MgSpall2 / umat.f

zubaexy 3 minutes ago MgUMAT as obtained from Jeff on 22 June 2015

**0** contributors

3045 lines (2538 sloc) 100.754 kB

Raw Blame History

```
c-------------------------------------------------------------------
c     This umat is called by ale3d and wraps around a regular
c     abaqus umat call. Main differences are cmname, and initialization
c     is handled differently.
c-------------------------------------------------------------------

      subroutine umat ( stress,  statev,  ddsdde,  sse,     spd,
     &                  scd,     rpl,     ddsddt,  drplde,  drpldt,
     &                  strain,  dstrain, time,    dtime,   temp,
     &                  dtemp,   predef,  dpred,   cmname,  ndi,
     &                  nshr,    ntens,   nstatv,  props,   nprops,
     &                  coords,  drot,    pnewdt,  celent,  dfgrd0,
     &                  dfgrd1,  noel,    npt,     layer,   kspt,
     &                  kstep,   kinc )

      implicit none

      ! Dimension variables passed into the UMAT sub (not all are used)
      integer ndi, nshr, ntens, nstatv, nprops, noel
      integer layer, kspt, kstep, kinc, npt
      double precision cmname, sse, celent, dtime, temp, dtemp, pnewdt
      double precision spd, scd, rpl, drpldt
      character*7 cmnameale ! Material name
      double precision coords(3), ddsdde(6,6), ddsddt(ntens)
      double precision dfgrd1(3,3), dfgrd0(3,3), dpred(1), drplde(ntens)
      double precision drot(3,3), dstrain(ntens), predef(1)
      double precision props(nprops), statev(nstatv), strain(ntens)
      double precision stress(ntens), time(2)

c     Helper variables
      integer orient, ngrains
      double precision seed
      integer nstatvpg, nstatvreg
      parameter (nstatvreg=25, nstatvpg=31)
      cmnameale = 'magnesm'

      if (nstatv.eq.nstatvreg) then
        ngrains = 1
      else
        ngrains = nstatv / nstatvpg
      end if

c     if first step perform initialization
c     statev 1 is random seed (0-1), 2 is orientation, 3 is num xtals
      if ((time(1)-dtime).le.(0.5d+0*dtime)) then
        seed = statev(1)
        orient = int(statev(2))
        call sdvinia(statev,nstatv,seed,orient,ngrains)
```

```fortran
      end if


      if (ngrains.eq.1) then
        call cumat(stress,  statev,  ddsdde,  sse,     spd,
     &    scd,     rpl,     ddsddt,  drplde,  drpldt,
     &    strain,  dstrain, time,    dtime,   temp,
     &    dtemp,   predef,  dpred,   cmnameale,  ndi,
     &    nshr,    ntens,   nstatv,  props,   nprops,
     &    coords,  drot,    pnewdt,  celent,  dfgrd0,
     &    dfgrd1,  noel,    npt,     layer,   kspt,
     &    kstep,   kinc)
      else
        call pumat(stress,  statev,  ddsdde,  sse,     spd,
     &    scd,     rpl,     ddsddt,  drplde,  drpldt,
     &    strain,  dstrain, time,    dtime,   temp,
     &    dtemp,   predef,  dpred,   cmnameale,  ndi,
     &    nshr,    ntens,   nstatv,  props,   nprops,
     &    coords,  drot,    pnewdt,  celent,  dfgrd0,
     &    dfgrd1,  noel,    npt,     layer,   kspt,
     &    kstep,   kinc)
      end if

      return
      end

c---------------------------------------------------------------------
c     ABAQUS STRESS - SIG11, SIG22, SIG33, SIG12, SIG13, SIG23
c     ABAQUS STRAIN - EPS11, EPS22, EPS33, GAM12, GAM13, GAM23
c                     WHERE GAM12 = 2*EPS12
c---------------------------------------------------------------------

      subroutine cumat ( stress,  statev,  ddsdde,  sse,     spd,
     &                  scd,     rpl,     ddsddt,  drplde,  drpldt,
     &                  strain,  dstrain, time,    dtime,   temp,
     &                  dtemp,   predef,  dpred,   cmname,  ndi,
     &                  nshr,    ntens,   nstatv,  props,   nprops,
     &                  coords,  drot,    pnewdt,  celent,  dfgrd0,
     &                  dfgrd1,  noel,    npt,     layer,   kspt,
     &                  kstep,   kinc )

