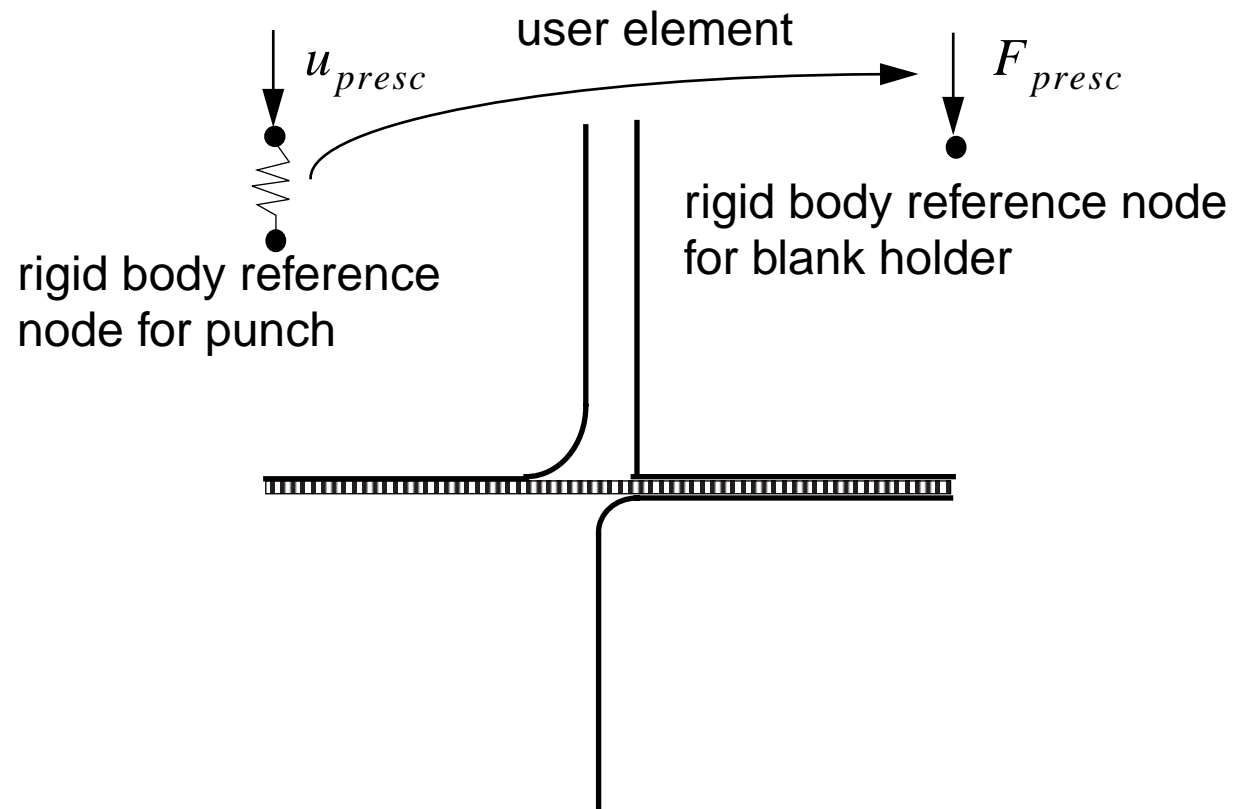


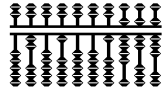
The problem is an adaptation of ABAQUS Example Problem 1.3.4, *Deep Drawing of a Cylindrical Cup*.





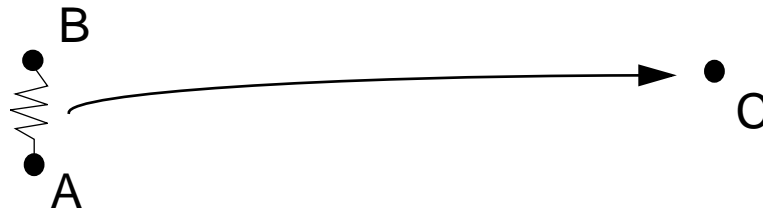
In this problem a user element is inserted that measures the punch force (by adding a stiff spring to the rigid body reference node) and, if necessary, modifies the blank holder force.





Element Formulation

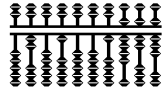
The element has three nodes, with a single degree of freedom (u_z), active at each node.



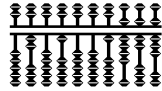
- Nodes A and B are connected by a spring with stiffness K , so the element internal forces at nodes A and B are

$$F_{int}^A = -F_{int}^B = K(u_z^A - u_z^B) .$$

- The stiffness is entered as a user element property.



- The magnitude of the force generated at node *C* is determined as follows:
 - Prescribe the initial value (as an element property).
 - Define the target value of the punch force.
 - If the spring force has not yet exceeded the target value, keep the force on node *C* the same.
 - If the spring force exceeds the target value by more than a user-defined tolerance, decrease the force on node *C*.
 - If the spring force drops below the target value by more than a user-defined tolerance, increase the force on node *C*.
- The algorithm uses the value of the spring force at the start of the increment.
 - Consequently, there is no stiffness contribution due to the application of the force on node *C*.

