APPENDIX A: User-Defined Materials

GETTING STARTED WITH USER-DEFINED FEATURES

We begin with a general introduction to user-defined features. This section is valid for any user-defined feature described in the remaining appendices and serves as a practical guide to enable you to begin coding. We will specifically discuss the different methods for linking your user-defined subroutines to LS-DYNA. For a comprehensive overview of user-defined features and their applications, please refer to Erhart [2010], which can be seen as a complement to the present text.

Download

First, download a *usermat version* of LS-DYNA for the computer architecture/platform of interest. This version is a compressed package (.tgz, .tar.gz, .zip) provided by Ansys or your local LS-DYNA distributor. The unpacked content contains a usermat directory, possibly including

- precompiled static object libraries (.a, .lib)
- Fortran source code files (.f, .f90)
- include files (.inc)
- makefile
- a shared library binary LS-DYNA executable

Choosing a version to download that is compatible with your computer environment in terms of architecture, operating system, and possible MPI implementation for MPP is essential. Selecting an appropriate version may not be obvious. For questions regarding the version, contact your LS-DYNA distributor. A usermat version does not require a special license but falls under the general LS-DYNA license agreement.

Static or dynamic linking

To get something that is runnable, it is necessary to *compile* the source code files and link the resulting object files either to a binary executable (see Figure 51-1) or a *shared object* (see Figure 51-2). The latter option requires a shared library version of the usermat package and a sharelib binary, while the former assumes a statically linked usermat package. Thus, this choice needs to be made before retrieving the package. Working with shared objects is more flexible because a binary executable can be installed once and for all, and the (small) shared object is dynamically linked at runtime. Upon execution, LS-DYNA will look for the shared object file in directories specified by a library path that is set either by the shell variable LD_LIBRARY_PATH, on the command line with module=, or by the *MODULE_LOAD keyword. The shared object is easily substituted, facilitating portability when working on large projects. A statically linked version, on the other hand, requires generating the entire binary executable during compilation. This option may be preferred if working on small projects or during the development phase.

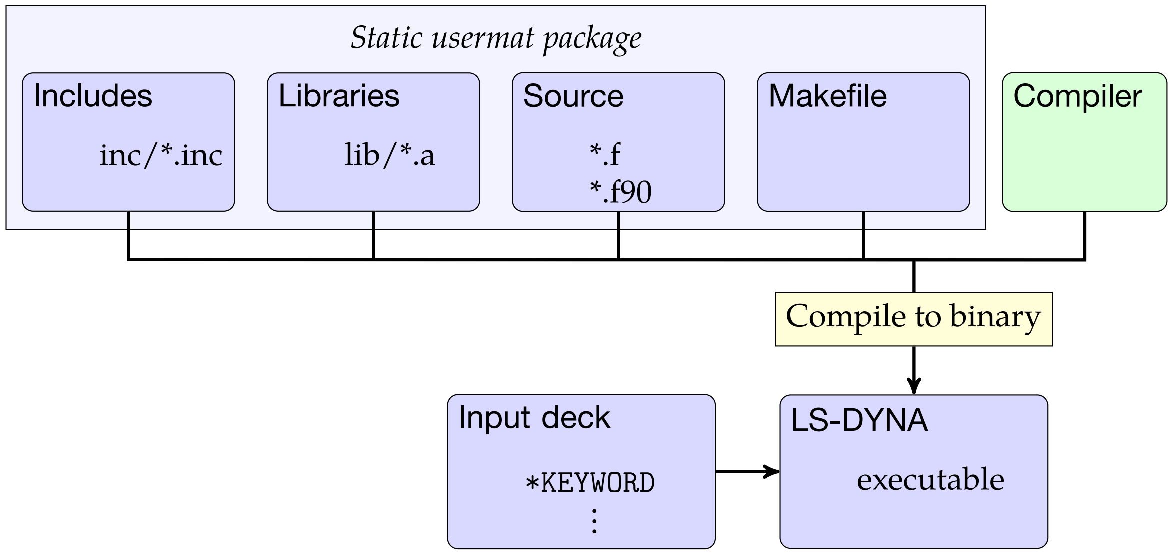

Figure 51-1. Statically linked usermat package

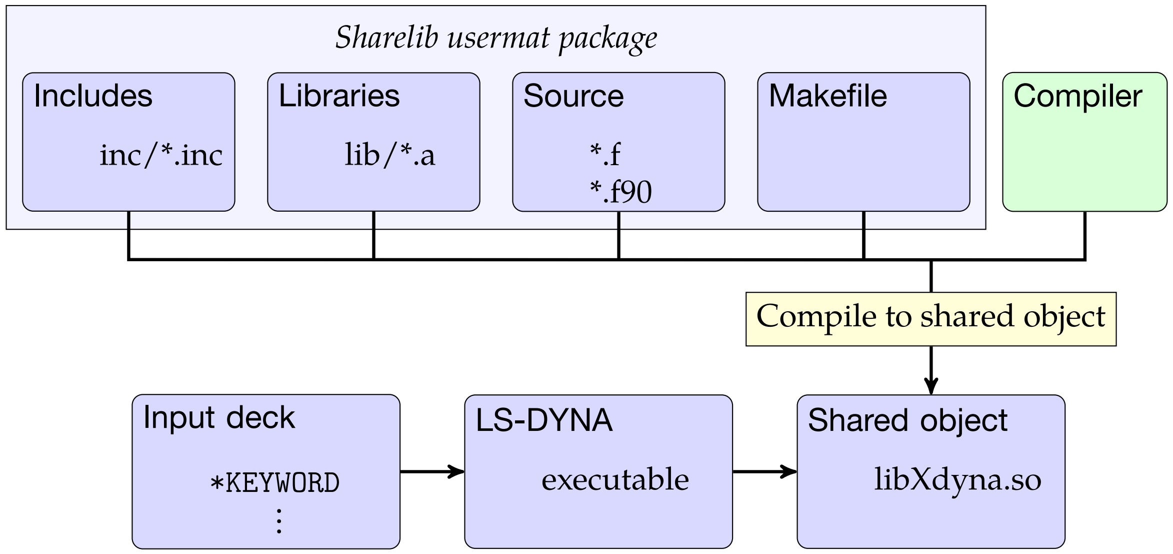

Figure 51-2. Dynamically linked usermat package

Compiler and compiling

The compilation is usually performed in the usermat (working) directory using a terminal window. The usermat package does not contain a compiler; it is assumed that the appropriate compiler is installed on your system and accessible from your working directory. To render compatibility between the pre-compiled libraries or binary executable and your compiled object files, the appropriate compiler is the (by LS-DYNA team) designated compiler with which the pre-compiled libraries or binary are readily compiled.

For example, on Linux, this is typically either the Intel Fortran Compiler (IFC) on Red Hat or CentOS or Portland Groups Compiler (PGF) on SUSE; on Windows, it is the Intel Fortran Compiler (IFC) in combination with Microsoft Visual Studio (MSVS). For MPP, you also need a wrapper for whatever MPI implementation you have installed, but again, consult your distributor for detailed information.