      IMPLICIT NONE

      ! loop variables
      integer i

      ! Dimension variables passed into the UMAT sub (not all are used)
      integer ndi      ! Number of direct stress components
      integer nshr     ! Number of shear stress components
      integer ntens    ! Size of stess or stran array (ndi + nshr)
      integer nstatv   ! Number of SDVs
      integer nprops   ! Number of material constants
      integer noel     ! Element number
      integer layer    ! Layer number (for composites)
      integer kspt     ! Section point number within layer
      integer kstep    ! Step number
      integer kinc     ! Increment number
      integer npt      ! Integration point number
      character*7 cmname ! Material name
      double precision sse ! Specific elastic stain energy

      double precision
     & celent,         ! Characteristic element length
     & dtime,          ! Time increment
     & temp,           ! Temperature at start of increment
```

```
     & dtemp,          ! Temperature increment
     & pnewdt,         ! Ratio of new time increment to time
                       ! increment being used
     & spd,            ! Specific plastic dissipation
     & scd,            ! Specific creep dissipation
     & rpl,            ! Volumetic heat generation per unit time
     & drpldt,         ! Varation of rpl with temperature
     & coords(3),      ! Coordinates of Gauss pt. being evaluated
     & ddsdde(ntens,ntens), ! Tangent Stiffness Matrix
     & ddsddt(ntens),  ! Change in stress per change in temperature
     & dfgrd1(3,3),    ! Deformation gradient at end of step
     & dfgrd0(3,3),    ! Deformation gradient at beginning of step
     & dpred(1),       ! Change in predefined state variables
     & drplde(ntens),  ! Change in heat generation per change in strain
     & drot(3,3),      ! Rotation matrix
     & dstrain(ntens), ! Strain increment tensor stored in vector form
     & predef(1),      ! Predefined state vars dependent on field
                       ! variables
     & props(nprops),  ! Material properties passed in
     & statev(nstatv), ! State Variables
     & strain(ntens),  ! Strain tensor stored in vector form
     & stress(ntens),  ! Cauchy stress tensor stored in vector form
     & time(2)         ! Step Time and Total Time

      !Variables fed in from props
      double precision lam      ! Elastic lambda (MPa)
      double precision mu       ! Shear modulus (MPa)
      double precision grun     ! Grun coeff of MgO ~1.6 White JAP, 66
      double precision temp0    ! Initial temperature
      double precision tempmelt ! Melt temperature
      double precision rho0     ! Initial density (ng/um^3)
      double precision qbastw   ! Interaction param for bas and twin
      double precision qbassl   ! "" bas and slip
      double precision qtwbas   ! "" twin and bas
      double precision qtwsl    ! "" twin and slip
      double precision qslbas   ! "" slip and bas
      double precision qsltw    ! "" slip and twin
      integer hbastype          ! Basal slip hardening model used
      double precision hbas1    ! Basal slip hardening parameter1
      double precision hbas2    ! Basal slip hardening parameter2
      double precision hbas3    ! Basal slip hardening parameter3
      double precision hbas4    ! Basal slip hardening parameter4
      double precision hbas5    ! Basal slip hardening parameter5
      double precision hbas6    ! Basal slip hardening parameter6
      integer htwtype           ! Twinning hardening model used
      double precision htw1     ! Twinning hardening parameter 1
      double precision htw2     ! Twinning hardening paremeter 2
      double precision htw3     ! Twinning hardening paremeter 3
      double precision htw4     ! Twinning hardening paremeter 4
      double precision htw5     ! Twinning hardening paremeter 5
      double precision htw6     ! Twinning hardening paremeter 6
