10 User Material Subroutines

Chapter Outline

10.1 Introduction

Commercial finite element (FE) codes contain a limited selection of material models that are suitable for certain materials and applications, but as has been highlighted in previous chapters it is many times desirable (or even necessary) to achieve more accurate predictions than what is possible with the built-in options. Under these circumstances the best approach is to use a suitable *user material subroutine* to represent the material response. A user material subroutine is a source code subroutine that is used by the FE software to calculate the stress for a given increment in time and deformation state. User material subroutines are typically written in Fortran, but can be written in any computer language. The purpose of the user subroutine is essentially to update the stress and the state variables during each increment, as in the following:

1. Known at time t: $\boldsymbol{\sigma}$, $\mathbf{F}$, ξ (state variables).
2. Known at time $t + \mathrm{d}t$: $\mathbf{F}$.
3. Use the user material subroutine to calculate the following at $t + \mathrm{d}t$: $\boldsymbol{\sigma}$, ξ.

Mechanics of Solid Polymers.
DOI: http://dx.doi.org/10.1016/B978-0-323-31150-2.00010-8

There are commercial options that work as plug-ins to the FE code. The PolyUMod library [1] that was discussed in Chapter 8 is one example. It is of course also possible to write your own user material model, and this chapter will briefly introduce how this can be done. The examples presented in this chapter are based on the Neo-Hookean hyperelastic material model with a stress given by:

$$\boldsymbol{\sigma} = \frac{\mu}{J}\,\mathrm{dev}[\mathbf{b}^*] + \kappa(J-1)\mathbf{I}, \tag{10.1}$$

where μ is the shear modulus, κ is the bulk modulus, $J = \det(\mathbf{F})$, $\mathbf{b}^* = J^{-2/3}\mathbf{F}\mathbf{F}^\top$ is the distortional left Cauchy-Green tensor, and $\boldsymbol{\sigma}$ is the Cauchy stress.

The following subsections provide the code for an Abaqus/Explicit VUMAT and an Abaqus/Standard UMAT. Subroutines for other FE solver can also be written but are left as an exercise. The code examples listed below use `real` to define a floating point variable. This code can be compiled into either single or double precision simply by specifying the command line argument `-r8` or not to the Fortran compiler.

10.2 Abaqus/Explicit VUMAT for the Neo-Hookean Model

Abaqus/Explicit requires that the stress is returned in a co-rotational coordinate frame:

$$\hat{\boldsymbol{\sigma}} = \mathbf{R}^T\boldsymbol{\sigma}\mathbf{R}. \tag{10.2}$$

Since $\mathbf{R}^T\mathbf{b}^*\mathbf{R} = \mathbf{R}^T(\mathbf{F}^*\mathbf{F}^{*T})\mathbf{R} = \mathbf{R}^T(\mathbf{R}\mathbf{U}^*\mathbf{U}^*\mathbf{R}^T)\mathbf{R} = \mathbf{U}^{*2}$, the required stress becomes:

$$\hat{\boldsymbol{\sigma}} = \frac{\mu}{J}\,\mathrm{dev}[\mathbf{U}^{*2}] + \kappa(J-1)\mathbf{I}. \tag{10.3}$$

