Explanation of Code for Implementation of Viscoelastic-Viscoplastic Constitutive Law for Momoh et al., 2024: Volumetric dissipation in geodynamic models and its impact on strain localization

Ekeabino Momoh[1*] Harsha S. Bhat[2], Stephen Tait[1,3] and Muriel Gerbault[1]

[1] *Geosciences Environnement Toulouse - GET, Université Toulouse III - Paul Sabatier, CNRS UMR 5563, 31400 Toulouse, France*

[2] *Laboratoire de Geologie, École Normale Supérieure, CNRS UMR 8538, PSL Research University, Paris, 75005 Paris, France*

[3] *Laboratoire Dynamiques des Fluides Géologiques, Institut de Physique du Globe de Paris, UMR 7154-CNRS, Université de Paris, 75005 Paris, France*

**Email: ekeabino.momoh@get.omp.eu, ekemomoh@gmail.com*

1 Introduction

This manual explains the structure of the User Material Mechanical (UMAT) subroutine and the User Material Heat Transfer (UMATHT) subroutine.

The explanation is based onthe subroutine ve_vp_umat_umatht.f which combined the mechanical and material subroutines. The benchmarking codes also followed the same template.

The core of the code is to update stresses, whether for elastic, viscous or viscoplastic flow laws, update the temperature during the irreversible deformation by estimating the heat production rate, and computing consistent algorithmic tangent moduli.

We commence from an elastic state, go into viscous creep and check for plastic flow. The subroutines are run in Abaqus finite element solver, and used an element library with temperature and displacements to enable an efficient interfacing of the mechanical and thermal subroutines. The code also works for heat transfer elements (if one is interested in solving solely a heat transfer problem) and elements which do not include temperature degrees of freedom. The drawback of using such elements is that temperature is not updated. Therefore, we utilized in Abaqus, the Coupled Temperature Displacement Analysis in the loading step to couple mechanical and thermal analysis within a time step.

In this manual, we explain the structure of the code. We refer the reader to Appendix A1 of the manuscript referred where we discussed the details of the computational implementation. The pseudo-code illustrated in Algorithm 1 of the manuscript summarizes the computational details implemented.

2 Interface between User material Fortran subroutine and Manuscript

To use the subroutines, we need the following:

1. An input (.inp) file
2. A fortran subroutine
3. Abaqus installation with appropriate linking to Intel oneAPI compiler and Microsoft Visual Studio in order to run the subroutines.

We explain the interface of specific parameters mentioned in the manuscript to the Fortran subroutine and the corresponding value either in the input files or given by Abaqus.

Parameter	Paper	UMAT	Default value
Elastic properties			
Young's modulus	E	E	PROPS(1)
Poisson ratio	v	v	PROPS(2)
Initial cohesion	c_0	c_0	PROPS(3)
Derived from elastic properties			
Bulk modulus	K	EBULK	
Shear modulus	G	G	
Thermal properties			
Thermal conductivity	λ	COND	PROPS(1)
Specific heat capacity	C_p	C_P	PROPS(2)
Drucker-Prager parameters			
Friction angle	ϕ	phi_f	PROPS(7)
Dilatancy angle	ψ	psi	PROPS(14)

Table 1: Input mechanical parameters for user material

Parameter	Symbol	UMAT	Default value
Creep exponent	m	PWR	PROPS(5)
Creep activation energy	E_a	EA	PROPS(10)
Molecular gas constant	R	R	PROPS(11)
Pre-exponential factor	A	A	PROPS(12)
Relative rate of vp strain	μ	PLAST	PROPS(13)
Time increment	Δt	DTIME	Abaqus

Table 2: Creep parameters for user material

Parameter	Paper	UMATHT	Default value
Thermal properties			
Thermal conductivity	λ	COND	PROPS(1)
Specific heat capacity	C_p	C_P	PROPS(2)

Table 3: Input thermal parameters for user material

Parameter	Symbol	UMAT
Elastic trial state		
Elastic stiffness matrix	C_e	DDSDDE_ELAS
Trial elastic stresses	$\boldsymbol{\sigma}^{\text{trial}}$	STRESS
Trial pressure	$P^{\text{e trial}}$	PTRIAL
Trial deviatoric stresses	$\boldsymbol{s}^{\text{e trial}}$	STRIAL
Trial elastic stress invariant	$\sqrt{J_{\text{II}}(\boldsymbol{s}^{\text{e trial}})}$	DEVJ2
Trial elastic strains	$\boldsymbol{\varepsilon}^{\text{trial}}$	STRAN
Elastic strain increments	$\Delta\boldsymbol{\varepsilon}$	DSTRAN
Trial elastic deviatoric strains	$e^{\text{e trial}}$	DEVSTRAN
Trial elastic deviatoric strains magnitude	$e^{\text{trial}}_{\text{norm}}$	ENORM
In the creep state		
Creep multiplier	$\Delta\gamma^{\text{v}}$	DGAMA_VISC
Creep residual function	$\tilde{\Phi}^{\text{v}}$	R_DGAMA_VISC
Updated deviatoric (viscous) stresses	$\boldsymbol{s}^{\text{v}}$	VSTRESS_DEV
Updated deviatoric stress invariant	$\sqrt{J_{\text{II}}(\boldsymbol{s}^{\text{v}})}$	SQRTJ2
Creep strain increments	$\Delta\varepsilon^{\text{v}}$	DEPSILON_V
In the Drucker-Prager yield state		
Yield potential	Φ^{DP}	PHI_YDP
Viscoplastic residual function	$\tilde{\Phi}^{\text{DP}}_{\text{R}}$	PHI_YDP_R
Cohesion	c_0	c0
Viscoplastic strain multiplier	$\Delta\gamma^{\text{vp}}$	DGAMA_VP
Updated deviatoric stresses	$\boldsymbol{s}$	S
Viscoplastic strain increments	$\Delta\boldsymbol{\epsilon}^{\text{vp}}$	DEPSILON_VP
Viscoplastic strains	$\boldsymbol{\varepsilon}^{\text{vp}}$	EPSILON_VP
Temperature	T	TEMP
Internal heat production	$\boldsymbol{\sigma}\dot{\boldsymbol{\varepsilon}}^{\text{v+vp}}$	RPL
Convergence matrix-related matrices		
2nd order identity matrix	$\boldsymbol{I}$	SOID
Deviatoric projection matrix	$\boldsymbol{I}_d$	DVPRJT
Consistent Jacobian Matrix	$\dfrac{\partial\boldsymbol{\sigma}}{\partial\boldsymbol{\varepsilon}^{\text{e trial}}}$	DDSDDE
Invariants		
Total volumetric strain rate invariant	$\dot{\boldsymbol{\varepsilon}}_{\text{I}}$	DOT_VPSTRAN_1ST_INV
Total deviatoric strain rate invariant	$\dot{\boldsymbol{\varepsilon}}_{\text{II}}$	DOT_VPSTRAN_2ND_INV
Visco-plastic volumetric strain rate invariant		VPSTRAN_1ST_INV
Visco-plastic deviatoric strain rate invariant		VPSTRAN_2ND_INV

Table 4: Parameters updated by UMAT

3 Viscoelascity-Viscoplasticity using Power-law Creep and Drucker-Prager Yielding

3.1 Initialization parameters and variables

3.1.1 Preamble

```
* ##########################################################
* USER SUBROUTINE TO COMPUTE DRUCKER−PRAGER VISCOPLASTICITY WITH
* CONSISTENT JACOBIAN OBTAINED BY LINEARIZING THE RETURN MAPPING
* EQUATION (RESIDUAL VISCOPLASTIC POTENTIAL FUCNTION).
*
* WE USE DISLOCATION CREEP TO MODEL HIGH−TEMPERATURE DUCTILE DEFORMATION.
*
* WRITTEN BY EKEABINO MOMOH, HARSHA BHAT AND STEVE TAIT
* ##########################################################
```

3.1.2 User Material (UMAT) Preamble

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

Header, description of quantities given and quantities to be updated.