To compile, execute '`make clean; make`' in the terminal window. In most cases, these commands will generate a shared object or binary executable depending on your type of LS-DYNA usermat version. If not, it comes down to interpreting error messages. A possible reason for failure is that the included makefile does not contain appropriate compiler directives and requires some editing. This could range from trivial tasks like altering the path used to find the compiler on your system to more complex endeavors such as adding or changing compiler flags. An even worse scenario is if your operating system is not up-to-date and may require updating your computer's library content. Preferably, your computer administrator resolves such scenarios in collaboration with your distributor.

Execution

When compiling a virgin instance of a usermat version, the generated executable (for static usermat) or the shared object combined with a sharelib executable (for sharelib usermat) form an exact replica of a regular executable. A customized version only generates when the source code is modified. The binary and/or shared object may be left in the working directory, moved to some other location on your system, or properly installed for execution on clusters by a queuing system. While both the executable and shared object may be renamed, a shared object should not be renamed if you use LD_LIBARY_PATH to point to its location. During the development phase, it may be convenient to leave the binary in its place to facilitate debugging but move the shared object to the execution directory to not have to edit the library path for the executable to find it. Two Linux examples on how to run an input file are given below, one assuming an SMP static object version has been downloaded and the other an MPP shared object version. Let `/path_to_my_source/usermat` be the complete path to the working directory and in.k be the name of the keyword input file. If located in the *execution* directory, meaning the directory containing in.k, the problem is run as

```
/path_to_my_source/usermat/lsdyna i=in.k
```

in the SMP case. For the MPP shared object case, you may copy the shared object file to the execution directory, the default directory where executables can look for dynamic object files. Assuming the MPI software is platform-MPI, the input file could be run on two cores as

```
/path_to_platform-mpi/bin/mpirun -np 2  /path_to_my_source/usermat/mppdyna i=in.k
```

These two examples are only intended to indicate how to treat the compiled files, and changes in details thereof are to be expected.

APPENDIX A

Coding

All subroutines for the user-defined features are collected in the files dyn21*.f and are ready for editing using your favorite text editor. Mechanical user materials are contained in dyn21umat*.f; user-defined loads in dyn21.f; user-defined contacts, wear, and friction in dyn21cnt.f; user-defined elements in dyn21usld.f and dyn21ushl.f; and thermal user materials are in dyn21tumat.f, just to give a few examples. The prevailing programming language is Fortran 77, but many compilers support Fortran 90. Writing C code may also be possible, but using C requires some manipulation of the makefile and function interfaces. Each feature is connected to one or a few keywords to properly take advantage of innovative coding. For user materials, this keyword is *MAT_USER_DEFINED_MATERIAL_MODELS and is described in detail below, and for user-defined loads, the corresponding keyword is *USER_LOADING. We refer to the keyword manual pages for details on their respective usage.

Module concept

Here, we delve further into the dynamically linked, shared object/library usermat version and the module concept. A share object usermat package only consists of a makefile and a few source code files. In principle, all you need to do is to (*i*) implement the feature of interest in the source code files, (*ii*) compile to a shared object using the makefile, and (*iii*) use the keyword input file to access the generated object.

There are several advantages to using shared objects:

1. The content and size of a shared object usermat package is significantly reduced compared to a statically compiled usermat. Also, the size of a shared object is small.

2. Several shared objects can be used simultaneously, expanding the number of available user features in a single model.

3. Shared objects can be delivered to or by third parties.

We will illustrate how LS-DYNA, in practice, accesses shared objects and, at the same time, addresses items 2 and 3 above with an example. Due to the modular nature of shared objects, they can also be called *modules*. We will describe how to access these using the *MODULE keyword.

In a shared library version, the keywords

*MODULE_PATH

*MODULE_LOAD

*MODULE_USE

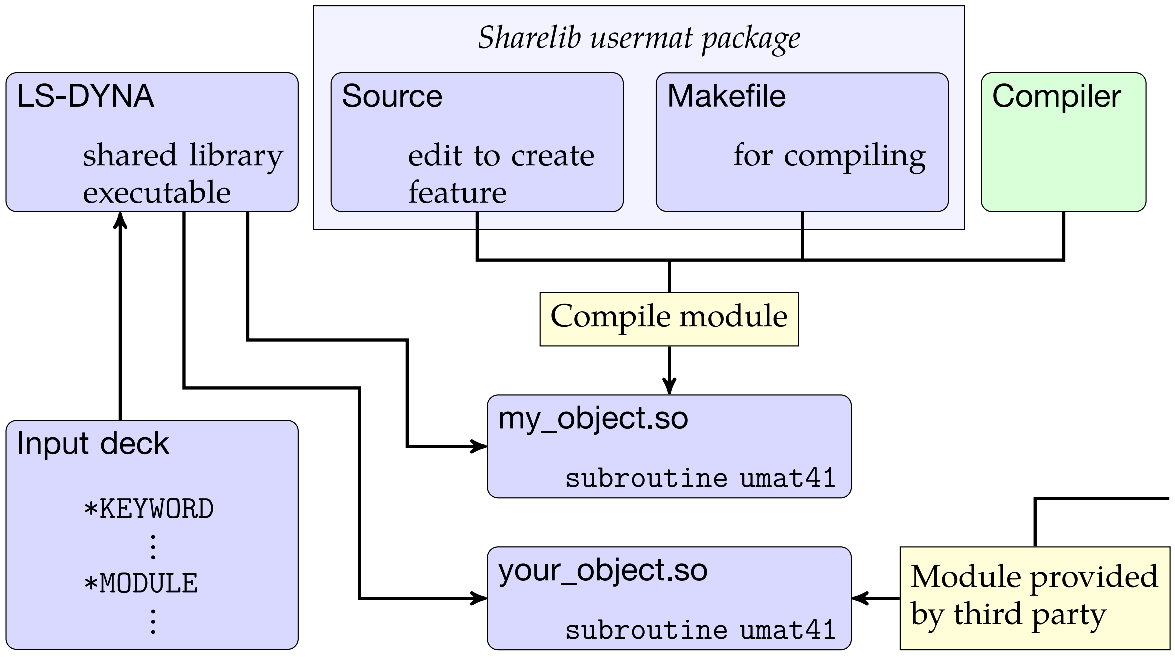

Figure 51-3. Dynamic linking with modules

are available. See the section on *MODULE for more detailed explanations of these keywords. With reference to Figure 51-3, we assume that my_object.so and your_object.so are two independently generated modules, and both are located in directory `/path_to_modules`. We also assume that these two modules contain different implementations for user material 41 (`subroutine umat41`). Without the present approach, executing both implementations with the same executable and keyword input would be at least intricate. A simple way of dealing with this using modules is to use the following set of keywords:

```
*MODULE_PATH
/path_to_modules
*MODULE_LOAD
$ MDLID                 TITLE
myid                    my library
$ FILENAME
my_object.so
*MODULE_LOAD
$ MDLID                 TITLE
yourid                  your library
$ FILENAME
your_object.so
*MODULE_USE
$ MDLID
myid
$ TYPE                  PARAM1                  PARAM2
UMAT                    1001                    41
*MODULE_USE
$ MDLID
```

```
yourid
$ TYPE                PARAM1                PARAM2
UMAT                  1002                  41
*MAT_USER_DEFINED_MATERIAL_MODELS
$       MID         RO         MT
          1   7.85e-9       1001
…
*MAT_USER_DEFINED_MATERIAL_MODELS
$       MID         RO         MT
          2   7.85e-9       1002
…
*PART
first part
$       PID      SECID        MID
          1          1          1
*PART
second part
$       PID      SECID        MID
          2          2          2
```