These equations can be implemented in the following code:

```
c     FILE:
c        vumat_nh.f
c     AUTHOR:
c        Jorgen Bergstrom  (jorgen@polymerFEM.com)
```

```
c     CONTENTS:
c        Abaqus VUMAT subroutine for the
c        Neo-Hookean (NH) model. The subroutine is an example
c        of how to write a VUMAT, and not is not designed to
c        be a commercial quality implementation.
c     USAGE:
c        2 material constants:
c           1: mu    (shear modulus)
c           2: kappa (bulk modulus)
c
c     |<- column 1 begins here
c     |
c     *User material, constants=2
c     **     mu,    kappa
c           5.0,    100.0
c     *Density
c     1000e-12
c
      subroutine vumat(nblock, ndi, nshr, nstatev, nfieldv, nprops,
     .     lanneal, stepTime, totTime, dt, cmname, coordMp,
     .     charLen, props, density, Dstrain, rSpinInc, temp0,
     .     U0, F0, field0, stressVec0, state0,
     .     intEne0, inelaEn0, temp1, U1,
     .     F1, field1, stressVec1, state1, intEne1, inelaEn1)
      implicit none
      integer nblock, ndi, nshr, nstatev, nfieldv, nprops, lanneal
      real stepTime, totTime, dt
      character*80 cmname
      real coordMp(nblock,*)
      real charLen, props(nprops), density(nblock),
     .     Dstrain(nblock,ndi+nshr), rSpinInc(nblock,nshr),
     .     temp0(nblock), U0(nblock,ndi+nshr),
     .     F0(nblock,ndi+nshr+nshr), field0(nblock,nfieldv),
     .     stressVec0(nblock,ndi+nshr), state0(nblock,nstatev),
     .     intEne0(nblock), inelaEn0(nblock), temp1(nblock),
     .     U1(nblock,ndi+nshr), F1(nblock,ndi+nshr+nshr),
     .     field1(nblock,nfieldv), stressVec1(nblock,ndi+nshr),
     .     state1(nblock,nstatev), intEne1(nblock), inelaEn1(nblock)

c     local variables
      real F(3,3), B(3,3), J, t1, t2, t3, mu, kappa
      integer i

      mu = props(1)
      kappa = props(2)

c     loop through all blocks
      do i = 1, nblock
c        setup F (upper diagonal part)
         F(1,1) = U1(i,1)
         F(2,2) = U1(i,2)
         F(3,3) = U1(i,3)
         F(1,2) = U1(i,4)
         if (nshr .eq. 1) then
            F(2,3) = 0.0
            F(1,3) = 0.0
         else
            F(2,3) = U1(i,5)
            F(1,3) = U1(i,6)
         end if

c        calculate J
         t1 = F(1,1) * (F(2,2)*F(3,3) - F(2,3)**2)
         t2 = F(1,2) * (F(2,3)*F(1,3) - F(1,2)*F(3,3))
         t3 = F(1,3) * (F(1,2)*F(2,3) - F(2,2)*F(1,3))
         J = t1 + t2 + t3
         t1 = J**(-2.0/3.0)
```

```
c        Bstar = J^(-2/3) F Ft   (upper diagonal part)
         B(1,1) = t1*(F(1,1)**2 + F(1,2)**2 + F(1,3)**2)
         B(2,2) = t1*(F(1,2)**2 + F(2,2)**2 + F(2,3)**2)
         B(3,3) = t1*(F(1,3)**2 + F(2,3)**2 + F(3,3)**2)
         B(1,2) = t1*(F(1,1)*F(1,2)+F(1,2)*F(2,2)+F(1,3)*F(2,3))
         B(2,3) = t1*(F(1,2)*F(1,3)+F(2,2)*F(2,3)+F(2,3)*F(3,3))
         B(1,3) = t1*(F(1,1)*F(1,3)+F(1,2)*F(2,3)+F(1,3)*F(3,3))

c        Stress = mu/J * Dev(Bstar) + kappa*log(J)/J * Eye
         t1 = (B(1,1) + B(2,2) + B(3,3)) / 3.0
         t2 = mu/J
         t3 = kappa * log(J) / J
         StressVec1(i,1) = t2 * (B(1,1) - t1) + t3
         StressVec1(i,2) = t2 * (B(2,2) - t1) + t3
         StressVec1(i,3) = t2 * (B(3,3) - t1) + t3
         StressVec1(i,4) = t2 * B(1,2)
         if (nshr .eq. 3) then
            StressVec1(i,5) = t2 * B(2,3)
            StressVec1(i,6) = t2 * B(1,3)
         end if

      end do
      return
      end
```

10.3 Abaqus/Implicit UMAT for the Neo-Hookean Model

The Abaqus UMAT subroutine is very similar to the VUMAT presented above but it is based on the total deformation gradient and it also needs the Jacobian matrix, which is the partial derivative of the stress increment with respect to the applied strain increment.