- DDSDDE = consistent Jacobian defines the change in the Ith stress caused by a small change in the Jth strain component

  This quantity must be updated.

- STRESS = Stress given at the beginning of the increment to be be updated at the end of the increment.

3.1.3 Size of Abaqus Variables

Declare dimensions of some quantities:

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

```
DIMENSION DEVSTRAN(NTENS),S(NTENS),STRIAL(NTENS),SOID(NTENS),
1     DVPRJT(NTENS,NTENS),VSTRESS_DEV(NTENS), EPSILON_VP(NTENS),
2     DEPSILON_VP(NTENS), SPRJD(NTENS,NTENS),ELAS(NTENS),
3     HC(NTENS),DEPSILON_V(NTENS), EPSILON_V(NTENS), DDSDDE_V(NTENS,NTENS)
```

3.1.4 Parameters used

These parameters remain unchanged during the implementation of this UMAT

```
PARAMETER (R0=0.0D0, P01=0.1D0, R1=1.0D0, R2=2.0D0, R3=3.0D0,
1         R4=4.0D0, R6=6.0D0, R7=7.0D0,R9=9.0D0, R12=12.0D0,
2         R40=40.0D0,P13=1.0D0/3.0D0, TOL=1E-6, INIT_GUESS=1E-21,
3         P05=0.5D0,P098=0.98D0,P23=2.0D0/3.0D0,P06=6.0D0/10.0D0,
4         P07=7.0D0/10.0D0, A11=1085.7D0,A22=132.9D0,A33=-5.1D0,
5         B11=1475.0D0,B22=80.D0,B33=-3.2D0, PWRFPT=1.5D0, TQC=1.0D0)
```

3.1.5 Initialize variables

```
* ##########################
*   INITIALIZE STATE VARIABLES VARIABLES
* ##########################
        EPSILON_V=R0
        EPSILON_VP=R0
        DEPSILON_V=R0
        DEPSILON_VP=R0
        PHI_Y=R0
        DGAMA_VISC=R0
        DGAMA_VISC_OLD=R0
        DGAMA_VP=R0
        c=R0
        FPT=R0
        T_SOLIDUS=R0
        T_LIQUIDUS=R0
        GAMMA_BAR=R0
        RPL=R0
        RPL_VOL=R0
        RPL_SHEAR=R0
        DOT_VPSTRAN_1ST_INV=R0
        VPSTRAN_1ST_INV=R0
        VPSTRAN_2ND_INV=R0
        EPSILON_BAR_C=R0
        DTEMPERATURE_VOL=R0
        DTEMPERATURE_SHEAR=R0
        VISCOSITYY=R0
        PHI_YDP=R0
        DO I=1, NTENS
                EPSILON_VP(I)=STATEV(I)
*       SDV 1-4
        END DO
```

3.1.6 Retrieve stored variables

```
* CALL HARDENING HISTORY FROM PREVIOUS TIME STEP
     GAMMA_BAR=STATEV(NTENS+1)
*       SDV 5     ! DEFORMATION HISTORY
        DGAMA_VISC_OLD=STATEV(NTENS+6)
*       SDV 6     !
        VPSTRAN_1ST_INV=STATEV(NTENS+11)
*       SDV 13    ! FIRST INVARIANT OF VISCOPLASTIC STRAIN
           VPSTRAN_2ND_INV=STATEV(NTENS+12)
*       SDV 14    ! SECOND INVARIANT OF VISCOPLASTIC STRAIN
*       INITIAL TEMPERATURE
        IF(KINC==1)THEN
          STATEV(NTENS+14)=TEMP
        END IF
        TEMP_INIT=STATEV(NTENS+14)

        EPSILON_V(1)=STATEV(NTENS+16)
*      SDV 20    ! VISCOUS STRAINS(11)
        EPSILON_V(2)=STATEV(NTENS+17)
*          SDV 21    ! VISCOUS STRAINS(22)
        EPSILON_V(3)=STATEV(NTENS+18)
*          SDV 22    ! VISCOUS STRAINS(33)
        EPSILON_V(4)=STATEV(NTENS+19)
*       SDV 23    ! VISCOUS STRAINS(12)
```

3.1.7 Retrieve material constants from input files

```
*  #########################
*    RETRIEVE RHEOLOGICAL PROPERTIES
*  #########################
      E=PROPS(1)
*    YOUNG'S MODULUS
      v=PROPS(2)
*      POISSON'S RATIO
      c_0=PROPS(3)
*      INITIAL COHESION
      H=PROPS(4)
*    HARDENING MODULUS
      PWR=PROPS(5)
*    STRESS EXPONENT
      phi_i=PROPS(6)
*    INITIAL FRICTION ANGLE
      phi_f=PROPS(7)
*    FINAL FRICTION ANGLE
*
*
*
      RHOD_REF=PROPS(8)
*      REFERENCE DENSITY

    C_P=PROPS(9)
*    SPECIFIC HEAT CAPACITY
      E_A=PROPS(10)
*      ACTIVATION ENERGY
      R=PROPS(11)
*     MOLECULAR GAS CONSTANT
      A=PROPS(12)
*      CREEP PRE-EXPONENTIAL CONSTANT
      PLAST=PROPS(13)
*     RELATIVE RATE OF VISCOPLASTIC STRAIN
      psi=PROPS(14)
*      DILATANCY ANGLE
      TREF=PROPS(15)
*      REFERENCE TEMPERATURE
      ALPH_LE=PROPS(16)
*      COEFFICIENT OF LINEAR EXPANSION

      EPSILON_BAR_C=PROPS(17)
*      CRITICAL EFFECTIVE PLASTIC STRAIN
*  #########################
```

3.1.8 Identifier in the manuscript

The identifier in the code is written in red, while the identifier in the manuscript is written in black for one-to-one correspondence.