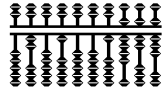


Element Definition in the Input File

The following data lines define the user element in the input file:

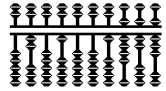
```
*USER ELEMENT, TYPE=U1, NODES=3, COORD=2, PROPERTIES=5, VARIABLES=3
2
*UEL PROPERTY, ELSET=FEEDBACK
1.E12, 7.5E4, 1.0E5, 0.04, 0.04
*ELEMENT, TYPE=U1, ELSET=FEEDBACK
1001, 200,500,300
```

- Five property values are specified: the spring stiffness, the target value of the punch force, the initial force on the blank holder, the allowable fractional change in holder force, and the punch force tolerance.
- Three state variables are stored: the punch force, the blank holder force, and the maximum punch force during the load history.
- The first and the last nodes are the rigid body reference nodes of the punch and blank holder, respectively.

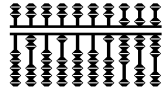


Coding for Force Control Element Example

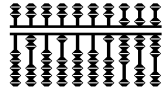
```
c
c Blankholder force control element for deep drawing applications
c
      subroutine uel(rhs,amatrix,svars,energy,ndofel,nrhs,nsvars,
1  props,nprops,coords,mcrd,nnode,u,du,v,a,jtype,time,dtime,
2  kstep,kinc,jelem,params,ndload,jdltyp,adlmag,predef,npredf,
3  lflags,mlvarx,ddlmag,mdload,pnewdt,jprops,njprop,period)
c
      include 'aba_param.inc'
c
      dimension rhs(mlvarx,*),amatrix(ndofel,ndofel),svars(*),props(*),
1  energy(7),coords(mcrd,nnode),u(ndofel),du(mlvarx,*),v(ndofel),
2  a(ndofel),time(2),params(*),jdltyp(mdload,*),adlmag(mdload,*),
4  ddlmag(mdload,*),predef(2,npredf,nnode),lflags(4),jprops(*)
c
c Pick up the input data
c
      sPunch          = props(1)          !Spring stiffness
      fPunchTarget    = props(2)          !Target punch force
      fHolderInit     = props(3)          !Initial blankholder force
      fractHolder     = props(4)          !Fractional change allowed
      tolPunch        = props(5)          !Tolerance on punch force
```



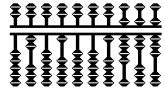
```
c
c Calculate the punch force
c
      fPunchNew = sPunch * (u(1)-u(2))
c
c Generate force vector and
c
      rhs(1,1) = -fPunchNew
      rhs(2,1) = +fpunchNew
c
c Generate stiffness matrix
c
      amatrix(1,1) = +sPunch
      amatrix(1,2) = -sPunch
      amatrix(2,1) = -sPunch
      amatrix(2,2) = +sPunch
c
c The holder force is only applied during steps 2 and 3
c
      if(kstep.eq.2) then
c
c Ramp the punch force to the desired starting value
```

```
c
      fHolder  =  time(1)*fHolderInit/period
      svars(2) =  fHolder
      rhs(3,1) = -fHolder
      else if(kstep.eq.3) then
c
c   Adjust the punch force to control the blankholder force
c
c   Values of state variables at start of increment
c
      fPunchOld  =  svars(1)           !Punch force
      fHolderOld =  svars(2)           !Blankholder force
      fPunchMax  =  svars(3)           !Maximum blankholder force
c
c   Allowed change in blankholder force
c
      dfHolderMax =  fractHolder * fHolderOld
c
c   Allowed tolerance in the targetforce
c
      dfPunchTol  =  tolPunch * fPunchTarget
c
```

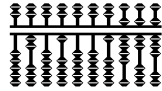


```
c Calculate the holder force
c
      if (fPunchOld.gt.fPunchTarget+dfPunchTol) then
        fHolderNew = fHolderOld - dfHolderMax    !Decrease
      else if(fPunchMax.lt.fPunchTarget+dfPunchTol .or.
1      fPunchOld.gt.fPunchTarget-dfPunchTol) then
        fHolderNew = fHolderOld                  !Keep constant
      else
        fHolderNew = fHolderOld + dfHolderMax    !Increase
      end if
c
c Generate holder force vector
c
      rhs(3,1) = -fHolderNew
c
c Update state variables
c
      svars(1) = fPunchNew
      svars(2) = fHolderNew
      svars(3) = max(fPunchMax, fPunchNew)
    end if
c
    return
  end
```



Remarks

- No stiffness contribution is associated with the change in blank holder force because the force does not depend on the solution of the *current* increment.
 - An additional advantage is that the change in blank holder force will always be based on a converged solution.
- In principle, this explicit algorithm is only conditionally stable.
 - However, since the punch force depends weakly on the blank holder force and the change in blank holder force is small, instability is unlikely to occur.
- More sophisticated feedback algorithms are readily implemented.