      integer hsltype           ! Slip hardening model used
      double precision hsl1     ! Slip hardening parameter 1
      double precision hsl2     ! Slip hardening parameter 2
      double precision hsl3     ! Slip hardening parameter 3
      double precision hsl4     ! Slip hardening parameter 4
      double precision hsl5     ! Slip hardening parameter 5
      double precision hsl6     ! Slip hardening parameter 6
      double precision hsl7     ! Slip hardening parameter 7
      double precision hsl8     ! Slip hardening parameter 8
      integer eosflag           ! 0 lin elast, 1 murn eos cv no art vis
c                                 2 murn eos cv wart visc
      double precision b0       ! Bulk modulus, Guinan and Stein, 74
      double precision dbdp     ! Bulk mod deriv w.r.t p, "" ref
      double precision cv       ! MPa/K, Lee, Int J Thermophys, 13
```

```fortran

      !Variables related to statev
      double precision re(3,3)     ! 3x3 rotation matrix
      double precision gambslip    ! Shear basal slip (notwin)
      double precision gamtw(6)    ! Shear strain on twin systems
      double precision epsl0       ! Nonbas effplastic strain prev step
      double precision epdsl       ! "      " rate
      double precision epdsl0      ! "      " from prev step step
      double precision energy      ! Int energy / ref vol
      double precision tempsv      ! Temperature stored as state var
      double precision depbas, deptw !change in epeff due to bas, tw

      !Utility
      integer nexit    ! Determines how calc_epdsl exited
c     0=reg, 1=noiter, 2=nobound, 3=maxit
      logical actbas, acttw, actsl !whether modes are active
      logical firstsl          !true if first sl iteration, f other
      double precision gbas, gsl   !trial shear on these
      double precision ysbas0, yssl0, yssl !initial and cur ys
      double precision dyssldepd       !change in ys wrt epd
      double precision depsl, dgambas, dgambslip !change in strains
      double precision p0, dp, p     !initial pressure, and increment
      double precision bmod       !bulk modulus
      double precision epdslmax   !estimate of max slip strain rate
      double precision epdslez    !epdsl calculation w/o bas or tw

      double precision
     & stw(3,6),         ! Slip dir vector for twin systems
     & mtw(3,6),         ! Slip norm vector for twin systems
     & mbas(3),          ! Slip norm vector for basal system
     & ptw(3,3,6),       ! P for twinning
     & phtw(3,3,6),      ! P hat for twinning
     & ystw0(6),         ! All 6 twin taus from last step
     & stressd0(3,3),    ! Deviatoric trial stress
     & stressd(3,3),     ! Deviatoric stress
     & sbas(3),          ! Slip dir vector for basal system
     & pbas(3,3),        ! P for basal slip
     & gtw(6),           ! Trial stress on twin systems
     & hmix0(6),         ! Interaction b/w twin and bas - doesnt change
     & dgamtw(6),        ! Delta gamma for twin in original order
     & dgamtwr(6),       ! Delta Gamma for twin in reduced order
     & sbslip(3),        ! Direction of bslip
     & wpdt(3,3),        ! Plastic spin times dt
     & wdt(3,3)          ! Spin times dt (from abaqus drot)

      integer
     & acttwsys(6)       ! List of active twin sys, 1 = act, 0 = inact

c     very solution oriented
      logical reactivate ! If true dont delete defm modes in slip step
      logical deactivatesl !If slip needs to be deactivated at end
      logical captw
      double precision reactTOL, deactTOL, actTOL
      integer itNum, itMax
      double precision twcap, gamtwtot

c-------------------------------------------------------------------
c     Solution related parameters
c-------------------------------------------------------------------
      actTOL = 1.0d-8
      deactTOL = 1.0d-4
      reactTOL = 1.0d-8
      itNum = 0
      itMax = 10
      reactivate = .false.
      deactivatesl = .false.
```

```
      firstsl = .true.