E (E) = Young's modulus

v (v) = Poisson's ratio

c_0 (c_0) = Initial cohesion (if it is used)

H (H) = Hardening modulus

PWR (m) = Power for dislocation creep or viscoplasticity

phi_i (ϕ_i) = Initial friction angle if friction hardening or softening is used according to Leroy & Ortiz (1989) or constant friction angle if not used. phi_f (ϕ_f) is the friction angle after a prescribed strain is reached.

rho_0 (ρ_0) = reference density if one is interested in density variation with temperature

C_p (C_p) = Specific heat capacity

E_a (E_a) = Activation energy

R (R) = Molecular gas constant

A (A) = Pre-exponential factor

PLAST (μ) = Relative rate of viscoplastic strain

psi (ψ)=dilation angle

TREF (T_{ref})=Reference temperature if used

ALPH_LE (α) = Coefficient of expansivity if used

EPSILON_BAR_C ($\bar{\varepsilon}$) = Saturation strain if frictional softening/hardening if used

3.1.9 Compute material-dependent variables

```
* ######################
*    COMPUTE QUANTITIES DEPENDENT ON THE RHEOLOGICAL PROPERTIES
*     SET FRICTION ANGLE AS A FUNCTION OF EFFECTIVE PLASTIC STRAIN RATE
*     CONSTAN=R2*(SIND(phi_f)-SIND(phi_i))*SQRT(EPSILON_BAR_C)
*     phi=ASIND(SIND(phi_i)+((CONSTAN*VPSTRAN_EFF)/(VPSTRAN_EFF
*    1    +EPSILON_BAR_C)))
      EBULK=E/(R3*(R1-(R2*v)))
*    BULK MODULUS
      EBULK3=E/((R1-(R2*v)))
      G=(E/(R2*(R1+v)))
*        SHEAR MODULUS
      R2G=R2*G
      ALPH_1=R3*TAND(phi_f)/(SQRT(R9+R12*(TAND(phi_f))**2))
      ALPH_2=R3/(SQRT(R9+R12*(TAND(phi_f))**2))
      ALPH_3=(R3*TAND(psi))/(SQRT(R9+R12*(TAND(psi))**2))
      DILATANCY_ANGLE=psi
```

3.1.10 Identifier in the manuscript

These quantities depend on the material properties extracted from the input files.

EBULK (BULK MODULUS), $K = \dfrac{E}{3(1-2v)}$.

G (SHEAR MODULUS), $G = \dfrac{E}{2(1+v)}$.

$\alpha_1 = \dfrac{3\tan\phi}{\sqrt{9+12\tan^2\phi}}$, $\alpha_2 = \dfrac{3}{\sqrt{9+12\tan^2\phi}}$, $\alpha_3 = \dfrac{3\tan\psi}{\sqrt{9+12\tan^2\psi}}$.

α_1, α_2 and α_3 are material dependent constants depending on the friction angle and dilation angle.

3.1.11 Declare some arrays

```
*  #######################################################
*   INITALIZE MATRICES AND ARRAYS
*  #######################################################
*   C^e          ----> FOR THE ELASTIC STATE, DDSDDE IS THE ELASTICITY MATRIX
*   DVPRJT       ----> DEVIATORIC PROJECTION MATRIX
*   SOID         ----> SECOND ORDER IDENTITY MATRIX SAVED IN ARRAY FORM
*   SPRJD        ----> FOURTH ORDER SYMMETRIC IDENTITY MATRIX
*   DEPSILON_VP  ----> VISCOPLASTIC STRAIN INCREMENT
*   TINC         ----> TEMPERATURE INCREMENT SAVED IN ARRAY FORM
      DO I=1,NTENS
       DO J=1,NTENS
       SPRJD(I,J)=R0
       DVPRJT(I,J)=R0
       END DO
         DEPSILON_VP(I)=R0
       DEPSILON_V(I)=R0
       SOID(I)=R0
       S(I)=R0
       ELAS(I)=R0
       STRIAL(I)=R0
       HC(I)=R0
      END DO
*       INITIALIZE STRAIN RATE INVARIANTS
      DOT_VPSTRAN_INV=R0
      DOT_VPSTRAN_1ST_INV=R0
      DOT_VPSTRAN_2ND_INV=R0
*   INITIALIZE TEMPERATURE CHANGE
      DT0=R0
*  #######################################################
*   ALLOCATE VALUES TO MATRICES AND ARRAYS
*   SPRJD      ----> FOURTH ORDER SYMMETRIC IDENTITY MATRIX
*   SOID      ----> SECOND ORDER IDENTITY MATRIX SAVED IN ARRAY FORM
*  #######################################################
      DO I=1,NDI
       SPRJD(I,I)=R1
       SOID(I)=R1
      END DO
      DO I=NDI+1,NTENS
       SPRJD(I,I)=P05
       SOID(I)=R0
      END DO
*   DVPRJT      ----> DEVIATORIC PROJECTION MATRIX
      DO  M=1,NTENS
       DO  N=1,NTENS
       DVPRJT(M,N)=SPRJD(M,N)-(SOID(M)*SOID(N)/R3)
       END DO
      END DO
```

DDSDDE is the Jacobian matrix which is the elastic stiffness matrix in the linear elastic case (C_e), DVPRJT ($\boldsymbol{I}_d$) is the deviatoric projection matrix, SPRJD (I) is the symmetric identity matrix.

We calculated the components of the deviatoric projection matrix using a simple Matlab script shown below. These terms arise from:

$$\boldsymbol{I}_d = \left(\frac{1}{2}\left(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}\right) - \frac{1}{3}\delta_{ij}\delta_{kl}\right)$$

3.1.12 Matlab script for deviatoric projection matrix

```
clear all
close all
clc
A11=[1,1,1,1,1,1,2,2,1,1,3,3,1,1,1,2];
A22=[2,2,1,1,2,2,2,2,2,2,3,3,2,2,1,2];
A33=[3,3,1,1,3,3,2,2, 3,3,3,3,3,3,1,2];
A12=[1,2,1,1,1,2,2,2,1,2,3,3,1,2,1,2];
% Each element in the vector represents indices in a given row
A=[A11;A22;A33;A12]; % Concatenate all elements
B=zeros(4,4); % Initialize Deviatoric projection matrix
C=B;
M=1;
while M<=4;
    N=1;
    Q=1;
    while ((Q+3)<=length(A(1,:)))&&(N<=4);
        I=A(M,Q);
        J=A(M,Q+1);
        K=A(M,Q+2);
        L=A(M,Q+3);
        if I==K && J==L
            D1=1;
        else
            D1=0;
        end
        if I==L && J==K
            D2=1;
        else
            D2=0;
        end
        if I==J && K==L
            D3=1;
        else
            D3=0;
        end
        B(M,N)=0.5*(D1+D2)-(1/3*D3);
        C(M,N)=D3;
        Q=Q+4;
        N=N+1;
    end
    M=M+1;
end
```

This Matlab script calculates the elements of the deviatoric projection matrix.

Useful identities:

$\mathbf{I} = \mathbf{I}_{ijkl} = \delta_{ik}\delta_{jl}$ fourth order identitiy tensor

$\mathbf{I}_s = \mathbf{I}_{ijkl} = \frac{1}{2}(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk})$,$\mathbf{I}_s$ is the fourth order symmetric identity tensor

$\boldsymbol{I}_d = \mathbf{I}_s - \frac{1}{3}\boldsymbol{I}\otimes\boldsymbol{I}, (\boldsymbol{I}\otimes\boldsymbol{I})_{ijkl} = \delta_{ij}\delta_{kl}, \boldsymbol{I}$ is the second order identity tensor

$$\boldsymbol{I}_d = \left(\frac{1}{2}(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}) - \frac{1}{3}\delta_{ij}\delta_{kl}\right)$$

Components of $\boldsymbol{I}_d$ are expressed as:

$$\begin{pmatrix} I_{1111} & I_{1122} & I_{1133} & I_{1112} \\ I_{2211} & I_{2222} & I_{2233} & I_{2212} \\ I_{3311} & I_{3322} & I_{3333} & I_{3312} \\ I_{1211} & I_{1222} & I_{1233} & I_{1212} \end{pmatrix} =$$

$$\begin{pmatrix} 0.6667 & -0.3333 & -0.3333 & 0.0000 \\ -0.3333 & 0.6667 & -0.3333 & 0.0000 \\ -0.3333 & -0.3333 & 0.6667 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.5000 \end{pmatrix}$$