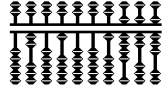
- The blank holder force can also be made dependent on other solution variables.
 - Subroutine **URDFIL** can be used to read any solution variable from the results (**.fil**) file during analysis.
 - The selected variable or variables (for example, the maximum strain or plastic strain anywhere in the model) can be stored in a common block, and the variables can be used in the **UEL**.
- All ABAQUS common block names start with the letter “C,” so name conflicts can easily be avoided.

Using Nonlinear User Elements in Various Analysis Procedures

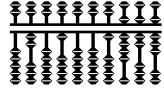
Overview of Procedures

Nonlinear user elements can be utilized in most ABAQUS/Standard analysis procedures.

- **LFLAGS (1)** indicates which procedure type is used:
 - **LFLAGS (1) =11**: Dynamic procedure with automatic time incrementation
 - **LFLAGS (1) =12**: Dynamic procedure with fixed time incrementation

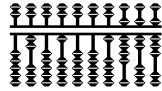


- To this point in the lecture, the usages described have applied only to `*STATIC (LFLAGS (1) =1, 2)`.
- The usage in many procedures is the same or similar to that for `*STATIC`:
 - `*VISCO`
 - `*HEAT TRANSFER, STEADY STATE`
 - `*COUPLED TEMPERATURE-DISPLACEMENT, STEADY STATE`
 - `*GEOSTATIC`
 - `*SOILS, STEADY STATE`
 - `*COUPLED THERMAL-ELECTRICAL, STEADY STATE`



A special case of static analysis is *STATIC, RIKS.

- An additional force vector containing only forces proportional to the applied loads, as well as the usual force vector and the Jacobian, must be supplied.
 - These additional forces must include thermal expansion effects if any are present in the element.
- If no forces are applied to the element, the usage is the same as that for a regular *STATIC analysis.



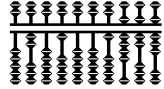
Perturbation Procedures

User elements can also be used in most “linear perturbation” procedures.

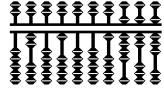
- For a static linear perturbation analysis (*STATIC, PERTURBATION), a stiffness matrix and two force vectors must be returned by the **UEL**.
 - The value of **LFLAGS (3)** denotes the matrix to be returned in a call.
 - **LFLAGS (3) = 1**: Assembly—return the stiffness matrix for the base state and the force vector that contains only external perturbation loads.
 - **LFLAGS (3) = 100**: Recovery—return the force vector that contains the difference between external perturbation loads and internal perturbation forces:

$$F^N = \Delta P^N - K^{NM} \Delta u^M.$$

This force vector is used for the reaction force calculation.



- For a *FREQUENCY analysis a stiffness and mass matrix must be returned by the UEL.
 - The value of **LFLAGS (3)** denotes the matrix to be returned in a call.
LFLAGS (3) = 2: Return the stiffness matrix
LFLAGS (3) = 4: Return the mass matrix
 - No element output is available for user elements utilized with the *FREQUENCY option.



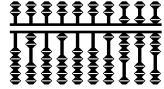
- The eigenfrequencies and eigenvectors obtained with the *FREQUENCY option can be used in all modal dynamics procedures:
 - *MODAL DYNAMIC
 - *STEADY STATE DYNAMICS
 - *RESPONSE SPECTRUM
 - *RANDOM RESPONSE
- User elements cannot be used in the *STEADY STATE DYNAMICS, DIRECT and *BUCKLE procedures.

Transient Analysis

First-order transient effects must be included in **UELs** that are used with the following procedures:

- *HEAT TRANSFER (transient)
- *SOILS, CONSOLIDATION
- *COUPLED TEMPERATURE-DISPLACEMENT (transient)
- *COUPLED THERMAL-ELECTRICAL, TRANSIENT

- The heat (pore fluid) capacity term must be included in the flux vector and the Jacobian.
- If the user element has no heat (pore fluid) capacity, the user element usage is the same as in the corresponding steady-state analysis.

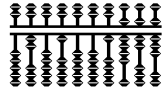


Transient Heat Transfer Analysis

LFLAGS (1) indicates the transient heat transfer procedure type being used:

- **LFLAGS (1) = 32**: Transient heat transfer analysis with automatic time incrementation
- **LFLAGS (1) = 33**: Transient heat transfer analysis with fixed time incrementation

Additional coding related to the transient terms in the equilibrium equation is required for the flux vector and Jacobian.



- The flux vector must contain the externally applied fluxes, the fluxes due to conduction, and the fluxes due to changes in internal energy:

$$F^N = F_{ext}^N + F_{cond}^N + F_{cap}^N.$$

- If the heat capacity matrix C^{NM} is constant, the flux due to the heat capacity terms is

$$F_{cap}^N = -C^{NM} \Delta\theta^M / \Delta t,$$

where $\Delta\theta^M$ is the temperature increment.

- If the heat capacity matrix varies with temperature (such as is the case during phase transformations), the flux vector must be calculated from the energy change vector:

$$F_{cap}^N = -\Delta U^M / \Delta t.$$

- The Jacobian will contain contributions from the conductivity and heat capacity terms.