The Jacobian for this model is given by: $\mathbf{c} = \partial \Delta \boldsymbol{\sigma} / \partial \Delta \boldsymbol{\varepsilon}$. Here, due to space constraints, each column of $[\mathbf{c}]_{ij}$ is presented individually:

$$\begin{bmatrix} \mathbf{c}_{11} \\ \mathbf{c}_{21} \\ \mathbf{c}_{31} \\ \mathbf{c}_{41} \\ \mathbf{c}_{51} \\ \mathbf{c}_{61} \end{bmatrix} = \frac{\mu}{9J} \begin{bmatrix} 8b^*_{11} + 2b^*_{22} + 2b^*_{33} \\ -4b^*_{11} - 4b^*_{22} + 2b^*_{33} \\ -4b^*_{11} + 2b^*_{22} - 4b^*_{33} \\ 3b^*_{12} \\ 3b^*_{13} \\ -6b^*_{23} \end{bmatrix} + \kappa \begin{bmatrix} J \\ J \\ J \\ 0 \\ 0 \\ 0 \end{bmatrix}, \qquad (10.4)$$

$$\begin{bmatrix} \mathbf{c}_{12} \\ \mathbf{c}_{22} \\ \mathbf{c}_{32} \\ \mathbf{c}_{42} \\ \mathbf{c}_{52} \\ \mathbf{c}_{62} \end{bmatrix} = \frac{\mu}{9J} \begin{bmatrix} -4b^*_{11} - 4b^*_{22} + 2b^*_{33} \\ 2b^*_{11} + 8b^*_{22} + 2b^*_{33} \\ 2b^*_{11} - 4b^*_{22} - 4b^*_{33} \\ 3b^*_{12} \\ -6b^*_{13} \\ 3b^*_{23} \end{bmatrix} + \kappa \begin{bmatrix} J \\ J \\ J \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{10.5}$$

$$\begin{bmatrix} \mathbf{c}_{13} \\ \mathbf{c}_{23} \\ \mathbf{c}_{33} \\ \mathbf{c}_{43} \\ \mathbf{c}_{53} \\ \mathbf{c}_{63} \end{bmatrix} = \frac{\mu}{9J} \begin{bmatrix} -4b^*_{11} + 2b^*_{22} - 4b^*_{33} \\ 2b^*_{11} - 4b^*_{22} - 4b^*_{33} \\ 2b^*_{11} + 2b^*_{22} + 8b^*_{33} \\ -6b^*_{12} \\ 3b^*_{13} \\ 3b^*_{23} \end{bmatrix} + \kappa \begin{bmatrix} J \\ J \\ J \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{10.6}$$

$$\begin{bmatrix} \mathbf{c}_{14} \\ \mathbf{c}_{24} \\ \mathbf{c}_{34} \\ \mathbf{c}_{44} \\ \mathbf{c}_{54} \\ \mathbf{c}_{64} \end{bmatrix} = \frac{\mu}{6J} \begin{bmatrix} 2b^*_{12} \\ 2b^*_{12} \\ -4b^*_{12} \\ 3b^*_{11} + 3b^*_{22} \\ 3b^*_{23} \\ 3b^*_{13} \end{bmatrix} \tag{10.7}$$

$$\begin{bmatrix} \mathbf{c}_{15} \\ \mathbf{c}_{25} \\ \mathbf{c}_{35} \\ \mathbf{c}_{45} \\ \mathbf{c}_{55} \\ \mathbf{c}_{65} \end{bmatrix} = \frac{\mu}{6J} \begin{bmatrix} 2b^*_{13} \\ -4b^*_{13} \\ 2b^*_{13} \\ 3b^*_{23} \\ 3b^*_{11} + 3b^*_{33} \\ 3b^*_{12} \end{bmatrix}, \tag{10.8}$$