We extracted the indices of the components and used the matlab script to evaluate $\boldsymbol{I}_d = \left(\frac{1}{2}(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}) - \frac{1}{3}\delta_{ij}\delta_{kl}\right)$

3.2 Elastic case

3.2.1 Elastic stiffness matrix UMAT

```
* #############################
*      DEFINE THE ELASTICITY MATRIX (C^e) I.E., THE CONSISTENT JACOBIAN (DDSDDE) FOR ELASTICITY PROBLEMS
* #############################
IF(NDI.EQ.3. AND. NSHR.EQ.1)THEN
*      PLANE STRAIN/AXISYMMETRIC PROBLEMS
         DDSDDE(1,1)=R1-v
         DDSDDE(2,2)=R1-v
         DDSDDE(3,3)=R1-v
         DDSDDE(4,4)=P05*(R1-(R2*v))
         DDSDDE(1,2)=v
         DDSDDE(1,3)=v
         DDSDDE(2,1)=v
         DDSDDE(2,3)=v
         DDSDDE(3,1)=v
         DDSDDE(3,2)=v
         DDSDDE=DDSDDE*E/((R1+v)*(R1-(R2*v)))
       ELSEIF(NDI.EQ.2 .AND. NSHR.EQ.1)THEN
*      PLANE STRESS PROBLEMS
         DDSDDE(1,1)=R1
         DDSDDE(2,2)=R1
         DDSDDE(3,3)=P05*(R1-v)
         DDSDDE(1,2)=v
         DDSDDE(2,1)=v
         DDSDDE=DDSDDE*E/(R1+(v*v))
       ELSE
*      3-D CONDITIONS
         DDSDDE(1,1)=R1-v
         DDSDDE(2,2)=R1-v
         DDSDDE(3,3)=R1-v
         DDSDDE(4,4)=P05*(R1-(R2*v))
         DDSDDE(5,5)=P05*(R1-(R2*v))
         DDSDDE(6,6)=P05*(R1-(R2*v))
         DDSDDE(1,2)=v
         DDSDDE(1,3)=v
         DDSDDE(2,1)=v
         DDSDDE(2,3)=v
         DDSDDE(3,1)=v
         DDSDDE(3,2)=v
         DDSDDE=DDSDDE*E/((R1+v)*(R1-(R2*v)))
       END IF
```

3.2.2 Elastic stiffness matrix in manuscript

This step assembles the elastic constants into the elastic stiffness matrix (also called the Jacobian (DDSDDE) only in the elastic case). We developed the code to solve for 2-D, 3-D or axisymmetric problems. The number of tensor components (NTENS) allows the code to switch into given problems.

For 2-D problems, we define the stiffness matrix as:

$$C^{\mathrm{e}} = 2G\boldsymbol{I}_s + \lambda_L \boldsymbol{I} \otimes \boldsymbol{I} =$$
$$2G\left(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}\right) + \lambda_L \delta_{ij}\delta_{kl} =$$
$$\frac{E}{(1+v)(1-2v)}\begin{pmatrix} 1-v & v & v & 0 \\ v & 1-v & v & 0 \\ v & v & 1-v & 0 \\ 0 & 0 & 0 & \frac{1}{2}(1-2v) \end{pmatrix}.$$

Where ALANDA, $\lambda_L = \dfrac{Ev}{(1+v)(1-2v)}$

3.2.3 Elastic state

```
*     ELASTIC STRESS STATE
*     DEPSILON        ----> DSTRAN
*     EPSILON^e       ----> ELAS
      DO I = 1, NTENS
       DO J = 1, NTENS
       STRESS(I)=STRESS(I)+(DDSDDE(I,J)*(DSTRAN(J)))
       END DO
       ELAS(I)=STRAN(I)+DSTRAN(I)
      END DO
      DOT_EPSTRAN_2ND_INV=R0
      DO M=1,NDI
        DOT_EPSTRAN_2ND_INV=DOT_EPSTRAN_2ND_INV+(DSTRAN(M)/DTIME)**R2
       END DO
      DO M=NDI+1,NTENS
       DOT_EPSTRAN_2ND_INV=DOT_EPSTRAN_2ND_INV+(R2*(DSTRAN(M)/DTIME)**R2)
      END DO
```

Given a time step n (t_n), our interest is finding the stress state at an updated time step (n + 1). We define the following for the elastic state when Abaqus provides an initial strain ($\boldsymbol{\varepsilon}_{n+1}^{\mathrm{e\ trial}}$) and corresponding strain increment $\Delta\varepsilon$ when a given load is applied at that time step.

$\boldsymbol{\varepsilon}_{n+1}^{\mathrm{e\ trial}} = \varepsilon_n^{\mathrm{e}} + \Delta\varepsilon$ represents the trial elastic strain

$\varepsilon_{n+1}^{\mathrm{p\ trial}} = \varepsilon_n^{p}$ represents the initial plastic strain

$\boldsymbol{\sigma}_{n+1}^{trial} = C^e \boldsymbol{\varepsilon}_{n+1}^{\mathrm{e\ trial}}$ represents the trial elastic stress

The second part computes the elastic strain rate invariant.

3.2.4 Assemble elastic quantities

Here we compute quantities from the elastic state that forms the input into the viscous correction step.

```
* ###########################
*  ASSEMBLE THE TRIAL QUANTITIES (PRESSURE, DEVIATORIC STRESSES, J2 AND ELASTIC STRAIN MAGNITUDE)
* ###########################
*    PTRIAL          ----> TRIAL ELASTIC PRESSURE
*    STRIAL          ----> TRIAL ELASTIC DEVIATORIC STRESS
*    DEVSTRAN        ----> TRIAL ELASTIC DEVIATORIC STRAIN
*    DEVSTRAN_NORM   ----> TRIAL ELASTIC DEVIATORIC STRAIN MAGNITUDE
*    DEVJ2           ----> J2
*    SQRTJ2          ----> SQUARE-ROOT OF J2
      PTRIAL=(STRESS(1)+STRESS(2)+STRESS(3))/R3
      DEVSTRAN_NORM=R0
      DEVSTRAN=R0
      DEVJ2=R0
      SQRTJ2=R0
      SQRTJ2T=R0
      DO I=1,NDI
        STRIAL(I)=STRESS(I)-(PTRIAL*SOID(I))
        DEVSTRAN(I)=STRIAL(I)/R2G
        DEVSTRAN_NORM=DEVSTRAN_NORM+DEVSTRAN(I)**2
        DEVJ2=DEVJ2+(STRIAL(I)*STRIAL(I))
      END DO
      DEVJ2=P05*DEVJ2
      DO I=NDI+1,NTENS
        STRIAL(I)=STRESS(I)-(PTRIAL*SOID(I))
        DEVSTRAN(I)=STRIAL(I)/G
        DEVSTRAN_NORM=DEVSTRAN_NORM+(R2*DEVSTRAN(I)**2)
        DEVJ2=DEVJ2+(STRIAL(I)*STRIAL(I))
      END DO
      DEVSTRAN_NORM=SQRT(DEVSTRAN_NORM)
      SQRTJ2=SQRT(DEVJ2)
      SQRTJ2T=SQRTJ2
      DOT_VPSTRAN_2ND_INV_ELAS=R0
      DO M=1,NDI
         DOT_VPSTRAN_2ND_INV_ELAS=DOT_VPSTRAN_2ND_INV_ELAS+(DSTRAN(M)/DTIME)**R2
      END DO
      DO M=NDI+1,NTENS
        DOT_VPSTRAN_2ND_INV_ELAS=DOT_VPSTRAN_2ND_INV_ELAS+(R2*(DSTRAN(M)/DTIME)**R2)
      END DO
```