Here, *MODULE_PATH lists the path(s) to the shared objects to be loaded, *MODULE_-LOAD loads the dynamic libraries, and *MODULE_USE tells the executable how to access routines in the shared object. In this particular example, a user material (*MAT_-USER_...) with MT set to 1001 (because PARAM1 = 1001) will execute `subroutine umat41` (because PARAM2 = 41) from the module with ID myid (because MDLID=myid). Similarly, user material with MT set to 1002 will also execute subroutine umat41 but now from the module with ID yourid. Hence, we have made parts 1 and 2 execute the *same* subroutine (by name) but from *different* modules. This generalizes to any number of modules by analog. See *MODULE_USE for more details.

GENERAL OVERVIEW OF USER-DEFINED MATERIALS

We now turn to the specific discussion of user-defined materials. You can simultaneously implement up to ten user subroutines to update the stresses in solids, shells, beams, discrete beams, and truss beams. This section serves as an introductory guide to implementing such a model. Note that the names of variables and subroutines below may differ from the actual ones depending on the platform and version of LS-DYNA.

When the keyword *MAT_USER_DEFINED_MATERIAL_MODELS is defined for a part in the keyword deck, LS-DYNA calls the subroutine `usrmat` with appropriate input data for the constitutive update. This routine calls `urmathn` for 2D and 3D solid elements, `urmats` for 2D plane stress and 3D shell elements, `urmatb` for beam elements, `urmatd` for discrete beam elements, and `urmatt` for truss beam elements. You may modify these routines if necessary. These routines initialize the following data structures:

`sig(6)`	–	stresses in the previous time step
`eps(6)`	–	strain increments

`epsp` –	effective plastic strain in the previous time step
`hsv(*)` –	history variables in the previous time step, excluding plastic strain
`dt1` –	current time step size
`temper` -	current temperature
`failel` –	flag indicating failure of element

A specific *scalar* material subroutine receives these data structures. Suppose you activate the vectorization flag (IVECT = 1 on the material card). In that case, the program stores variables in vector blocks of length `nlq`, with vector indexes ranging from `lft` to `llt`. This storage method allows for more efficient execution of the material routine. As an example, the data structures mentioned above are exchanged in the vectorized case for

`sig`*`X`*`(nlq)` –	stresses in the previous time step
`d`*`X`*`(nlq)` –	strain increments
`epsps(nlq)` –	effective plastic strains in the previous time step
`hsvs(nlq,*)` –	history variables in the previous time step
`dt1siz(nlq)` -	current time step sizes
`temps(nlq)` –	current temperatures
`failels(nlq)` –	flags indicating failure of elements

where *`X`* ranges from 1 to 6 for the different components. Each entry in a vector block is associated with an element in the finite element mesh for a fixed integration point.

The number of entries in the history variables array (indicated by `*` above) matches the number of history variables requested on the material card (NHV). Hence, the number NHV should equal the number of history variables excluding the effective plastic strain. We give effective plastic strain special treatment. All history variables, including the effective plastic strain, are initially zero. Furthermore, all user-defined material models require bulk and shear moduli for transmitting boundaries, contact interfaces, rigid body constraints, and time-step calculations. Thus, the length of the material constants array, `LMC`, must, generally, be increased by two to store these parameters. In addition to the variables mentioned above, user-material routines may receive the following data, regardless of whether vectorization is used or not:

`cm(*)` –	material constants array
`capa` –	transverse shear correction factor for shell elements
`tt` –	current time
`crv(lq1,2,*)` –	array representation of curves defined in the keyword deck

Next, the previously mentioned element-specific routine calls a specific material routine, `umat`*`XX`* in the scalar case or `umat`*`XX`*`v` in the vector case. *`XX`* is a number between 41 and 50 and matches MT on the material card. You write the specific material subroutine. This subroutine should update the stresses and history variables to the current time. For shells and beams, your subroutine must also determine the strain increments in the directions of constrained zero stress. To be able to write different stress updates for distinct element types, your subroutine receives the following character string:

etype – character string that equals `solid`, `shell`, `beam`, `dbeam`, or `tbeam`

A sample user subroutine of a hypoelastic material in the scalar case is provided below. This sample and the others below are from the dyn21.F file distributed with version R6.1.