- If the heat capacity matrix is constant, the Jacobian has the form

$$K^{NM} + C^{NM} / \Delta t,$$

where K^{NM} is the conductivity matrix.

- If the heat capacity matrix is a function of temperature, the Newton algorithm requires the heat capacity at the temperature at the end of the increment:

$$C^{NM} = C^{NM}(\theta_{t + \Delta t}).$$

For cases in which the heat capacity varies strongly (such as in case of latent heat), convergence may be difficult.

If the user element contains no heat capacity terms, the formulation for transient heat transfer is the same as for steady-state heat transfer.

Dynamic Analysis

Second-order transient (inertial) effects must be included in **UELs** that are used with direct integration dynamic analysis (*DYNAMIC).

LFLAGS (1) indicates the dynamics procedure type being used:

- **LFLAGS (1) =11**: Dynamic procedure with automatic time incrementation
- **LFLAGS (1) =12**: Dynamic procedure with fixed time incrementation

Additional coding related to transient terms, sudden changes in velocities or accelerations, and evaluation of the half-step residual (if automatic time incrementation is used) is required in the **UEL**.

The value of **LFLAGS (3)** indicates the coding being executed and the matrices to be returned.

LFLAGS (3) = 1

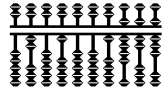
Normal time increment. The user must specify the forces and Jacobian corresponding to the integration procedure used.

- The force vector has the form

$$F^N = -M^{NM} \ddot{u}_{t+\Delta t}^M + (1 + \alpha) G_{t+\Delta t}^N - \alpha G_t^N,$$

where M^{NM} is the element mass matrix, G^N is the “static” force vector, and α is the Hughes-Hilbert-Taylor integration operator.

- The static force vector G^N must also contain the rate-dependent (damping) terms.
- The vector G_t^N must be stored as a set of state variables.
- The parameter α is passed into the subroutine as **PARAMS (1)**.



- The Jacobian has the form

$$M^{NM} \left(\frac{d\ddot{u}}{du} \right) + (1 + \alpha) C^{NM} \left(\frac{d\dot{u}}{du} \right) + (1 + \alpha) K^{NM} ,$$

where C^{NM} is the element damping matrix and K^{NM} is the static tangent stiffness matrix.

- $(d\ddot{u}/du)$ and $(d\dot{u}/du)$ follow from the integration operator. For the HHT operator,

$$(d\ddot{u}/du) = \frac{1}{\beta \Delta t^2}, (d\dot{u}/du) = \frac{\gamma}{\beta \Delta t} ,$$

where $\beta = (1/4)(1 - \alpha)^2$ and $\gamma = 1/2 - \alpha$ are the coefficients in the Newmark- β operator

- The parameters β and γ are passed into the subroutine as **PARAMS (2)** and **PARAMS (3)**.

Remarks

- The coding is simplified considerably if the HHT parameter $\alpha=0$.
 - In particular, there is no need to store the static residual vector, G_t^N .
 - The variable, α , can be set to zero with the ALPHA parameter on the *DYNAMIC option.
- If the user element has no inertia or damping terms (i.e., if the force vector does not depend on the velocities and accelerations), the α parameter can be ignored in the subroutine.
 - If the user element includes viscous effects but no inertia terms, the same approach can be used as for transient heat transfer analysis. The force vector then should contain the term $-C^{NM} \Delta u^M / \Delta t$, and the term $C^{NM} / \Delta t$ must be added to the stiffness.
 - In that case the α parameter can again be ignored.

LFLAGS (3) =5

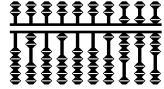
Half step residual calculation, which is needed only for automatic time incrementation.

- Only the force vector must be supplied, which has the form

$$F^N = M^{NM} \ddot{u}_{t+\Delta t/2} + (1 + \alpha) G_{t+\Delta t/2}^N - \frac{\alpha}{2} (G_t^N + G_{t^0}^N),$$

where $G_{t^0}^N$ is the static residual at the beginning of the previous increment.

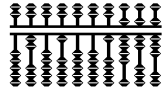
- $G_{t^0}^N$ must be stored as a solution-dependent state vector.
- The vector $\ddot{u}_{t+\Delta t/2}$ is passed into the subroutine, $G_{t+\Delta t/2}^N$, and G_t^N must be calculated.
- It is obvious that this expression simplifies considerably if $\alpha=0$.



LFLAGS (3) =4

Velocity jump calculation, which will be done at the start of each dynamic step and after contact changes.

- The purpose of this calculation is to make the velocities conform to constraints imposed by *MPC, *EQUATION, or contact conditions while preserving momentum.
- The Jacobian is equal to the mass matrix, and the force vector should be set to zero.

**LFLAGS (3) = 6**

Acceleration calculation, which will be done at the start of each dynamic step (unless INITIAL=NO on the *DYNAMIC option) and after contact changes.

- The purpose of this calculation is to create dynamic equilibrium at the start of a step or after contact changes.
- The Jacobian is equal to the mass matrix, and the force vector should contain static and damping contributions only.