PTRIAL, $P_{n+1}^{\text{trial}} = \frac{1}{3}\sigma_{kk}\delta_{ij}$

STRIAL, $s_{ij}^{\text{e trial}} = \sigma_{ij} - \frac{1}{3}\sigma_{kk}\delta_{ij}$ where σ_{ij} represents the total elastic stress components

DEVSTRAN, $e_{ij} = \frac{\boldsymbol{s}_{ij}}{2G}$ where e_{ij} represents deviatoric strains

EDNORM, $||\varepsilon_{d\ n+1}^{\text{e trial}}|| = \sqrt{\boldsymbol{e}_{n+1}^{\text{e trial}} : \boldsymbol{e}_{n+1}^{\text{e trial}}} = \sqrt{e_{ij}^{\text{trial}} e_{ij}^{\text{trial}}}$

SNORM, $||s|| = \sqrt{\boldsymbol{s}_{n+1}^{\text{trial}} : \boldsymbol{s}_{n+1}^{\text{trial}}} = \sqrt{s_{ij}^{\text{trial}} s_{ij}^{\text{trial}}}$

DEVJ2, $J_{\text{II}}(\boldsymbol{s}_{\text{d n+1}}^{\text{e trial}}) = \sqrt{\frac{1}{2} s_{ij}^{\text{e trial}} s_{ij}^{\text{e trial}}}$

3.2.5 Assemble elastic quantities

```

      DOT_VPSTRAN_2ND_INV_ELAS=P05*DOT_VPSTRAN_2ND_INV_ELAS
      P=PTRIAL
      DGAMA_VP=R0
      KK=1
*     c               ----> COHESION HARDENING
      c=c_0!+(H*GAMMA_BAR)
*     PHI_Y           ----> DRUCKER-PRAGER YIELD CRITERION
*      PHI_Y=SQRTJ2+(ALPH_1*PTRIAL)-(ALPH_2*c)
*     CRP             ----> DISLOCATION CREEP
*      CRP=((A*EXP(-E_A/(R*TEMP))))
*      PWC=G+(ALPH_1*ALPH_3*EBULK)+(ALPH_2*ALPH_2*H)
* ########################
*     CHECK IF THE MATERIAL YIELDS AND ACTIVATE PLASTICITY MODULE
*     R_DGAMA         ----> RESIDUAL YIELD FUNCTION (R(DGAMA))
*     DR_DGAMA        ----> DERIVATIVE OF RESIDUAL YIELD FUNCTION
* ########################
```