Sample user subroutine 41

```
      subroutine umat41 (cm,eps,sig,epsp,hsv,dt1,capa,etype,tt,
     1 temper,failel,crv,nnpcrv,cma,qmat,elsiz,idele,reject)
c
c******************************************************************
c|  Livermore Software Technology Corporation  (LSTC)             |
c|  ------------------------------------------------------------  |
c|  Copyright 1987-2008 Livermore Software Tech.  Corp             |
c|  All rights reserved                                           |
c******************************************************************
c
c     isotropic elastic material (sample user subroutine)
c
c     Variables
c
c     cm(1)=first material constant, here Young's modulus
c     cm(2)=second material constant, here Poisson’s ratio
c        .
c        .
c        .
c     cm(n)=nth material constant
c
c     eps(1)=local x  strain increment
c     eps(2)=local y  strain increment
c     eps(3)=local z  strain increment
c     eps(4)=local xy strain increment
c     eps(5)=local yz strain increment
c     eps(6)=local zx strain increment
c
c     sig(1)=local x  stress
c     sig(2)=local y  stress
c     sig(3)=local z  stress
c     sig(4)=local xy stress
c     sig(5)=local yz stress
c     sig(6)=local zx stress
c
c     hsv(1)=1st history variable
c     hsv(2)=2nd history variable
c        .
c        .
c        .
c        .
c     hsv(n)=nth history variable
c
c     dt1=current time step size
c     capa=reduction factor for transverse shear
c     etype:
c        eq."solid" for solid elements
c        eq."sld2d" for shell forms 13, 14, and 15 (2D solids)
c        eq."shl_t" for shell forms 25, 26, and 27 (shells with thickness
c         stretch)
c        eq."shell" for all other shell elements plus thick shell forms 1
c         and 2
c        eq."tshel" for thick shell forms 3 and 5
c        eq."hbeam" for beam element forms 1 and 11
```

```
c        eq."tbeam" for beam element form 3 (truss)
c        eq."dbeam" for beam element form 6 (discrete)
c        eq."beam " for all other beam elements
c
c     tt=current problem time.
c
c     temper=current temperature
c
c     failel=flag for failure, set to .true.  to fail an integration point,
c            if .true.  on input, the integration point has failed earlier
c
c     crv=array representation of curves in keyword deck
c
c     nnpcrv=# of discretization points per crv()
c
c     cma=additional memory for material data defined by LMCA at
c       6th field of 2nd card of *DATA_USER_DEFINED
c
c     elsiz=characteristic element size
c
c     idele=element id
c
c     reject (implicit only) = set to .true.  if this implicit iterate is
c                              to be rejected for some reason
c
c     All transformations into the element’s local system are
c     performed before entering this subroutine.  Transformations
c     back to the global system are performed after exiting this
c     routine.
c
c     All history variables are initialized to zero in the input
c     phase.  Initialization of history variables to nonzero values
c     may be done during the first call to this subroutine for each
c     element.
c
c     Energy calculations for the dyna3d energy balance are done
c     outside this subroutine.
c
      include 'nlqparm'
      include 'bk06.inc'
      include 'iounits.inc'
      dimension cm(*),eps(*),sig(*),hsv(*),crv(lq1,2,*),cma(*)
      integer nnpcrv(*)
      logical failel,reject
      character*5 etype
c
      if (ncycle.eq.1) then
        if (cm(16).ne.1234567) then
          call usermsg('mat41')
        endif
      endif
c
c     compute shear modulus, g
c
      g2 =abs(cm(1))/(1.+cm(2))
      g  =.5*g2
c
      if (etype.eq.'solid'.or.etype.eq.'shl_t'.or.
     1     etype.eq.'sld2d'.or.etype.eq.'tshel') then
        if (cm(16).eq.1234567) then
          call mitfail3d(cm,eps,sig,epsp,hsv,dt1,capa,failel,tt,crv)
        else
          if (.not.failel) then
          davg=(-eps(1)-eps(2)-eps(3))/3.
          p=-davg*abs(cm(1))/(1.-2.*cm(2))
          sig(1)=sig(1)+p+g2*(eps(1)+davg)
          sig(2)=sig(2)+p+g2*(eps(2)+davg)
          sig(3)=sig(3)+p+g2*(eps(3)+davg)
          sig(4)=sig(4)+g*eps(4)
          sig(5)=sig(5)+g*eps(5)
          sig(6)=sig(6)+g*eps(6)
          if (cm(1).lt.0.) then
            if (sig(1).gt.cm(5)) failel=.true.
          endif
          endif
        end if
c
      else if (etype.eq.'shell') then
        if (cm(16).eq.1234567) then
          call mitfailure(cm,eps,sig,epsp,hsv,dt1,capa,failel,tt,crv)
        else
          if (.not.failel) then
          gc    =capa*g
          q1    =abs(cm(1))*cm(2)/((1.0+cm(2))*(1.0-2.0*cm(2)))
          q3    =1./(q1+g2)
          eps(3)=-q1*(eps(1)+eps(2))*q3
          davg  =(-eps(1)-eps(2)-eps(3))/3.
          p     =-davg*abs(cm(1))/(1.-2.*cm(2))
          sig(1)=sig(1)+p+g2*(eps(1)+davg)
          sig(2)=sig(2)+p+g2*(eps(2)+davg)
          sig(3)=0.0
          sig(4)=sig(4)+g *eps(4)
          sig(5)=sig(5)+gc*eps(5)
          sig(6)=sig(6)+gc*eps(6)
          if (cm(1).lt.0.) then
            if (sig(1).gt.cm(5)) failel=.true.
          endif
          endif
        end if
      elseif (etype.eq.'beam ' ) then
          q1    =cm(1)*cm(2)/((1.0+cm(2))*(1.0-2.0*cm(2)))
          q3    =q1+2.0*g
          gc    =capa*g
          deti  =1./(q3*q3-q1*q1)
          c22i  = q3*deti
          c23i  =-q1*deti
          fac   =(c22i+c23i)*q1
          eps(2)=-eps(1)*fac-sig(2)*c22i-sig(3)*c23i
          eps(3)=-eps(1)*fac-sig(2)*c23i-sig(3)*c22i
          davg  =(-eps(1)-eps(2)-eps(3))/3.
          p     =-davg*cm(1)/(1.-2.*cm(2))
          sig(1)=sig(1)+p+g2*(eps(1)+davg)
          sig(2)=0.0
          sig(3)=0.0
          sig(4)=sig(4)+gc*eps(4)
          sig(5)=0.0
          sig(6)=sig(6)+gc*eps(6)

c
      elseif (etype.eq.'tbeam') then
        q1    =cm(1)*cm(2)/((1.0+cm(2))*(1.0-2.0*cm(2)))
        q3    =q1+2.0*g
        deti  =1./(q3*q3-q1*q1)
        c22i  = q3*deti
        c23i  =-q1*deti
        fac   =(c22i+c23i)*q1
        eps(2)=-eps(1)*fac
        eps(3)=-eps(1)*fac
        davg  =(-eps(1)-eps(2)-eps(3))/3.
        p     =-davg*cm(1)/(1.-2.*cm(2))
        sig(1)=sig(1)+p+g2*(eps(1)+davg)
        sig(2)=0.0
        sig(3)=0.0
c
      else
c       write(iotty,10) etype
c       write(iohsp,10) etype
c       write(iomsg,10) etype
c       call adios(TC_ERROR)
        cerdat(1)=etype
        call lsmsg(3,MSG_SOL+1150,ioall,ierdat,rerdat,cerdat,0)
      endif
c
c10   format(/
c    1 ' *** Error element type ',a,' can not be',
c    2 '           run with the current material model.')
      return
      end
```

Based on the subroutine `umat41` shown above, the following material input:

```
*MAT_USER_DEFINED_MATERIAL_MODELS
$#     mid        ro        mt       lmc       nhv    iortho     ibulk        ig
         1 7.8300E-6        41         4         0         0         3         4
$#   ivect     ifail    itherm    ihyper      ieos
         0         0         0         0         0
$        E        PR      BULK         G
$#      p1        p2        p3        p4        p5        p6        p7        p8
  2.000000  0.300000  1.667000  0.769200     0.000     0.000     0.000     0.000
```

is functionally equivalent to:

```
*MAT_ELASTIC
$#     mid        ro         e        pr        da        db  not used
         1 7.8300E-6  2.000000  0.300000     0.000     0.000         0
```

ADDITIONAL FEATURES

Load curves and tables

If the material of interest requires load curves, predefined routines easily obtain curve and table data. For instance, your material implementation may need a curve defining yield stress as a function of effective plastic strain. For curves, call

```
subroutine curveval(eid,xval,yval,slope)
```

or

```
subroutine curveval_v(eid,xval,yval,slope,lft,llt)
```

where the former routine works for scalar routines and the latter for vectorized. These subroutines have the following arguments:

`eid` - external load curve ID, meaning the load curve ID taken from the keyword deck

GT.0: Use a rediscretized representation of the curve

	LT.0: Use exact representation of the curve (with ID –`eid`)
`xval` -	abscissa value
`yval` -	ordinate value (output from routine)
`slope` -	slope of curve (output from routine)
`lft` -	first index of vector
`llt` -	final index of vector

where `xval`, `yval`, and `slope` are scalars in the scalar routine and vectors of length `nlq` in the vectorized routine. Note that `eid` should be passed as a float. A positive value for `eid` causes the routine to use a rediscretized representation of the curve. In contrast, a negative value causes the extraction to be made on the curve as it is defined in the keyword input deck.