c--------------------------------------------------------------------
c     Read in material properties
c--------------------------------------------------------------------

      lam = props(1)
      mu = props(2)
      grun = props(3)
      temp0 = props(4)
      tempmelt = props(5)
      rho0 = props(6)
      qbastw = props(7)
      qbassl = props(8)
      qtwbas = props(9)
      qtwsl = props(10)
      qslbas = props(11)
      qsltw = props(12)
      hbastype = props(13)
      hbas1 = props(14)
      hbas2 = props(15)
      hbas3 = props(16)
      hbas4 = props(17)
      hbas5 = props(18)
      hbas6 = props(19)
      htwtype = props(20)
      htw1 = props(21)
      htw2 = props(22)
      htw3 = props(23)
      htw4 = props(24)
      htw5 = props(25)
      htw6 = props(26)
      hsltype = props(27)
      hsl1 = props(28)
      hsl2 = props(29)
      hsl3 = props(30)
      hsl4 = props(31)
      hsl5 = props(32)
      hsl6 = props(33)
      hsl7 = props(34)
      hsl8 = props(35)
      eosflag = props(36)
      b0 = props(37)
      dbdp = props(38)
      cv = props(39)

c--------------------------------------------------------------------
c     Read in state variables
c--------------------------------------------------------------------

      !Read in statev
      call init_statevs(statev, re, gambslip, gamtw, epsl0,
     &  epdsl0, energy, tempsv)
      temp = temp0
      gamtwtot = gamtw(1)+gamtw(2)+gamtw(3)+gamtw(4)+gamtw(5)+gamtw(6)

c--------------------------------------------------------------------
c     Store things before big loop
c--------------------------------------------------------------------

c     Store initial pressure
c     - could also solve hydrostatic part at beginning of time step
      p0 = - (stress(1) + stress(2) + stress(3))/3.0d+0

c     Based on initial re from the previous step,
```

```
c      calculate stw, mtw, and mbas, as well as ptw, and phat tw
      call ensurerot(re)
      call calc_stw_mtw_mbas(re, stw, mtw, mbas)
      call calc_ptw_phtw(stw, mtw, mbas, ptw, phtw)

c     Calculate trial stress
      call calc_trial_devstress(stress, dstrain, mu, stressd0)

c     Calculate sbas, and pbas based on trial stress
      call calc_spbas(stressd0, mbas, sbas, pbas)

c     Calculate trial stresses on each mode
      call calc_taus(stressd0, pbas, ptw, gbas, gtw, gsl)

c     Calculate strengths for basal, twin, and nb slip
      call calc_str_bas(gambslip, gamtw, epsl0, temp, hbastype, hbas1,
     &  hbas2, hbas3, hbas4, hbas5, hbas6, qbastw, qbassl, ysbas0)
      call calc_str_tw(gambslip, gamtw, epsl0, temp, htwtype,htw1,htw2,
     &  htw3, htw4, htw5, htw6, qtwbas, qtwsl, ystw0, twcap, captw)
      call calc_str_sl(.true., gambslip, gamtw, epsl0, epdsl0, temp,
     &  tempmelt, hsltype,hsl1,hsl2,hsl3,hsl4, hsl5, hsl6, hsl7, hsl8,
     &  qslbas, qsltw, yssl0, dyssldepd)

c===================================================================
c     Start the solve for increment in plastic strain of each mech
c===================================================================

c     Determine what deformation modes may potentially activate based
c     solely on the trial stress
      call calc_potactive_modes(gbas, gtw, gsl, ysbas0, ystw0, captw,
     &  yssl0, actTOL, mu, actbas, acttw, actsl)

c     Do necessary pre-calcs and initializations
      if (acttw) then
        call init_hmix(ptw, pbas, hmix0)
      end if
      depsl = 0.0d+0
      dgambas = 0.0d+0
      epdsl = 0.0d+0
      yssl = yssl0

c     -- Basal slip routine --
c      actbas = .false.  !Manually deactivate basal slip
      if (actbas) then
        call calc_dgambas(gbas, ysbas0, mu, depsl, yssl0, dgambas)
      end if

c     -- Twinning routine --
c      acttw = .false.   !Manually deactivate twinning
      if (acttw) then
        call calc_dgamtw(gtw, hmix0, ystw0, yssl0, mu, dgambas,
     &    depsl, deactTOL, acttwsys, dgamtwr)
      else
        do i=1,6
          dgamtwr(i) = 0.0d+0
          acttwsys(i) = 0
        end do
      end if

c     -- Non-basal slip routine --
c      actsl = .false.   !Manually deactivate nb slip
   22 if (actsl) then
c        Check if twinning or basal supressing nb slip - OBSOLETE
c        call suppress_slip_query(actbas, acttw, acttwsys, dgambas,
c     &    pbas, dgamtwr, phtw, stressd0, mu, yssl0, actsl)
```

```
        if (actsl) then
c         Only calculate epdslez once, on first sl iteration
          if (firstsl) then
c         Calculate epdsl as if it is the only mechanism is active
            call calc_max_epdsl(dstrain,gsl,lam,mu,dtime,hsl8,epdslmax)
            call calc_epdsl_ez(gambslip, gamtw, epsl0, gsl,
     &        hsltype, hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8,
     &        tempmelt,qslbas,qsltw, stressd0,mu,temp, dtime, epdslmax,
     &        epdslez)
            epdsl = epdslez
            firstsl = .false.