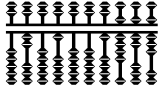
Remarks

Implementation of a user element with inertia effects in a dynamic analysis is fairly complicated. Simplifications to the **UEL** can be realized if:

- The ALPHA parameter on the *DYNAMIC option is set to zero.
- No inertia effects are included in the user element.
 - Inertia effects can be included “indirectly” by overlaying standard ABAQUS elements on top of user elements.

In this case, the ABAQUS elements should have negligible stiffness.

For example, it is possible to overlay B23 elements on top of the beam elements with nonlinear section behavior shown in the first example.



Workshop Preliminaries

Setting Up the Workshop Directories and Files

If you are taking this seminar in an HKS office, the steps in the following section have already been done for you: skip to **Basic Operating System Commands** (p. WP.3). If everyone in your group is familiar with the operating system, skip directly to the workshops.

The workshop files are included on the ABAQUS release CD. If you have problems finding the files or setting up the directories, ask your systems manager for help.

Note for systems managers: If you are setting up these directories and files for someone else, please make sure that there are appropriate privileges on the directories and files so that the user can write to the files and create new files in the directories.

Workshop File Setup

(Note: UNIX is case-sensitive. Therefore, lowercase and uppercase letters must be typed as they are shown or listed.)

1. Find out where the ABAQUS release is installed by typing

UNIX and Windows NT: **abqxxx whereami**

where **abqxxx** is the name of the ABAQUS execution procedure on your system. It can be defined to have a different name. For example, the command for the 6.2-1 version might be aliased to **abq621**.

This command will give the full path to the directory where ABAQUS is installed, referred to here as *abaqus_dir*.

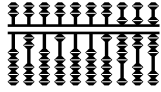
2. Extract all the workshop files from the course tar file by typing

UNIX: *abaqus_dir/exec/perl*
 abaqus_dir/samples/course_setup.pl

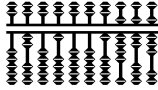
Windows NT: *abaqus_dir\exec\Perl*
 abaqus_dir\samples\course_setup.pl

An alternative method is to edit the script **course_setup.pl** and change the first line of the script to the local installation of the Perl interpreter or the one available in *abaqus_dir/exec*. For example:

#!/usr/bin/perl becomes **#!/abaqus_dir/exec/perl**



3. The script will install the files into the current working directory. You will be asked to verify this and to choose which files you wish to install. Choose “**y**” for the appropriate lecture series when prompted. Once you have selected the lecture series, type “**q**” to skip the remaining lectures and to proceed with the installation of the chosen workshops.



Basic Operating System Commands

(You can skip this section and go directly to the workshops if everyone in your group is familiar with the operating system.)

Note: The following commands are limited to those necessary for doing the workshop exercises.

Working with Directories

1. Start in the current working directory. List the directory contents by typing

UNIX: **ls**

Windows NT: **dir**

Both subdirectories and files will be listed. On some systems the file type (directory, executable, etc.) will be indicated by a symbol.

2. Change directories to a workshop subdirectory by typing

Both UNIX and Windows NT: **cd** *dir_name*

3. To list with a long format showing sizes, dates, and file, type

UNIX: **ls -l**

Windows NT: **dir**

4. Return to your home directory:

UNIX: **cd**

Windows NT: **cd** *home-dir*

List the directory contents to verify that you are back in your home directory.

5. Change to the workshop subdirectory again.
6. The ***** is a wildcard character and can be used to do a partial listing. For example, list only ABAQUS input files by typing

UNIX: **ls *.inp**

Windows NT: **dir *.inp**

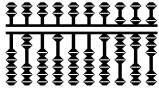
Working with Files

Use one of these files, *filename.inp*, to perform the following tasks:

1. Copy *filename.inp* to a file with the name **newcopy.inp** by typing

UNIX: **cp filename.inp newcopy.inp**

Windows NT: **copy filename.inp newcopy.inp**



2. Rename (or move) this new file to **newname.inp** by typing

UNIX: **mv newcopy.inp newname.inp**

Windows NT: **rename newcopy.inp newname.inp**

(Be careful when using **cp** and **mv** since UNIX will overwrite existing files without warning.)

3. Delete this file by typing

UNIX: **rm newname.inp**

Windows NT: **erase newname.inp**

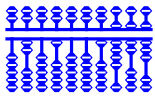
4. View the contents of the files *filename.inp* by typing

UNIX: **more filename.inp**

Windows NT: **type filename.inp | more**

This step will scroll through the file one page at a time.

Now you are ready to start the workshops.



Workshop 1

User Subroutine FILM

Goals

- To learn how to find the compile and link commands used on your system.
- To see how sensitive the rate of convergence is on the value of $\frac{dh}{d\theta}$.
- To see if the results are sensitive to the value of $\frac{dh}{d\theta}$.

Problem Description

User subroutine **FILM** will be used to define

$$h(\theta) = 500|\theta_w - \theta_i^s|^{1/3}, \quad (\text{EQ 1})$$

where $\theta_i^s = 100^\circ\text{C}$ is the fluid temperature, and θ_w is the surface temperature.