For tables, we have two subroutines available for extracting values. The scalar version is

```
      subroutine tableval(eid,dxval,yval,dslope,xval,slope)
```

and the vector version is

```
      subroutine
     1 tableval_v(eid,dxval,yval,dslope,lft,llt,xval,slope)
```

where

`eid` -	external table ID (data type real), meaning table ID taken from the keyword deck GT.0: Use a rediscretized representation of the curve LT.0: Use exact representation of curve (with ID –`eid`)
`dxval` -	abscissa value (x_2-axis)
`yval` -	ordinate value (y-axis, output from routine)
`dslope` -	slope of curve (dy/dx_2, output from routine)
`xval` -	abscissa value (x_1-axis)
`slope` -	slope of curve (dy/dx_1, output from routine)
`lft` -	vector index
`llt` -	vector index

In the scalar routine, `dxval`, `yval`, `dslope`, `xval`, and `slope` are all scalars, whereas, in the vector routine, they are vectors of length `nlq`. A positive and negative value of `eid` works in the same way for tables as it does for curves.

Local coordinate system

Invoke the local coordinate system option if the material model has directional properties, such as composite and anisotropic plasticity models. To do this, set IORTHO to 1 on the material card. A local coordinate requires two additional cards for the material with values for forming and updating the coordinate system. When invoked, the constitutive routine, `umat`*`XX`* or `umat`*`XX`*`v`, receives all the data in the local system. You do not transform the data to the global system since another routine performs the transformation unless your model requires the deformation gradient. See Deformation gradient for more details.

Temperature

Setting ITHERMAL to 1 on the material card for a material with thermal properties makes temperature available to your subroutine. The `temper` variable for a scalar and `temps` array for the vectorized implementation contain this data. For a coupled thermal-structural analysis, LS-DYNA solves the thermal problem first, making temperatures at the current time available in the user-defined subroutine. Note that LS-DYNA calculates the dissipated heat in the presence of plastic deformation. Thus, you do not include it in your subroutine. If the stress update requires the time derivative of the temperature, request a history variable that contains the temperature in the previous time step. Adding a backward finite difference estimate to your subroutine gives the time derivative.

Failure

Your material model may include failure, resulting in deleting elements that fulfill a failure criterion. To accomplish this, set IFAIL to 1 or a negative number on the material card. For a scalar implementation, the variable `failel` is set to `.true.` when a failure criterion is met. For a vectorized implementation, the corresponding entry in the `failels` array is set to `.true`.

Deformation gradient

For some material models, the stresses depend on the deformation gradient, **F**, instead of incremental strains. For instance, this dependency applies to hyperelastic and hyperplastic materials. To make the deformation gradient available for bricks and shells in the user-defined material subroutines, set IHYPER to 1 on the material card. LS-DYNA puts the deformation gradient components F_{11}, F_{21}, F_{31}, F_{12}, F_{22}, F_{32}, F_{13}, F_{23}, and F_{33} in the history variables array in positions NHV+1 to NHV+9, that is, the positions right after the requested number of history variables.

For shell elements, the components of the deformation gradient are with respect to the co-rotational system for the element. Note that the user-defined material routine needs to properly update the third row of the deformation gradient, components F_{31}, F_{32}, and F_{33}. These components depend on the thickness strain increment, which must be determined so that the normal stress in the shell vanishes. For a given thickness strain increment `d3`, the following subroutines can determine these three components, `f31`, `f32` and `f33`:

```
        subroutine compute_f3s(f31,f32,f33,d3)
```

for a scalar implementation and

```
        subroutine compute_f3(f31,f32,f33,d3,lft,llt)
```

for a vector implementation. The first four arguments are scalars for the scalar routine and arrays of length `nlq` for the vector routine.

For hyperelastic materials, the user-defined subroutines can call push-forward operations. These routines are

```
        subroutine push_forward_2(sig1,sig2,sig3,sig4,sig5,sig6,
     .          f11,f21,f31,f12,f22,f32,f13,f23,f33,lft,llt)
```

and

```
        subroutine push_forward_2s(sig1,sig2,sig3,sig4,sig5,sig6,
     .          f11,f21,f31,f12,f22,f32,f13,f23,f33)
```

which perform push-forward operations on the stress tensor for the vectorized and scalar routines, respectively. In the latter subroutine, all arguments are scalars, whereas the corresponding entries in the vectorized routine are vectors of length `nlq`. The `sig1` to `sig6` are components of the stress tensor, and `f11` to `f33` are components of the deformation gradient.

Suppose you invoke the local coordinate system option (IORTHO = 1). In that case, the program transforms the deformation gradient into the local system with the following expression before passing it to the user-defined subroutine:

$$\bar{F}_{ij} = Q^s_{ki} F_{kj} \ .$$

Here Q^s_{ij} refers to a transformation between the current global and material frames. For IORTHO equal to 1, setting IHYPER equal to –1 results in the following transformation of the deformation gradient:

$$\bar{F}_{ij} = F_{ik} Q^r_{kj} \ .$$

Q^r_{ij} is the transformation between the reference global and material frames. For this latter option the spatial frame remains the global one, so the user-defined routine must express the stresses in this frame of reference upon exiting the routine. The suitable choice of IHYPER depends on the formulation of the material model.

For shells only, a particular option invoked by setting IHYPER to 3 causes LS-DYNA to compute the deformation gradient from the nodal coordinates in the global coordinate system. With this option, you must calculate the stress in the local system of interest. LS-DYNA passes a transformation matrix between the global and this local system to the user material routines (`qmat`). The columns in this matrix correspond to local basis vectors expressed in global coordinates. Your subroutine must compute the stress in this system. Note that the resulting deformation gradient may not be consistent with the theory for the element for your material because of how LS-DYNA calculates it. To account for thickness changes due to membrane straining, we provide the following subroutines:

```
        subroutine usrshl_updatfs(f,t,s,e)
```

and

```
        subroutine usrshl_updatfv(f,t,s,e,lft,llt)
```

for the scalar and vector versions. The subroutine recomputes the deformation gradient based on the thickness strain increment, `e`, and the nodal thicknesses, `t`. The history variables array stores the current nodal thicknesses immediately after the deformation gradient. Your subroutine must call this routine with these four values as `t`. This subroutine produces a deformation, `f`, and the new thicknesses, `s`. Your subroutine finds the strain increment, `e`, giving zero thickness stress through this subroutine. After obtaining zero thickness stress, your subroutine needs to store the new thicknesses, `s`, in the history variables array. To achieve this, copy the new thicknesses, `s`, to the location for the nodal thicknesses. We provide sample code in the object library.