3.3 Creep state (Newton-Raphson and Bisection) and Drucker Prager Plasticity

This is the part where we implement the viscous correction and plastic correction.
For the two cases, we compute a consistent Jacobian depending on which mode is dominant.
The equations implemented here are Appendix A1 of the paper. We restate the equations

```
      IF(SQRTJ2.GT.1E1.AND.KSTEP.GT.R1)THEN
        CRP=A*EXP(-E_A/(R*TEMP))
*     INITIAL GUESSES FOR VISCOUS CREEP MULTIPLIERS
        DGAMA_OLD=R0
        DGAMA_VISC=R0
*     RESIDUAL VISCOUS CREEP EQUATION
      R_DGAMA_VISC=(DTIME*CRP)*(SQRTJ2-(G*DGAMA_VISC))**PWR-DGAMA_VISC
      DR_DGAMA_VISC=-G*PWR*(DTIME*CRP)*(SQRTJ2-(G*DGAMA_VISC))**(PWR-R1)-R1
      K=R1
      RESIDUALS=R1
*     ENTER NEWTON-RAPHSON LOOP TO COMPUTE THE VISCOUS CREEP MULTIPLIER
        DO WHILE(ABS(RESIDUALS).GT.TOL)
          R_DGAMA_VISC=(DTIME*CRP)*(SQRTJ2-(G*DGAMA_VISC))**PWR-DGAMA_VISC
          DR_DGAMA_VISC=-G*PWR*(DTIME*CRP)*(SQRTJ2-(G*DGAMA_VISC))**(PWR-R1)-R1
          DDGAMA=R_DGAMA_VISC/DR_DGAMA_VISC
          DGAMA_VISC=DGAMA_OLD-DDGAMA
          DJ_NEW=DGAMA_VISC
          IF(DGAMA_VISC.GE.SQRTJ2/G)THEN
            DJ_OLD=DGAMA_OLD
            DJ_NEW=DGAMA_VISC
            DO WHILE(RESIDUALS.GT.TOL .AND. DJ_NEW .GT. R0)
              DJ_NEW=(DJ_NEW-DGAMA_OLD)/R2
              R_DGAMA_VISC=(DTIME*CRP)*(SQRTJ2-(G*DJ_NEW))**PWR-DJ_NEW
              RESIDUALS=ABS(R_DGAMA_VISC)
            END DO
          ELSEIF(DGAMA_VISC.LT.R0)THEN
            DJ_OLD=R0
            DJ_NEW=SQRTJ2/G
            DO WHILE (RESIDUALS.GT.TOL)
              DJ_NEW=(DJ_NEW-DJ_OLD)/R2
              R_DGAMA_VISC=(DTIME*CRP)*(SQRTJ2-(G*DJ_NEW))**PWR-DJ_NEW
```

```
              RESIDUALS=ABS(R_DGAMA_VISC)
            END DO
          END IF
          DGAMA_VISC=DJ_NEW
          RESIDUALS=ABS(R_DGAMA_VISC)
          DGAMA_OLD=DGAMA_VISC
          K=K+1
       END DO
       IF(SQRTJ2.EQ.R0)THEN
          FACTOR=R1
       ELSE
          FACTOR=R1-(G*DGAMA_VISC/(SQRTJ2))
       END IF
       VSTRESS_DEV=R0
       VSTRESS_DEV_NORM=R0
       DEVJ2=R0
       DO I=1,NDI
         VSTRESS_DEV(I)=FACTOR*STRIAL(I)
         VSTRESS_DEV_NORM=VSTRESS_DEV_NORM+(VSTRESS_DEV(I))**R2
       END DO
       VSTRESS_DEV_NORM=P05*VSTRESS_DEV_NORM
       DO I=NDI+1,NTENS
         VSTRESS_DEV(I)=FACTOR*STRIAL(I)
         VSTRESS_DEV_NORM=VSTRESS_DEV_NORM+(VSTRESS_DEV(I))**R2
       END DO
       VSTRESS_DEV_NORM=SQRT(VSTRESS_DEV_NORM)
       SQRTJ2_V=VSTRESS_DEV_NORM
       DO I=1,NTENS
         STRESS(I)=VSTRESS_DEV(I)+PTRIAL*SOID(I)
         DEPSILON_V(I)=DGAMA_VISC*STRIAL(I)/(R2*SQRTJ2)
         EPSILON_V(I)=EPSILON_V(I)+DEPSILON_V(I)
       END DO
       DEVJ2=R0
       SQRTJ2T=R0
       DO I=1,NDI
         DEVJ2=DEVJ2+(VSTRESS_DEV(I)*VSTRESS_DEV(I))
       END DO
       DEVJ2=P05*DEVJ2
       DO I=NDI+1,NTENS
         DEVJ2=DEVJ2+(VSTRESS_DEV(I)*VSTRESS_DEV(I))
       END DO
       SQRTJ2T=SQRT(DEVJ2)
*     ASSEMBLE CONSISTENT JACOBIAN MATRIX FOR CREEP DEFORMATION
      b1=(SQRT(R2)*G*(DTIME*CRP)**(R1/PWR))/
     1  (((DGAMA_VISC)**((R1-PWR)/PWR))/PWR+G*(DTIME*CRP)**(R1/PWR))
       FVP1=R2G*(R1-DGAMA_VISC/(SQRT(R2)*DEVSTRAN_NORM))
       FVP2=SQRT(R2)*G*(DGAMA_VISC/DEVSTRAN_NORM-b1)
       DO M=1,NTENS
          DO N=1,NTENS
            DEV=FVP1*DVPRJT(M,N)+
     1       FVP2*DEVSTRAN(M)*DEVSTRAN(N)/(DEVSTRAN_NORM*DEVSTRAN_NORM)
            VOL=EBULK*SOID(M)*SOID(N)
            DDSDDE_V(M,N)=DEV
            DDSDDE(M,N)=DEV+VOL
          END DO
        END DO
*     CHECK FOR YIELD (VISCOPLASTICITY)
       PHI_YDP=SQRTJ2_V+(ALPH_1*PTRIAL)-(ALPH_2*c_0)
       c1=G+(ALPH_1*ALPH_3*EBULK)
```

```
        DEPSILON_VP=R0
        IF(PHI_YDP .GT. R0)THEN
*     INITIAL GUESSES FOR VISCOPLASTIC MULTIPLIERS
          DGAMA_OLD=INIT_GUESS
          DGAMA_VP=INIT_GUESS
*     RESIDUAL VISCOPLASTIC RETURN MAPPING EQUATION
          R_DGAMA_VP=(DTIME/PLAST)*((PHI_YDP-(c1*DGAMA_VP))/c_0)**PWR-DGAMA_VP
          DR_DGAMA_VP=-c1*PWR*DTIME/(c_0*PLAST)*((PHI_YDP-(c1*DGAMA_VP))
     1              /c_0)**(PWR-R1)-R1
          RESIDUALS=R1
*     ENTER NEWTON-RAPHSON LOOP TO COMPUTE THE VISCOPLASTIC MULTIPLIER
*     BY SOLVING THE RESIDUAL VISCOPLASTIC FUNCTION
          DO WHILE(ABS(RESIDUALS).GT.TOL)
            R_DGAMA_VP=(DTIME/PLAST)*((PHI_YDP-(c1*DGAMA_VP))/c_0)**PWR-DGAMA_VP
            DR_DGAMA_VP=-c1*PWR*DTIME/(c_0*PLAST)*((PHI_YDP-(c1*DGAMA_VP))
     1              /c_0)**(PWR-R1)-R1
            DDGAMA=R_DGAMA_VP/DR_DGAMA_VP
            DGAMA_VP=DGAMA_OLD-DDGAMA
            DJ_NEW=DGAMA_VP
            IF(DJ_NEW.GE.PHI_YDP/c1)THEN
              DJ_OLD=DGAMA_OLD
              DJ_NEW=DGAMA_VP
              DO WHILE(RESIDUALS.GT.TOL .AND. DJ_NEW .GT. R0)
                DJ_NEW=(DJ_NEW-DGAMA_OLD)/R2
                R_DGAMA_VP=(DTIME/PLAST)*((PHI_YDP-(c1*DJ_NEW))/c_0)**PWR-DJ_NEW
                RESIDUALS=ABS(R_DGAMA_VP)
              END DO
            ELSEIF(DGAMA_VP.LE.R0)THEN
              DJ_OLD=R0
              DJ_NEW=PHI_YDP/c1
              DO WHILE(RESIDUALS.GT.TOL)
                DJ_NEW=(DJ_NEW-DJ_OLD)/R2
                R_DGAMA_VP=(DTIME/PLAST)*((PHI_YDP-(c1*DJ_NEW))/c_0)**PWR-DJ_NEW
                RESIDUALS=ABS(R_DGAMA_VP)
              END DO
            END IF
            DGAMA_VP=DJ_NEW
            RESIDUALS=ABS(R_DGAMA_VP)
            DGAMA_OLD=DGAMA_VP
            K=K+1
          END DO
*     UPDATE STATE VARIABLES
          IF (SQRTJ2_V.EQ.R0)THEN
            FACTOR=R1
          ELSE
            FACTOR=R1-(G*DGAMA_VP/(SQRTJ2_V))
          END IF
*     UPDATE PRESSURE
          P=PTRIAL-(ALPH_3*EBULK*DGAMA_VP)
*     UPDATE DEVIATORIC STRESSES, STRESSES, INCREMENTS
          DO I=1,NTENS
            S(I)=FACTOR*VSTRESS_DEV(I)
            STRESS(I)=S(I)+(P*SOID(I))
            DEPSILON_VP(I)=DGAMA_VP*(VSTRESS_DEV(I)/(R2*SQRTJ2_V)
     1                  +(ALPH_3/R3*SOID(I)))
          END DO
          DEVJ2=R0
          SQRTJ2T=R0
          DO I=1,NDI
```

```fortran
          DEVJ2=DEVJ2+(S(I)*S(I))
        END DO
        DEVJ2=P05*DEVJ2
        DO I=NDI+1,NTENS
          DEVJ2=DEVJ2+(S(I)*S(I))
        END DO
        SQRTJ2T=SQRT(DEVJ2)
*     ASSEMBLE CONSISTENT JACOBIAN MATRIX FOR RETURN MAPPING TO THE SMOOTH PART OF THE DRUCKER-PRAGER CONE
        PHI_YDPR=SQRTJ2_V+(ALPH_1*PTRIAL)-(ALPH_2*c_0)-c1*DGAMA_VP
        b2=R1/c_0*(DTIME/PLAST)**(R1/PWR)/
     1       (((DGAMA_VP)**((R1-PWR)/PWR))
     2        /PWR+c1/c_0*(DTIME/PLAST)**(R1/PWR))
        FVP1=R1-G*DGAMA_VP/SQRTJ2_V
        FVP2=ALPH_1*b2*EBULK*G/(SQRTJ2_V)
        FVP3=G/SQRTJ2_V
        FVP4=SQRT(R2)*G*b2*(R1-b1/R2)
        FVP5=SQRT(R2)*DGAMA_VP/SQRTJ2_V*(b1/SQRT(R2)-R1)
        FVP6=R1-ALPH_1*ALPH_3*b2*EBULK
        FVP7=ALPH_3*b2*EBULK*(b1-SQRT(R2)*G)
        DO M=1,NTENS
          DO N=1,NTENS
            DEV=FVP1*DDSDDE_V(M,N)-
     1            FVP2*SOID(M)*VSTRESS_DEV(N)/(SQRTJ2_V)
     2            -FVP3*(FVP4*DEVSTRAN(M)/DEVSTRAN_NORM
     3            +(FVP5*DEVSTRAN(M)/DEVSTRAN_NORM))
     4             *VSTRESS_DEV(N)/(SQRTJ2_V)
            VOL=EBULK*FVP6*SOID(M)*SOID(N)
     1              +(FVP7*DEVSTRAN(M*SOID(M))/DEVSTRAN_NORM)
            DDSDDE(M,N)=DEV+VOL
        END DO
       END DO
*    UPDATE DEFORMATION AND HEAT GENERATION
      RPL=R0
      RPL_SHEAR=R0
      RPL_VOL=R0
      VPSTRAN_1ST_INV=R0
      DOT_VPSTRAN_2ND_INV=R0
      VPSTRAN_2ND_INV=R0
      DOT_VPSTRAN_EFF=R0
      DO M=1,NTENS
        EPSILON_VP(M)=EPSILON_VP(M)+DEPSILON_VP(M)
        RPL=RPL+(STRESS(M)*(DEPSILON_VP(M))/DTIME)
        VPSTRAN_1ST_INV=VPSTRAN_1ST_INV+EPSILON_VP(M)*SOID(M)
        DOT_VPSTRAN_1ST_INV=DOT_VPSTRAN_1ST_INV+(DEPSILON_VP(M))
     1                    /DTIME*SOID(M)
      END DO
      DO M=1,NDI
         DOT_VPSTRAN_2ND_INV=DOT_VPSTRAN_2ND_INV+((DEPSILON_VP(M))
     1                    /DTIME)**R2
         VPSTRAN_2ND_INV=VPSTRAN_2ND_INV+(EPSILON_VP(M))**R2
      END DO

      DO M=NDI+1,NTENS
        DOT_VPSTRAN_2ND_INV=DOT_VPSTRAN_2ND_INV+
     1                    (R2*((DEPSILON_VP(M))/DTIME)**R2)
        VPSTRAN_2ND_INV=VPSTRAN_2ND_INV+(R2*(EPSILON_VP(M))**R2)
      END DO
      DOT_VPSTRAN_2ND_INV=P05*DOT_VPSTRAN_2ND_INV
```

```
      VPSTRAN_2ND_INV=P05*VPSTRAN_2ND_INV
      DOT_VPSTRAN_2ND_INV=SQRT(DOT_VPSTRAN_2ND_INV)
      VPSTRAN_2ND_INV=SQRT(VPSTRAN_2ND_INV)
      ELSE
*   UPDATE DEFORMATION AND HEAT GENERATION
      RPL=R0
      RPL_SHEAR=R0
      RPL_VOL=R0
      VPSTRAN_1ST_INV=R0
      DOT_VPSTRAN_2ND_INV=R0
      VPSTRAN_2ND_INV=R0
      DOT_VPSTRAN_EFF=R0
      DO M=1,NTENS
        EPSILON_VP(M)=EPSILON_VP(M)+DEPSILON_V(M)
        RPL=RPL+(STRESS(M)*(DEPSILON_V(M))/DTIME)
        VPSTRAN_1ST_INV=VPSTRAN_1ST_INV+EPSILON_VP(M)*SOID(M)
        DOT_VPSTRAN_1ST_INV=DOT_VPSTRAN_1ST_INV+(DEPSILON_V(M))
     1                       /DTIME*SOID(M)
      END DO
      DO M=1,NDI
         DOT_VPSTRAN_2ND_INV=DOT_VPSTRAN_2ND_INV+((DEPSILON_V(M))
     1                       /DTIME)**R2
         VPSTRAN_2ND_INV=VPSTRAN_2ND_INV+(EPSILON_VP(M))**R2
      END DO

      DO M=NDI+1,NTENS
        DOT_VPSTRAN_2ND_INV=DOT_VPSTRAN_2ND_INV+
     1                       (R2*((DEPSILON_V(M))/DTIME)**R2)
        VPSTRAN_2ND_INV=VPSTRAN_2ND_INV+(R2*(EPSILON_V(M))**R2)
      END DO
      DOT_VPSTRAN_2ND_INV=P05*DOT_VPSTRAN_2ND_INV
      VPSTRAN_2ND_INV=P05*VPSTRAN_2ND_INV
      DOT_VPSTRAN_2ND_INV=SQRT(DOT_VPSTRAN_2ND_INV)
      VPSTRAN_2ND_INV=SQRT(VPSTRAN_2ND_INV)
      END IF
      END IF
*   ####################################################
*     END OF (VISCO)PLASTICITY ROUTINES
*   ####################################################
```