The subroutine will be tested on a two-element model, which is shown in Figure W1–1. All nodes are initially at 77°C .

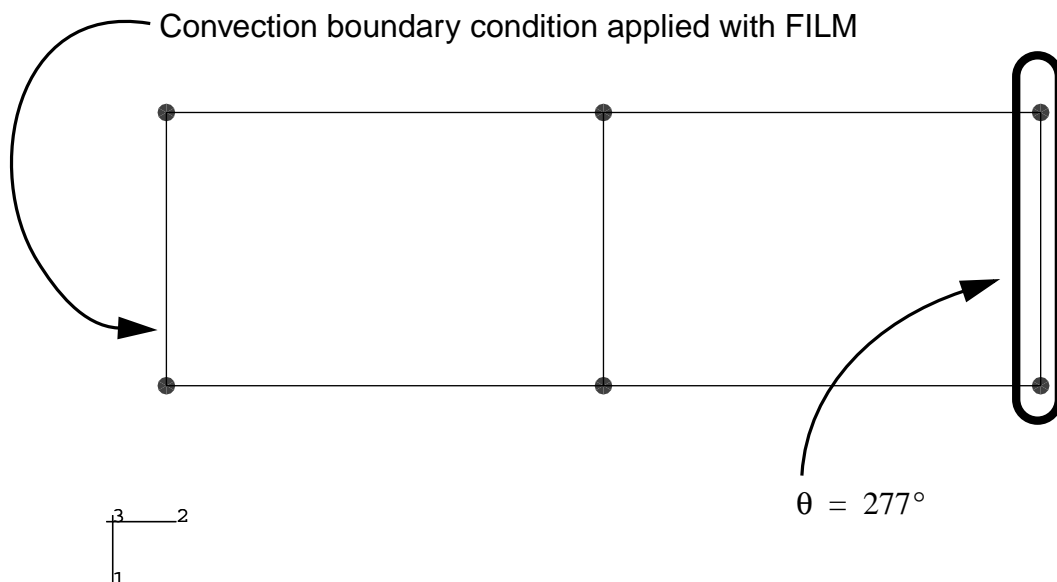
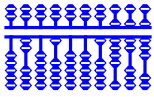


Figure W1–1. Two-Element Model



Steady-State Solution: Correct Model

The input file **film_test-1.inp** calculates the steady-state solution for the problem shown in Figure W1-1.

1. Look at the file **film_test-1.f**. It is in the directory **user_subroutines/film**.

This file contains the correct format for h and $\frac{dh}{d\theta}$.

Question W1-1: Which variable is assigned the value of $\frac{dh}{d\theta}$?

Question W1-2: What is the derivative of the absolute value of a variable; that is, what is $\frac{dy}{dx}$ when $y = |x|$?

2. Submit the input file **film_test-1.inp** to ABAQUS. The command to submit the file is

```
abaqus j=film_test-1 user=film_test-1.f
```

By default on our system, the analysis will run as a background process. Look at the contents of the **film_test-1.log** file to see how the analysis is progressing.

3. Look at the number of iterations ABAQUS needed for each increment of the analysis; this information can be found in either the status (**.sta**) or the message (**.msg**) files. Enter the total number of iterations that ABAQUS needed during the analysis to obtain the solution in Table W1-1.

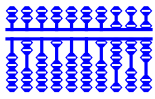
Question W1-3: Where is the easiest place to find the total number of iterations used during an analysis?

4. If you want to provide the functionality of a user subroutine to another user, but you do not want to provide them with the source code, you will have to compile the user subroutine prior to running ABAQUS. However, to do this, you need to know what the proper compile command is for your system.

Question W1-4: How can you find out the proper compile command for your system?

Optional:

1. Once you find the proper compile commands for your system, try precompiling the user subroutine.
2. Create symbolic links to the **ABA_PARAM.INC** file. The syntax to create this link is found in the communications (**.com**) file; it will vary from system to system.



3. Submit the input file and the precompiled user subroutine to ABAQUS:
`abaqus j=film_test-2 input=film_test-1 user=film_test-1.o`
4. Compare the wall clock times for the two simulations. How much time was saved?

Steady-State Solution: Model with Error 1

1. Submit the input file `film_test-e1.inp` to ABAQUS using the subroutine in `film_test-e1.f`.

This model contains an error in the definition of $\frac{dh}{d\theta}$:

```
h(2) = third*500.0d0*(abs(temp-sink))**(-two*third)
```

The correct definition of `h(2)` should contain a term, `a1`, which is defined by the sign of `temp-sink`:

```
a1 = sign(1.0, temp-sink)
```

```
h(2) = a1*third*500.0d0*(abs(temp-sink))**(-two*third)
```

Question W1-5: Do you think this is a significant error?

2. Look at the number of iterations ABAQUS needed for each increment of the analysis. Enter the total number of iterations that ABAQUS needed during the analysis to obtain a solution in Table W1-1.
3. Compare the final temperature values from this analysis with those from the `film_test-1` analysis.