          else
            epdsl = epdslez
          end if

c         If basal slip or twin active, do full solve
          if (acttw.or.actbas) then
            epdslmax = epdsl
            call calc_epdsl(actbas, acttw, acttwsys, hmix0, yssl0, gbas,
     &        ysbas0, pbas,gambslip, gtw, ystw0, gamtw, phtw, epsl0,gsl,
     &        hsltype, hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8,
     &        tempmelt,qslbas,qsltw,stressd0,mu, temp, dtime, epdsl0,
     &        epdslmax, reactivate, epdsl, nexit)
          end if
c         if nexit = 2, slip stop by twin/bas, nexit=3 did not converge
        else
          epdsl = 0.0d+0
        end if
        depsl = epdsl * dtime

        call calc_str_sl(.true., gambslip, gamtw, epsl0, epdsl, temp,
     &    tempmelt, hsltype,hsl1,hsl2,hsl3,hsl4, hsl5, hsl6, hsl7, hsl8,
     &    qslbas, qsltw, yssl, dyssldepd)
        if (epdsl.lt.hsl8) then
          epdsl = 0.0d+0
          actsl = .false.
        end if
      end if

c     Slip is converged. Recalculate basal and twin if slip occurred.
c     -- Basal slip routine --
   23 if (actbas) then
        if ((actsl).or.(deactivatesl)) then
          call calc_dgambas(gbas, ysbas0, mu, depsl, yssl, dgambas)
        end if
      else
        dgambas = 0.0d+0
      end if

c     -- Twinning routine --
      if (acttw) then
        if ((actsl).or.(deactivatesl)) then
          call calc_dgamtw(gtw, hmix0, ystw0, yssl, mu, dgambas,
     &      depsl, deactTOL, acttwsys, dgamtwr)
        end if
      else
        do i=1,6
          dgamtwr(i) = 0.0d+0
          acttwsys(i) = 0
        end do
      end if

c     At the end: calculate the deviatoric part of the stress
      call calc_return_dev_stress(stressd0, acttwsys, mu, dgamtwr,
     & dgambas, depsl, yssl, phtw, pbas, stressd)
```

```
c     Based on new stress, check if mechanisms should be reactivated
      call reactivate_mechanisms(actbas, acttw, actsl, acttwsys, captw,
     &  stressd, pbas, ptw, ysbas0, ystw0, yssl, epdsl,
     &  reactTOL, deactTOL, mu, reactivate, deactivatesl)

      itNum = itNum + 1

      !slip needs to be deactivated, dont change anything else
      if ((deactivatesl).and.(itNum.le.itMax).and.(reactivate)) go to 23
      !twinning or basal slip needs to be reactivated
      if ((reactivate).and.(itNum.le.itMax)) go to 22

c==================================================================
c     End of large solve, solution has converged, calc state vars
c==================================================================

c     Calculate dgambslip, sbslip
      call calc_dgamsbslip(dgambas, dgamtwr, sbas, mbas, ptw, phtw,
     &  dgambslip, sbslip)

c     Update re by calculating plastic spin
      call calc_wpdt(sbslip, mbas, dgambslip, wpdt)
      call calc_wdt_abq(drot, wdt)
      call update_re(actbas, wpdt, wdt, re)

c     Transform dgamtw from the reduced to full frame
      call calc_full_dgamtw(acttwsys, dgamtwr, dgamtw)
      call update_dep(actbas, acttw, dgambslip, sbslip, mbas, dgamtw,
     &  ptw, depbas, deptw)

c     Calculate the pressure contribution and return it to the stress
      if (eosflag.eq.0) then
        bmod = lam+2.0d+0*mu/3.0d+0
        dp = - bmod*(dstrain(1) + dstrain(2) + dstrain(3))
        p = p0+dp
        tempsv = temp0
      else
        p = p0
        call eos(eosflag, dfgrd0, dfgrd1, dtime, temp0, rho0, grun, b0,
     &   dbdp, cv, ysbas0, ystw0, yssl, dgambslip, dgamtw, depsl,
     &   p, energy, tempsv)
      end if

      stress(1) = stressd(1,1) - p