$$\begin{bmatrix} \mathbf{c}_{16} \\ \mathbf{c}_{26} \\ \mathbf{c}_{36} \\ \mathbf{c}_{46} \\ \mathbf{c}_{56} \\ \mathbf{c}_{66} \end{bmatrix} = \frac{\mu}{6J} \begin{bmatrix} -4b^*_{23} \\ 2b^*_{23} \\ 2b^*_{23} \\ 3b^*_{13} \\ 3b^*_{12} \\ 3b^*_{22} + 3b^*_{33} \end{bmatrix}. \tag{10.9}$$

```
!
! FILE:
!   umat_NH.F90
! AUTHOR:
!   Jorgen Bergstrom, Ph.D.
!   Copyright Jorgen Bergstrom.
! CONTENTS:
!   Abaqus UMAT subroutine for the Neo-Hookean (NH) model.
!   The subroutine is an example and not is not designed to
!   be a commercial quality implementation.
! USAGE:
!   2 material constants:
!      1: mu    (shear modulus)
!      2: kappa (bulk modulus)

subroutine umatErr(str)
    implicit none
    character(LEN=*), intent(in) :: str
    print '(a,a)', "UMAT Error:    ", trim(str)
    stop 3
end subroutine umatErr

subroutine umat(strVec, statev, ddsdde, &
        energyElast, energyPlast, energyVisc, &
        rpl, ddsddt, drplde, drpldt, stran, dstran, time, dtime, &
        temp, dtemp, predef, dpred, cmname, ndi, nshr, ntens, &
        nstatev, inProps, nrInProps, coords, drot, pnewdt, celent, &
        dfgrd0, dfgrd1, noel, npt, layer, kspt, kstep, kinc)
    implicit none
    integer, intent (in) :: ndi, nshr, ntens, nstatev, nrInProps, &
        noel, npt, layer, kspt, kstep, kinc
    character(len=8), intent(in) :: cmname
    real, intent(inout) :: strVec(ntens), statev(nstatev)
    real, intent(inout) :: energyElast, energyPlast, energyVisc
    real, intent(out) :: ddsdde(ntens,ntens), rpl, &
        ddsddt(ntens), drplde(ntens), drpldt
    real, intent(in) :: stran(ntens), dstran(ntens), time(2), dtime, &
        temp, dtemp, predef(1), dpred(1)
    real, intent(in) :: inProps(nrInProps), coords(3), drot(3,3)
    real, intent(out) :: pnewdt
    real, intent(in) :: celent, dfgrd0(3,3)
    real, intent(inout) :: dfgrd1(3,3)

    ! local variables
    real :: J, a1, a2, kk, F(3,3), b(3,3), bs(3,3), Stress(3,3),
    devbs(3,3)
    real :: tmp, Eye(3,3), mu, kappa, I1s

    ! setup variables
    if (nrInProps /= 2) call umatErr("Wrong number of input parameters")
    mu = inProps(1)
    kappa = inProps(2)
    if (mu < 0) call umatErr("Parameter mu has to be positive")
    if (kappa < 0) call umatErr("Parameter kappa has to be positive")
    if (ndi /= 3 .or. nshr /= 3) call umatErr("This umat only works for
    3D elements")
    pnewdt=2.0

    ! take the time-step
    F = dfgrd1
    J = F(1,1) * (F(2,2)*F(3,3) - F(2,3)*F(3,2)) + &
        F(1,2) * (F(2,3)*F(3,1) - F(2,1)*F(3,3)) + &
        F(1,3) * (F(2,1)*F(3,2) - F(2,2)*F(3,1))