Sample user subroutine 45

Here we provide an example of a user-defined material that requires the deformation gradient. Our user-defined subroutine models a Neo-Hookean material. With λ and μ being the Lame parameters in the linearized theory, the following gives the strain energy density for this material:

$$\psi = \frac{1}{2}\lambda(\ln(\det \mathbf{F}))^2 - \mu \ln(\det \mathbf{F}) + \frac{1}{2}\mu\left(\mathrm{tr}\left(\mathbf{F}^T\mathbf{F}\right) - 3\right) \ .$$

Thus, we can express the Cauchy stress as

$$\sigma = \frac{1}{\det \mathbf{F}}\left(\lambda \ln(\det \mathbf{F})\,\mathbf{I} + \mu\left(\mathbf{F}\mathbf{F}^T - \mathbf{I}\right)\right) \ .$$

```
      subroutine umat45 (cm,eps,sig,epsp,hsv,dt1,capa,
     . etype,time,temp,failel,crv,nnpcrv,cma,qmat,elsiz,idele,reject)
c
c*****************************************************************
c|  Livermore Software Technology Corporation  (LSTC)           |
c|  ------------------------------------------------------------ |
c|  Copyright 1987-2008 Livermore Software Tech.  Corp           |
c|  All rights reserved                                         |
c*****************************************************************
c
c     Neo-Hookean material (sample user subroutine)
c
c     Variables
c
c     cm(1)=first material constant, here young's modulus
c     cm(2)=second material constant, here poisson's ratio
c        .
c        .
c        .
c     cm(n)=nth material constant
c
c     eps(1)=local x  strain increment
c     eps(2)=local y  strain increment
c     eps(3)=local z  strain increment
c     eps(4)=local xy strain increment
c     eps(5)=local yz strain increment
c     eps(6)=local zx strain increment
c
c     sig(1)=local x  stress
c     sig(2)=local y  stress
c     sig(3)=local z  stress
c     sig(4)=local xy stress
c     sig(5)=local yz stress
```

```
c     sig(6)=local zx stress
c
c     hsv(1)=1st history variable
c     hsv(2)=2nd history variable
c        .
c        .
c        .
c        .
c     hsv(n)=nth history variable
c
c     dt1=current time step size
c     capa=reduction factor for transverse shear
c     etype:
c       eq."solid" for solid elements
c       eq."sld2d" for shell forms 13, 14, and 15 (2D solids)
c       eq."shl_t" for shell forms 25, 26, and 27 (shells with thickness
c        stretch)
c       eq."shell" for all other shell elements plus thick shell forms 1
c        and 2
c       eq."tshel" for thick shell forms 3 and 5
c       eq."hbeam" for beam element forms 1 and 11
c       eq."tbeam" for beam element form 3 (truss)
c       eq."dbeam" for beam element form 6 (discrete)
c       eq."beam " for all other beam elements
c
c     time=current problem time.
c
c     temp=current temperature
c
c     failel=flag for failure, set to .true.  to fail an integration point,
c            if .true.  on input the integration point has failed earlier
c
c     crv=array representation of curves in keyword deck
c
c     nnpcrv=# of discretization points per crv()
c
c     cma=additional memory for material data defined by LMCA at
c       6th field of 2nd crad of *DATA_USER_DEFINED
c
c     elsiz=characteristic element size
c
c     idele=element id
c
c     reject (implicit only) = set to .true.  if this implicit iterate is
c                              to be rejected for some reason
c
c     All transformations into the element local system are
c     performed prior to entering this subroutine.  Transformations
c     back to the global system are performed after exiting this
c     routine.
c
c     All history variables are initialized to zero in the input
c     phase.   Initialization of history variables to nonzero values
c     may be done during the first call to this subroutine for each
c     element.
c
c     Energy calculations for the dyna3d energy balance are done
c     outside this subroutine.
c
      include 'nlqparm'
      include 'iounits.inc'
      include 'bk06.inc'
      character*5 etype
      dimension cm(*),eps(*),sig(*),hsv(*),crv(lq1,2,*),cma(*)
      logical failel
c
```

```
      if (ncycle.eq.1) then
        call usermsg('mat45')
      endif
c
c     compute lame parameters
c
      xlambda=cm(1)*cm(2)/((1.+cm(2))*(1.-2.*cm(2)))
      xmu=.5*cm(1)/(1.+cm(2))
c
      if (etype.eq.'solid'.or.etype.eq.'shl_t'.or.
     1     etype.eq.'sld2d'.or.etype.eq.'tshel') then
c
c       deformation gradient stored in hsv(1),...,hsv(9)
c
c       compute jacobian
c
        detf=hsv(1)*(hsv(5)*hsv(9)-hsv(6)*hsv(8))
     1      -hsv(2)*(hsv(4)*hsv(9)-hsv(6)*hsv(7))
     2      +hsv(3)*(hsv(4)*hsv(8)-hsv(5)*hsv(7))
c
c       compute left cauchy-green tensor
c
        b1=hsv(1)*hsv(1)+hsv(4)*hsv(4)+hsv(7)*hsv(7)
        b2=hsv(2)*hsv(2)+hsv(5)*hsv(5)+hsv(8)*hsv(8)
        b3=hsv(3)*hsv(3)+hsv(6)*hsv(6)+hsv(9)*hsv(9)
        b4=hsv(1)*hsv(2)+hsv(4)*hsv(5)+hsv(7)*hsv(8)
        b5=hsv(2)*hsv(3)+hsv(5)*hsv(6)+hsv(8)*hsv(9)
        b6=hsv(1)*hsv(3)+hsv(4)*hsv(6)+hsv(7)*hsv(9)
c
c       compute cauchy stress
c
        detfinv=1./detf
        dmu=xmu-xlambda*log(detf)
        sig(1)=detfinv*(xmu*b1-dmu)
        sig(2)=detfinv*(xmu*b2-dmu)
        sig(3)=detfinv*(xmu*b3-dmu)
        sig(4)=detfinv*xmu*b4
        sig(5)=detfinv*xmu*b5
        sig(6)=detfinv*xmu*b6
c
      else if (etype.eq.'shell') then
c
c       deformation gradient stored in hsv(1),...,hsv(9)
c
c       compute part of left cauchy-green tensor
c       independent of thickness strain increment
c
        b1=hsv(1)*hsv(1)+hsv(4)*hsv(4)+hsv(7)*hsv(7)
        b2=hsv(2)*hsv(2)+hsv(5)*hsv(5)+hsv(8)*hsv(8)
        b4=hsv(1)*hsv(2)+hsv(4)*hsv(5)+hsv(7)*hsv(8)
c
c       secant iterations for zero normal stress
c
        do iter=1,5
c
c         first thickness strain increment initial guess
c         assuming Poisson's ratio different from zero
c
          if (iter.eq.1) then
            eps(3)=-xlambda*(eps(1)+eps(2))/(xlambda+2.*xmu)
c
c         second thickness strain increment initial guess
c
          else if (iter.eq.2) then
            sigold=sig(3)
            epsold=eps(3)
            eps(3)=0.
c
c         secant update of thickness strain increment
c
          else if (abs(sig(3)-sigold).gt.0.0) then
            deps=-(eps(3)-epsold)/(sig(3)-sigold)*sig(3)
            sigold=sig(3)
            epsold=eps(3)
            eps(3)=eps(3)+deps
          endif
c
c         compute last row of deformation gradient
c
          call compute_f3s(hsv(3),hsv(6),hsv(9),eps(3))
c
c         compute jacobian
c
          detf=hsv(1)*(hsv(5)*hsv(9)-hsv(6)*hsv(8))
     1        -hsv(2)*(hsv(4)*hsv(9)-hsv(6)*hsv(7))
     2        +hsv(3)*(hsv(4)*hsv(8)-hsv(5)*hsv(7))
c
c         compute normal component of left cauchy-green tensor
c
          b3=hsv(3)*hsv(3)+hsv(6)*hsv(6)+hsv(9)*hsv(9)
c
c         compute normal stress
c
          detfinv=1./detf
          dmu=xmu-xlambda*log(detf)
          sig(1)=detfinv*(xmu*b1-dmu)
          sig(2)=detfinv*(xmu*b2-dmu)
          sig(3)=detfinv*(xmu*b3-dmu)
          sig(4)=detfinv*xmu*b4
c
c         exit if normal stress is sufficiently small
c
          if (abs(sig(3)).le.1.e-5*
     1      (abs(sig(1))+abs(sig(2))+abs(sig(4)))) goto 10
         enddo
c
c       compute remaining components of left cauchy-green tensor
c
 10     b5=hsv(2)*hsv(3)+hsv(5)*hsv(6)+hsv(8)*hsv(9)
        b6=hsv(1)*hsv(3)+hsv(4)*hsv(6)+hsv(7)*hsv(9)
c
c       compute remaining stress components
c
        sig(5)=detfinv*xmu*b5
        sig(6)=detfinv*xmu*b6
c
c     material model only available for solids and shells
c
      else
        cerdat(1)=etype
        call lsmsg(3,MSG_SOL+1151,ioall,ierdat,rerdat,cerdat,0)
      endif
      return
      end
```