3.3.1 Compute strain rate invariants

```
      DOT_VPSTRAN_1ST_INV=R0
      DOT_VPSTRAN_2ND_INV=R0
      DO M=1,NDI
        DOT_VPSTRAN_2ND_INV=DOT_VPSTRAN_2ND_INV+((DEPSILON_V(M))/
     1           DTIME*(DEPSILON_V(M))/DTIME)+
     2          ((DEPSILON_VP(M))/DTIME*(DEPSILON_VP(M))/DTIME)+
     3           (DSTRAN(M)/DTIME*DSTRAN(M)/DTIME)
       DOT_VPSTRAN_1ST_INV=DOT_VPSTRAN_1ST_INV+(DEPSILON_V(M))
     1            /DTIME+(DEPSILON_VP(M))/DTIME+(DSTRAN(M))/DTIME
      END DO
      DO M=NDI+1,NTENS
       DOT_VPSTRAN_2ND_INV=DOT_VPSTRAN_2ND_INV+
     1          R2*(((DEPSILON_V(M))/DTIME)*((DEPSILON_V(M))/DTIME)+
     2             ((DEPSILON_VP(M))/DTIME*(DEPSILON_VP(M))/DTIME)+
     3             DSTRAN(M)/DTIME*DSTRAN(M)/DTIME)
      END DO
      DOT_VPSTRAN_2ND_INV=P05*DOT_VPSTRAN_2ND_INV
      DOT_VPSTRAN_2ND_INV=SQRT(DOT_VPSTRAN_2ND_INV)
*    ##################################
      VPSTRAN_1ST_INV=R0
      VPSTRAN_2ND_INV=R0
      DO M=1,NTENS
       VPSTRAN_1ST_INV=VPSTRAN_1ST_INV+EPSILON_VP(M)*SOID(M)
      END DO
      DO M=1,NDI
        VPSTRAN_2ND_INV=VPSTRAN_2ND_INV+(EPSILON_VP(M))**R2
      END DO
      DO M=NDI+1,NTENS
       VPSTRAN_2ND_INV=VPSTRAN_2ND_INV+(R2*(EPSILON_VP(M))**R2)
      END DO
      VPSTRAN_2ND_INV=SQRT(P05*VPSTRAN_2ND_INV)

      PRESS=-R1*(STRESS(1)+STRESS(2)+STRESS(3))/R3
      PRESS=PRESS/1E9
      T_SOLIDUS=A11+(A22*PRESS)+(A33*PRESS**2)
*      FROM KATZ, SPIEGELMAN & LANGMUIR (2003), A NEW PARAMETERIZATION FOR HYDROUS MANTLE MELTING (GCUBED)
      T_LIQUIDUS=B11+(B22*PRESS)+(B33*PRESS**2)
*     FROM KATZ, SPIEGELMAN & LANGMUIR (2003), A NEW PARAMETERIZATION FOR HYDROUS MANTLE MELTING (GCUBED)
      T_SOLIDUS=T_SOLIDUS+273.15D0
      T_LIQUIDUS=T_LIQUIDUS+273.15D0
*       COMPUTE FRACTION OF MELT AT TIME STEP
       IF(T_SOLIDUS.LT.TEMP.AND.TEMP.LT.T_LIQUIDUS)THEN
          FPT=((TEMP-T_SOLIDUS)/(T_LIQUIDUS-T_SOLIDUS))**PWRFPT
       ELSE
          FPT=R0
       END IF
*      UPDATE STATE VARIABLES
      MAXSTRESS=MAX(STRESS(1), STRESS(2), STRESS(3))
      MINSTRESS=MIN(STRESS(1), STRESS(2), STRESS(3))
      DIFF_STRESS=MAXSTRESS-MINSTRESS
```