      stress(2) = stressd(2,2) - p
      stress(3) = stressd(3,3) - p
      stress(4) = stressd(1,2)
      stress(5) = stressd(1,3)
      stress(6) = stressd(2,3)

      call update_statevs(re, dgambslip, dgamtw, depsl, epdsl,
     &  energy, tempsv, depbas, deptw, dtime, statev)
c     Return ddsdde for abaqus implicit
      call elastddsdde(lam, mu, ntens, ndi, ddsdde)

      return
      end

c==================================================================
c==================================================================
c  Construct state variable array from other variables
c  -- NOTE: epdeff is approx as epeff/(dtime+TOL) so no NaN if dtime=0
c------------------------------------------------------------------
      subroutine update_statevs(re, dgambslip, dgamtw, depsl, epdsl,
     &  energy, tempsv, depbas, deptw, dtime, statev)
```

```

      implicit none

c     input
      double precision re(3,3), dgambslip, dgamtw(6), depsl, epdsl
      double precision energy, tempsv, depbas, deptw, dtime

c     output
      double precision statev(25)

c     util
      double precision GAMTW, TOL
      parameter (GAMTW=0.128917d+0, TOL=1.0d-12)

      !Update state variables
      statev(1) = re(1,1)
      statev(2) = re(1,2)
      statev(3) = re(1,3)
      statev(4) = re(2,1)
      statev(5) = re(2,2)
      statev(6) = re(2,3)
      statev(7) = re(3,1)
      statev(8) = re(3,2)
      statev(9) = re(3,3)
      statev(10) = statev(10) + dgambslip
      statev(11) = statev(11) + dgamtw(1)
      statev(12) = statev(12) + dgamtw(2)
      statev(13) = statev(13) + dgamtw(3)
      statev(14) = statev(14) + dgamtw(4)
      statev(15) = statev(15) + dgamtw(5)
      statev(16) = statev(16) + dgamtw(6)
      statev(17) = statev(17) + depsl
      statev(18) = epdsl
      statev(19) = energy
      statev(20) = tempsv
      statev(21) = statev(21) + depbas
      statev(22) = statev(22) + deptw
      statev(23) = (statev(11) + statev(12) + statev(13) + statev(14) +
     &  statev(15) + statev(16))/GAMTW !vf twin
      statev(24) = statev(24) + (depbas+deptw+depsl)
      statev(25) = (depbas+deptw+depsl) / (dtime+TOL)
      return
      end

c===================================================================
c===================================================================
c  Read in state variables array to other variables
c-------------------------------------------------------------------

      subroutine init_statevs(statev, re, gambslip, gamtw, epsl0,
     &  epdsl0, energy, tempsv)
      implicit none

c     input
      double precision statev(25)

c     output
      double precision re(3,3), gambslip, gamtw(6), epsl0, epdsl0
      double precision energy, tempsv

      !Read in statev
      re(1,1) = statev(1)
      re(1,2) = statev(2)
      re(1,3) = statev(3)
      re(2,1) = statev(4)
      re(2,2) = statev(5)
```

```fortran

      re(2,3) = statev(6)
      re(3,1) = statev(7)
      re(3,2) = statev(8)
      re(3,3) = statev(9)
      gambslip = statev(10)
      gamtw(1) = statev(11)
      gamtw(2) = statev(12)
      gamtw(3) = statev(13)
      gamtw(4) = statev(14)
      gamtw(5) = statev(15)
      gamtw(6) = statev(16)
      epsl0 = statev(17)
      epdsl0 = statev(18)
      energy = statev(19)
      tempsv = statev(20)
      !svs 21-25 do not need to be read in, they are just for output

      return
      end

c=====================================================================
c=====================================================================
c  Calculate depbas, deptw.