Steady-State Solution: Model with Error 2

1. Submit the input file `film_test-e2.inp` to ABAQUS using the subroutine in `film_test-e2.f`

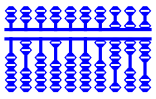
This model contains an error in the definition of $\frac{dh}{d\theta}$:

```
h(2) = third*500.0d0*(abs(temp-sink))**(-third)
```

The correct definition of `h(2)` should have the term `(abs(temp-sink))` raised to the $-2/3$ power:

```
h(2) = a1*third*500.0d0*(abs(temp-sink))**(-two*third)
```

2. Look at the number of iterations ABAQUS needed for each increment of the analysis. Enter the total number of iterations that ABAQUS needed during the analysis to obtain a solution in Table W1-1.



3. Compare the final temperature values from this analysis with those from the `film_test-1` analysis.

Question W1-6: Were there any differences in the results? Were they significant?

Steady-State Solution: Model with Error 3

1. Submit the input file `film_test-e3.inp` to ABAQUS using the subroutine in `film_test-e3.f`.

This model contains an error in the definition of $\frac{dh}{d\theta}$:

```
h(2) = third*500.0d0*(abs(temp-sink))**(two*third)
```

The correct definition of `h(2)` should have the term `(abs(temp-sink))` raised to the $-2/3$:

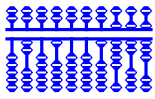
```
h(2) = a1*third*500.0d0*(abs(temp-sink))**(-two*third)
```

2. Look at the number of iterations ABAQUS needed for each increment of the analysis. Enter the total number of iterations that ABAQUS needed during the analysis to obtain a solution in Table W1-1.
3. The analysis has failed to converge.

Question W1-7: Why does the analysis fail to converge?

Table W1-1

Model	Number of Iterations
<code>film_test-1.inp</code>	
<code>film_test-e1.inp</code>	
<code>film_test-e2.inp</code>	
<code>film_test-e3.inp</code>	



Answers 1

User Subroutine FILM

Question W1-1: Which variable is assigned the value of $\frac{dh}{d\theta}$?

Answer: The second position in the array **H**, **H(2)**, is given the value.

Question W1-2: What is the derivative of the absolute value of a variable; that is, what is $\frac{dy}{dx}$ when $y = |x|$?

Answer: The derivative of the absolute value of a variable is either -1 or $+1$ depending on the sign of the variable.

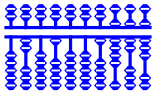
Question W1-3: Where is the easiest place to find the total number of iterations used during an analysis?

Answer: The easiest place to find the total number of iterations is in the summary at the end of the message file.

ANALYSIS SUMMARY:

TOTAL OF	1	INCREMENTS
	5	CUTBACKS IN AUTOMATIC INCREMENTATION
	45	ITERATIONS
	45	PASSES THROUGH THE EQUATION SOLVER OF WHICH
	45	INVOLVE MATRIX DECOMPOSITION, INCLUDING
	0	DECOMPOSITION(S) OF THE MASS MATRIX
	0	ADDITIONAL RESIDUAL EVALUATIONS FOR LINE SEARCHES
	0	ADDITIONAL OPERATOR EVALUATIONS FOR LINE SEARCHES
	0	WARNING MESSAGES DURING USER INPUT PROCESSING
	0	WARNING MESSAGES DURING ANALYSIS
	0	ANALYSIS WARNINGS ARE NUMERICAL PROBLEM MESSAGES
	0	ANALYSIS WARNINGS ARE NEGATIVE EIGENVALUE MESSAGES
	1	ERROR MESSAGES

THE SPARSE SOLVER HAS BEEN USED FOR THIS ANALYSIS.



The status file does not provide a total number of iterations used in an analysis if there are any cutbacks in the increment size and may not provide the correct number if there are any severe discontinuity iterations.

Question W1-4: How can you find out the proper compile command for your system?

Answer: You could look in the communications (**.com**) file, or you could use the **abaqus help=environment** command. If you look in the communications file, you will find the following lines:

```
ln -s /usr/abaqus60/abqver/sgi4000/../../../../release/sgi4000/
r5-07-001/site/aba_param_dp.inc aba_param.inc
```

```
ln -s /usr/abaqus60/abqver/sgi4000/../../../../release/sgi4000/
r5-07-001/site/aba_param_dp.inc ABA_PARAM.INC
```

```
.
.
.
```

```
echo Compiling... film_test-e3.f
```

```
f77 -G 0 -mp -c -mips2 -32 -O2 film_test-e3.f
```

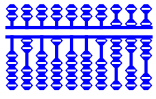
The symbolic links (**ln -s** commands) are needed to include the **ABA_PARAM.INC** file during the compile command. It is important to use all the flags and settings on the compile command that are found by either method.

Question W1-5: Do you think this is a significant error?

Answer: It is not likely to be a significant error because the sign of the derivative will always be positive in this example ($\theta_w - \theta_i^s > 0$).

Question W1-6: Were there any differences in the results? Were they significant?

Answer: There were differences between the two analyses; however, the differences were minor errors in the fourth significant figure of the numbers.



Question W1-7: Why does the analysis fail to converge?