3.4 Update and store variables that would be called at the next time step

```
      DO I=1,NTENS
        STATEV(I)=EPSILON_VP(I)
*      SDV 1-4   ! VISCOPLASTIC STRAINS (11,22,33,12)
       END DO
       STATEV(NTENS+1)=GAMMA_BAR
*      SDV 5     ! HARDENING HISTORY
       STATEV(NTENS+2)=PHI_YDP
*      SDV 6     ! DRUCKER-PRAGER YIELD CRITERION
       STATEV(NTENS+3)=SQRTJ2
*      SDV 7     ! ELASTIC J2
       STATEV(NTENS+4)=SQRTJ2T
*      SDV 8     ! VISCOPLASTIC J2
       STATEV(NTENS+5)=PRESS*1E9
*      SDV 9     ! PRESSURE
       STATEV(NTENS+6)=DGAMA_VP
*      SDV 10    ! VISCOPLASTIC SMOOTH CONE MULTIPLIER
       STATEV(NTENS+7)=DOT_VPSTRAN_1ST_INV
*      SDV 11    ! FIRST INVARIANT OF VISCOPLASTIC STRAIN RATE
       STATEV(NTENS+8)=LOG10(DOT_VPSTRAN_2ND_INV)
*      SDV 12    ! SECOND INVARIANT OF VISCOPLASTIC STRAIN RATE
       STATEV(NTENS+9)=VPSTRAN_1ST_INV
*      SDV 13    ! FIRST INVARIANT OF VISCOPLASTIC STRAIN
       STATEV(NTENS+10)=VPSTRAN_2ND_INV
*      SDV 14    ! SECOND INVARIANT OF VISCOPLASTIC STRAIN
       STATEV(NTENS+11)=DIFF_STRESS
*      SDV 15    ! DIFFERENTIAL STRESS
*      THIS IS A PLACEHOLDER TO STORE THE EFFECTIVE VISCOSITY IN OTHER VERSIONS OF THE CODE, I
       .E., WE CAN REPLACE THIS FIELD WITH THAT VARIABLE
       STATEV(NTENS+12)=RPL
*      SDV 16    ! HEAT GENERATION PER UNIT TIME
       STATEV(NTENS+13)=DTEMP/DTIME
*      SDV 17    ! CHECK FOR THERMAL STEADY STATE
       STATEV(NTENS+15)=TEMP-TEMP_INIT
*      SDV 19    ! THERMAL PERTURBATIONS
       STATEV(NTENS+16)=DEPSILON_V(1)
*      SDV 20    ! VISCOUS STRAIN INCREMENT(11)
       STATEV(NTENS+17)=DEPSILON_V(2)
*      SDV 21    ! VISCOUS STRAIN INCREMENT(22)
       STATEV(NTENS+18)=DEPSILON_V(3)
*      SDV 22    ! VISCOUS STRAIN INCREMENT(33)
       STATEV(NTENS+19)=DEPSILON_V(4)
*      SDV 23    ! VISCOUS STRAIN INCREMENT(12)
       STATEV(NTENS+20)=DEPSILON_VP(1)
*      SDV 24    ! VISCOPLASTIC STRAIN INCREMENT(11)
       STATEV(NTENS+21)=DEPSILON_VP(2)
*      SDV 25    ! VISCOPLASTIC STRAIN INCREMENT(22)
       STATEV(NTENS+22)=DEPSILON_VP(3)
*      SDV 26    ! VISCOPLASTIC STRAIN INCREMENT(33)
       STATEV(NTENS+23)=DEPSILON_VP(4)
*      SDV 27    ! VISCOPLASTIC STRAIN INCREMENT(12)
       STATEV(NTENS+24)=FPT
*      SDV 28    ! FRACTION OF MELT
       RETURN
       END
*     #################################
*      END OF USER MATERIAL SUBROUTINE
*     #################################
```

```
*   ###################################
```

4 UMATH

We include the UMAT subroutine summarized above.

In the User Material Heat Transfer (UMATHT) subroutine, we compute the heat flux vector, internal energy, variation of internal energy with respect to temperature, and variation of heat flux vector with respect to spatial gradients of temperature. Based on these, Abaqus solves the energy conservation problem. The input from the mechanical deformations is the internal heat production. In principle, various sources of heating can be considered. In our case, we consider shear or shear and volumetric heating.

```
*###########################################
     SUBROUTINE UMATHT(U,DUDT,DUDG,FLUX,DFDT,DFDG,
     1 STATEV,TEMP,DTEMP,DTEMDX,TIME,DTIME,PREDEF,DPRED,
     2 CMNAME,NTGRD,NSTATV,PROPS,NPROPS,COORDS,PNEWDT,
     3 NOEL,NPT,LAYER,KSPT,KSTEP,KINC)
*
      INCLUDE 'ABA_PARAM.INC'
      COMMON /CONDUCTIVITY_PARAMS/ TREF, ALPH_LE, RHOD_REF, TEMPERA, PRESS
      CHARACTER*80 CMNAME

       DIMENSION DUDG(NTGRD),FLUX(NTGRD),DFDT(NTGRD),
     1 DFDG(NTGRD,NTGRD),STATEV(NSTATV),DTEMDX(NTGRD),
     2 TIME(2),PREDEF(1),DPRED(1),PROPS(NPROPS),COORDS(3)

       PARAMETER (EXMULT=0.0000004D0)
*        IF(CMNAME.EQ.'CRUST')THEN
*           COND=1.18D0+474.0D0/(TEMP+77.0D0)
*        ELSEIF(CMNAME.EQ.'LITHOSPHERE'.OR.CMNAME.EQ.'WEAKSHEARZONE')THEN
*           COND=0.73D0+1293.0D0/(TEMP+77.0D0)
*       ELSEIF(CMNAME.EQ.'Matrix'.OR.CMNAME.EQ.'Inclusion')THEN
*           COND=1.72D0+807.0D0/(TEMP+350.0D0)
*        ELSE
*           COND=0.73D0+1293.0D0/(TEMP+77.0D0)
*        END IF
       COND = PROPS(1)
       C_P = PROPS(2)
       RHOD = PROPS(3)
       DUDT = C_P
       DU = C_P*DTEMP
       U = U+DU
       DO I=1,NTGRD
         FLUX(I) = -COND*DTEMDX(I)
         DFDG(I,I) = -COND
       END DO
      RETURN
      END
*###########################################
```

5 Postscript

The code has been developed for 2-D, 3-D, and axisymmetric problems. The number of tensor (NTENS) components determines the switch to a given class of problems. We tested our code on 2-D problems and axisymmetric problems. While the code has not been used on 3-D problems yet, the extension is to set up a

3-D geometry which increases the number of tensor components and the places where the history-dependent variables are stored are changed.