Implicit analysis

We also support user-defined material models with implicit analysis for solid, shell, thick shell, and Hughes-Liu beam elements. When you request implicit analysis in the input deck, LS-DYNA calls the subroutine `urtanh` for solids (and thick shell types 3, 5, and 7), `urtans` for shells (and thick shell types 1, 2, and 6), and `urtanb` for beams with appropriate input data for calculating the material tangent modulus. This subroutine then calls a user-defined subroutine for calculating the tangent modulus for your material: `utanXX` for the scalar implementation or `utanXXv` for the vectorized implementation. Again, *`XX`* is the number that matches MT on the material card. `utanXX` includes

`es(6,6)` – material tangent modulus

as an argument, while `utanXX` has the corresponding vector block

`dsave(nlq,6,6)` – material tangent modulus

as an argument. Your user-defined subroutine builds the tangent modulus matrix needed for assembling the tangent stiffness matrix. This matrix equals the zero matrix when entering the user-defined subroutine. The matrix must be symmetric. If you invoke the local coordinate system option for solids, it should be in this local system. For shell elements, it should be in the co-rotational system defined for the current shell element. All transformations back to the global system are made after exiting the user-defined subroutine.

The user-defined material routine (`umatXX` or `umatXXv`) includes the argument `reject` to improve convergence characteristics. To use this parameter, have your user-defined subroutine set it to `.true.` when a criterion declares the iteration unacceptable. For example, the plastic strain could increase too much in one step. If your material subroutine rejects the iteration, LS-DYNA prints a warning message that says, 'Material model rejected current iterate.' LS-DYNA then retries the step with a smaller time step. If chosen carefully (by way of experimenting), this feature may result in a good trade-off between the number of implicit iterations per step and the step size for overall speed.

If the material is hyperelastic, you can apply push-forward operations on the tangent modulus tensor with

```
      subroutine push_forward_4(dsave,
     .      f11,f21,f31,f12,f22,f32,f13,f23,f33,lft,llt)
```

or

```
      subroutine push_forward_4s(es,
     .      f11,f21,f31,f12,f22,f32,f13,f23,f33)
```

for the vector and scalar implementations, respectively. In the latter subroutine, all arguments are scalars, whereas the corresponding entries in the vectorized routine are vectors of length `nlq`. `f11` to `f33` are components of the deformation gradient.

Sample user-subroutine 42, tangent modulus

The following sample user subroutine illustrates how to implement the tangent stiffness modulus for the Neo-Hookean material covered in the Sample user subroutine 45 subsection of the Deformation gradient section. The material tangent modulus is for this material given by