    b(1,1) = F(1,1)*F(1,1) + F(1,2)*F(1,2) + F(1,3)*F(1,3)
    b(1,2) = F(1,1)*F(2,1) + F(1,2)*F(2,2) + F(1,3)*F(2,3)
```

```
b(1,3) = F(1,1)*F(3,1) + F(1,2)*F(3,2) + F(1,3)*F(3,3)
b(2,1) = b(1,2)
b(2,2) = F(2,1)*F(2,1) + F(2,2)*F(2,2) + F(2,3)*F(2,3)
b(2,3) = F(2,1)*F(3,1) + F(2,2)*F(3,2) + F(2,3)*F(3,3)
b(3,1) = b(1,3)
b(3,2) = b(2,3)
b(3,3) = F(3,1)*F(3,1) + F(3,2)*F(3,2) + F(3,3)*F(3,3)

bs = J**(-2.0/3.0) * b

tmp = bs(1,1) + bs(2,2) + bs(3,3)
devbs = bs
devbs(1,1) = bs(1,1) - tmp/3.0
devbs(2,2) = bs(2,2) - tmp/3.0
devbs(3,3) = bs(3,3) - tmp/3.0

Eye = 0.0
Eye(1,1) = 1.0
Eye(2,2) = 1.0
Eye(3,3) = 1.0

Stress = mu/J * devbs + kappa*(J-1.0) * Eye

I1s = bs(1,1) + bs(2,2) + bs(3,3)
energyElast = 0.5*mu*(I1s - 3.0) + 0.5*kappa*(J-1.0)**2.0

! calculate the Jacobian
a1 = mu / (9.0 * J)
a2 = mu / (6.0 * J)
kk = kappa * J

ddsdde(1,1) = a1 * ( 8*bs(1,1) + 2*bs(2,2) + 2*bs(3,3)) + kk
ddsdde(2,1) = a1 * (-4*bs(1,1) - 4*bs(2,2) + 2*bs(3,3)) + kk
ddsdde(3,1) = a1 * (-4*bs(1,1) + 2*bs(2,2) - 4*bs(3,3)) + kk
ddsdde(4,1) = a1 * 3*bs(1,2)
ddsdde(5,1) = a1 * 3*bs(1,3)
ddsdde(6,1) = a1 * (-6)*bs(2,3)

ddsdde(1,2) = a1 * (-4*bs(1,1) - 4*bs(2,2) + 2*bs(3,3)) + kk
ddsdde(2,2) = a1 * ( 2*bs(1,1) + 8*bs(2,2) + 2*bs(3,3)) + kk
ddsdde(3,2) = a1 * ( 2*bs(1,1) - 4*bs(2,2) - 4*bs(3,3)) + kk
ddsdde(4,2) = a1 * 3*bs(1,2)
ddsdde(5,2) = a1 * (-6)*bs(1,3)
ddsdde(6,2) = a1 * 3*bs(2,3)

ddsdde(1,3) = a1 * (-4*bs(1,1) + 2*bs(2,2) - 4*bs(3,3)) + kk
ddsdde(2,3) = a1 * ( 2*bs(1,1) - 4*bs(2,2) - 4*bs(3,3)) + kk
ddsdde(3,3) = a1 * ( 2*bs(1,1) + 2*bs(2,2) + 8*bs(3,3)) + kk
ddsdde(4,3) = a1 * (-6)*bs(1,2)
ddsdde(5,3) = a1 * 3*bs(1,3)
ddsdde(6,3) = a1 * 3*bs(2,3)

ddsdde(1,4) = a2 * 2*bs(1,2)
ddsdde(2,4) = a2 * 2*bs(1,2)
ddsdde(3,4) = a2 * (-4)*bs(1,2)
ddsdde(4,4) = a2 * 3*(bs(1,1) + bs(2,2))
ddsdde(5,4) = a2 * 3*bs(2,3)
ddsdde(6,4) = a2 * 3*bs(1,3)

ddsdde(1,5) = a2 * 2*bs(1,3)
ddsdde(2,5) = a2 * (-4)*bs(1,3)
ddsdde(3,5) = a2 * 2*bs(1,3)
ddsdde(4,5) = a2 * 3*bs(2,3)
ddsdde(5,5) = a2 * 3*(bs(1,1) + bs(3,3))
ddsdde(6,5) = a2 * 3*bs(1,2)
```

```
    ddsdde(1,6) = a2 * (-4)*bs(2,3)
    ddsdde(2,6) = a2 * 2*bs(2,3)
    ddsdde(3,6) = a2 * 2*bs(2,3)
    ddsdde(4,6) = a2 * 3*bs(1,3)
    ddsdde(5,6) = a2 * 3*bs(1,2)
    ddsdde(6,6) = a2 * 3*(bs(2,2) + bs(3,3))

    ! return variables:
    strVec(1) = Stress(1,1)
    strVec(2) = Stress(2,2)
    strVec(3) = Stress(3,3)
    strVec(4) = Stress(1,2)
    strVec(5) = Stress(1,3)
    strVec(6) = Stress(2,3)
    energyPlast = 0.0
    energyVisc = 0.0
    rpl = 0.0
    ddsddt(1) = 0.0
    drplde = 0.0
    drpldt = 0.0
end subroutine umat
```

Reference

[1] PolyUMod, http://PolyUMod.com/.