Answer: The analysis fails to converge because the rate of convergence is very slow and ABAQUS continues to cut back the increment size. The rate of convergence is slow because the value calculated for $\frac{dh}{d\theta}$ contains a large error.

Workshop 2

User Subroutine UMAT: Tangent Stiffness

Goals

- To investigate the effect of the tangent stiffness matrix on convergence.
- To see how a poorly developed material model can cause problems in simulations that are under load control.

Problem Description

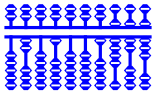
The tangent stiffness matrix calculated in user subroutine **UMAT** plays a critical role in the Newton-Raphson algorithm in ABAQUS/Standard. Errors in the formulation of the tangent stiffness matrix will result in analyses that require more iterations and, in some cases, diverge. In this workshop the importance of calculating tangent stiffness correctly is explored by analyzing a small problem under uniaxial tension.

Both displacement and load control simulations are considered. The material response being modeled is elastic-plastic; Mises plasticity is assumed.

The analysis is carried out by modifying the material stiffness calculated with user subroutine **UMAT**, which effectively introduces an error into the tangent stiffness matrix. To build confidence in user subroutine **UMAT**, the problem is first solved using the built-in Mises plasticity algorithm in ABAQUS/Standard.

Analyses with the *PLASTIC Material Model

1. Submit the file **umat/axi1.inp** to ABAQUS. This input file is set up for a displacement-control analysis and uses the Mises plasticity routine that is built into ABAQUS.



2. Enter the number of increments and the total number of iterations required to complete this displacement control analysis in Table W2–1. Under the heading “Flag Setting” enter “built in.”

Table W2–1

Displacement Control		
Flag Setting	Number of Increments	Number of Iterations

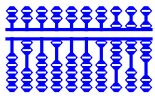
Load Control		
Flag Setting	Number of Increments	Number of Iterations

3. Copy **axi1.inp** into another file name, and modify it as follows: remove the nonzero displacement boundary conditions, and add a negative pressure loading of 40000 on the element in the axial direction.
4. Submit the new file to ABAQUS.
5. Repeat Step 2, except put the results in the load control portion of the table.

Displacement-Control Analyses with the UMAT Material Model

Now solve the same problem using user subroutine **UMAT**.

- The material constants used in the user subroutine are specified with the *USER MATERIAL option.
- Seven material constants are specified in this problem.
- The seventh material constant is used as a flag in user subroutine **UMAT**: when it is set to 1, **UMAT** returns the actual (i.e., correct) material stiffness; when it is set to 0, **UMAT** returns only the elastic stiffness.



1. The file **umat/axi2.inp** contains the input data for the model, and the file **umat/iso_mises_umat.f** contains the source code for the **UMAT**. The file **axi2.inp** is set up for a displacement-control analysis; the seventh material constant is set to 1, which provides ABAQUS/Standard with the correct tangent stiffness.
2. Submit the input file.
3. Compare the results of this analysis with those obtained for the displacement-control analysis using the built-in Mises plasticity routine.
4. Enter the flag setting, number of increments, and the total number of iterations required to complete the analysis in the displacement-control section of Table W2–1.
5. Modify the material constants so that the seventh constant under the *USER MATERIAL option is set to 0, which provides ABAQUS/Standard with the incorrect tangent stiffness.
6. Repeat Step 2–Step 4.

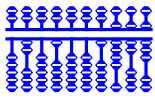
Question W2–1: Are the results obtained with the modified stiffness matrix correct?

Load-Control Analyses with the UMAT Material Model

Now repeat the analysis under a state of load control.

1. Copy **axi2.inp** into another file name, and modify it as follows: remove the nonzero displacement boundary conditions, and add a negative pressure loading of 40000 on the element in the axial direction.
2. Set the seventh material constant to 1.
3. Submit the job. Enter the flag setting, number of increments, and the total number of iterations required to complete the analysis in Table W2–1.
4. Repeat Step 3 when the seventh material constant is set to 0.

Question W2–2: What can you say about the difference in the convergence behavior of this problem when a pressure loading is applied instead of a boundary condition?



Answers 2

User Subroutine UMAT: Tangent Stiffness

Question W2-1: Are the results obtained with the modified stiffness matrix correct?

Answer: Yes. Errors in the tangent stiffness will only affect the convergence behavior, not the final result. When the elastic stiffness is used instead of the true stiffness, the number of iterations is increased. The results (when obtained) are still correct.

Question W2-2: What can you say about the difference in the convergence behavior of this problem when a pressure loading is applied instead of a boundary condition?

Answer: Displacement-control problems are more stable than load-control problems. When the elastic stiffness is used instead of the true stiffness, the displacement-control analysis ran successfully but the load-control analysis failed.

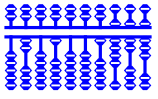


Table W2–1

Displacement Control		
Flag Setting	Number of Increments	Number of Iterations
built in	6	8
1	6	8
0	6	21

Load Control		
Flag Setting	Number of Increments	Number of Iterations
built in	9	28
1	9	28
0	6*	59*

*Did not run to completion.