$$\mathbf{C} = \frac{1}{\det \mathbf{F}}\left(\lambda \mathbf{I} \otimes \mathbf{I} + 2\left(\mu - \lambda \ln(\det \mathbf{F})\, \mathbf{I}\right)\right) \ .$$

```
      subroutine utan42(cm,eps,sig,epsp,hsv,dt1,capa,
     .     etype,tt,temper,es,crv)
c*******************************************************************
c|  livermore software technology corporation  (lstc)             |
c|  ------------------------------------------------------------  |
c|  copyright 1987-1999                                           |
c|  all rights reserved                                           |
c*******************************************************************
c
c     Neo-Hookean material tangent modulus (sample user subroutine)
c
c     Variables
c
c     cm(1)=first material constant, here young's modulus
c     cm(2)=second material constant, here poisson's ratio
c        .
c        .
c        .
c      cm(n)=nth material constant
c
c     eps(1)=local x  strain increment
c     eps(2)=local y  strain increment
c     eps(3)=local z  strain increment
c     eps(4)=local xy strain increment
c     eps(5)=local yz strain increment
c     eps(6)=local zx strain increment
c
c     sig(1)=local x  stress
c     sig(2)=local y  stress
c     sig(3)=local z  stress
c     sig(4)=local xy stress
c     sig(5)=local yz stress
c     sig(6)=local zx stress
c
c      epsp=effective plastic strain
c
c     hsv(1)=1st history variable
c     hsv(2)=2nd history variable
c        .
c        .
c        .
c        .
c     hsv(n)=nth history variable
c
c     dt1=current time step size
c     capa=reduction factor for transverse shear
c     etype:
c        eq."brick" for solid elements
c        eq."shell" for all shell elements
c        eq."beam"  for all beam elements
c        eq."dbeam"  for all discrete beam elements
c
c     tt=current problem time.
c
c     temper=current temperature
c
c     es=material tangent modulus
c
c      crv=array representation of curves in keyword deck
c
```

```
c      The material tangent modulus is set to 0 before entering
c      this routine.  It should be expressed in the local system
c      upon exiting this routine.  All transformations back to the
c      global system are made outside this routine.
      include 'nlqparm'
      character*(*) etype
      dimension cm(*),eps(*),sig(*),hsv(*),crv(lq1,2,*)
      dimension es(6,*)
c
c     no history variables, NHV=0
c     deformation gradient stored in hsv(1),...,hsv(9)
c
c     compute jacobian
c
      detf=hsv(1)*(hsv(5)*hsv(9)-hsv(6)*hsv(8))
     1     -hsv(2)*(hsv(4)*hsv(9)-hsv(6)*hsv(7))
     2     +hsv(3)*(hsv(4)*hsv(8)-hsv(5)*hsv(7))
c
c     compute lame parameters
c
      xlambda=cm(1)*cm(2)/((1.+cm(2))*(1.-2.*cm(2)))
      xmu=.5*cm(1)/(1.+cm(2))
c
c     compute tangent stiffness
c     same for both shells and bricks
c
      detfinv=1./detf
      dmu=xmu-xlambda*log(detf)
      es(1,1)=detfinv*(xlambda+2.*dmu)
      es(2,2)=detfinv*(xlambda+2.*dmu)
      es(3,3)=detfinv*(xlambda+2.*dmu)
      es(4,4)=detfinv*dmu
      es(5,5)=detfinv*dmu
      es(6,6)=detfinv*dmu
      es(2,1)=detfinv*xlambda
      es(3,2)=detfinv*xlambda
      es(3,1)=detfinv*xlambda
      es(1,2)=es(2,1)
      es(2,3)=es(3,2)
      es(1,3)=es(3,1)
c
      return
      end
```

User-defined materials with equations-of-state

The following example user-defined subroutine uses an equation-of-state (EOS). Unlike standard models, it updates only the deviatoric stress and assigns a value to PC, the pressure cut-off. The pressure cut-off limits the amount of hydrostatic pressure that can be carried in tension (that is, when the pressure is negative). The default value is zero. A large negative number will allow the material to carry an unlimited pressure load in tension. Typically, the pressure cut-off is a function of the current state of the material and varies with time. Thus, the material model subroutine needs to calculate it. We made the pressure cut-off a constant value for simplicity in this example. The named common block `eosdloc` stores the pressure cut-off array. Depending on the computing environment, compiler directives may be required (such as the task common directive in the example) for correct SMP execution.

In addition, the number of history variables, NHV, must be increased by 4 in the input file to allocate the extra storage required for the EOS. The first 4 variables in `hsvs` contain the storage. The user-defined material model must not alter it.

```
      subroutine umat44v(cm,d1,d2,d3,d4,d5,d6,sig1,sig2,
     . sig3,sig4,sig5,sig6,eps,hsvs,lft,llt,dt1siz,capa,
     . etype,tt,temps,failels,nlqa,crv)
      parameter (third=1.0/3.0)
      include 'nlqparm'
c
c***  isotropic plasticity with linear hardening
c
c***  updates only the deviatoric stress so that it can be used with
c     an equation of state
c
      character*5 etype
      logical failels
c
C_TASKCOMMON (eosdloc)
      common/eosdloc/pc(nlq)
c
      dimension cm(*),d1(*),d2(*),d3(*),d4(*),d5(*),d6(*),
     & sig1(*),sig2(*),sig3(*),sig4(*),sig5(*),sig6(*),
     & eps(*),hsvs(nlqa,*),dt1siz(*),temps(*),crv(lq1,2,*),
     & failels(*)
c
c***  shear modulus, initial yield stress, hardening, and pressure cut-off
      g   =cm(1)
      sy0 =cm(2)
      h   =cm(3)
      pcut=cm(4)
c
      ofac=1.0/(3.0*g+h)
      twog=2.0*g
c
      do i=lft,llt
c
c***    trial elastic deviatoric stress
        davg=third*(d1(i)+d2(i)+d3(i))
        savg=third*(sig1(i)+sig2(i)+sig3(i))
        sig1(i)=sig1(i)-savg+twog*(d1(i)-davg)
        sig2(i)=sig2(i)-savg+twog*(d2(i)-davg)
        sig3(i)=sig3(i)-savg+twog*(d3(i)-davg)
        sig4(i)=sig4(i)+g*d4(i)
        sig5(i)=sig5(i)+g*d5(i)
        sig6(i)=sig6(i)+g*d6(i)
c
c***    radial return
        aj2=sqrt(1.5*(sig1(i)**2+sig2(i)**2+sig3(i)**2)+
     &           3.0*(sig4(i)**2+sig5(i)**2+sig6(i)**2))
        sy=sy0+h*eps(i)
        eps(i)=eps(i)+ofac*max(0.0,aj2-sy)
        synew=sy0+h*eps(i)
        scale=synew/max(synew,aj2)
c
c***    scaling for radial return.  note that the stress is now deviatoric.
        sig1(i)=scale*sig1(i)
        sig2(i)=scale*sig2(i)
        sig3(i)=scale*sig3(i)
        sig4(i)=scale*sig4(i)
        sig5(i)=scale*sig5(i)
        sig6(i)=scale*sig6(i)
c
c***    set pressure cut-off
        pc(i)=pcut
```

```
c
      enddo
c
      return
      end
```

Post-processing a user-defined material

Post-processing a user-defined material is very similar to post-processing a regular LS-DYNA material. However, we would like to stress some aspects of post-processing, all dealing with how to post-process history variables.

First, LS-DYNA always writes the effective plastic strain to the d3plot database and thus need not be requested by you. LS-PREPOST treats it the same as for the result from any other LS-DYNA material.

NEIPH and NEIPS on *DATABASE_EXTENT_BINARY specify the number of additional history variables written to the d3plot database for solids and shells, respectively. For instance, if NEIPH (NEIPS) equals 2, LS-DYNA writes the first two history variables in the history variables array as history var#1 and history var#2 in the d3plot database. By setting NEIPH (NEIPS) equal to `NHV`, LS-DYNA outputs all history variables to the d3plot database. Furthermore, if the material model uses the deformation gradient (IHYPER = 1), you must request an additional nine variables to make it available for post-processing, meaning set NEIPH (NEIPS) equal to `NHV+9`. The deformation gradient becomes available in the d3plot database as history variables `NHV+1` to `NHV+9`. Note that LS-DYNA outputs the deformation gradient in the co-rotational system for shells. If using the local coordinate system option (IORTHO = 1), LS-DYNA expresses the deformation gradient in this local system. Set NEIPH (NEIPS) equal to `NHV+9+9(=NHV+18)` to output the deformation gradient in the global system for bricks and the co-rotational system for shells. LS-DYNA outputs these as history variables `NHV+10` to `NHV+18`.