c---------------------------------------------------------------------

      subroutine update_dep(actbas, acttw, dgambslip, sbslip, mbas,
     &  dgamtw, ptw, depbas, deptw)
      implicit none

c     input
      logical actbas, acttw
      double precision dgambslip, sbslip(3), mbas(3)
      double precision dgamtw(6), ptw(3,3,6)

c     output
      double precision depbas, deptw

c     util
      double precision dbas(6), dtw(6)

c     depeff = dsqrt(2/3*dp:dp)*dt = dsqrt(2/3*deltadp:deltadp)

c     calculate deltaDP for basal slip
      if (actbas) then
        dbas(1)=dgambslip*sbslip(1)*mbas(1)
        dbas(2)=dgambslip*sbslip(2)*mbas(2)
        dbas(3)=dgambslip*sbslip(3)*mbas(3)
        dbas(4)=0.5d+0*dgambslip*(sbslip(1)*mbas(2)+sbslip(2)*mbas(1))
        dbas(5)=0.5d+0*dgambslip*(sbslip(1)*mbas(3)+sbslip(3)*mbas(1))
        dbas(6)=0.5d+0*dgambslip*(sbslip(2)*mbas(3)+sbslip(3)*mbas(2))
        depbas = dsqrt(2.0d+0/3.0d+0*(dbas(1)**2+dbas(2)**2+
     &    dbas(3)**2+2.0d+0*(dbas(4)**2+dbas(5)**2+dbas(6)**2)))
      else
        depbas = 0.0d+0
      end if

c     calculate deltaDP for twinning
      if (acttw) then
        dtw(1)=dgamtw(1)*ptw(1,1,1)+dgamtw(2)*ptw(1,1,2)+
     &    dgamtw(3)*ptw(1,1,3)+dgamtw(4)*ptw(1,1,4) +
     &    dgamtw(5)*ptw(1,1,5)+dgamtw(6)*ptw(1,1,6)
        dtw(2)=dgamtw(1)*ptw(2,2,1)+dgamtw(2)*ptw(2,2,2)+
     &    dgamtw(3)*ptw(2,2,3)+dgamtw(4)*ptw(2,2,4) +
     &    dgamtw(5)*ptw(2,2,5)+dgamtw(6)*ptw(2,2,6)
```

```
        dtw(3)=dgamtw(1)*ptw(3,3,1)+dgamtw(2)*ptw(3,3,2)+
     &    dgamtw(3)*ptw(3,3,3)+dgamtw(4)*ptw(3,3,4) +
     &    dgamtw(5)*ptw(3,3,5)+dgamtw(6)*ptw(3,3,6)
        dtw(4)=dgamtw(1)*ptw(1,2,1)+dgamtw(2)*ptw(1,2,2)+
     &    dgamtw(3)*ptw(1,2,3)+dgamtw(4)*ptw(1,2,4) +
     &    dgamtw(5)*ptw(1,2,5)+dgamtw(6)*ptw(1,2,6)
        dtw(5)=dgamtw(1)*ptw(1,3,1)+dgamtw(2)*ptw(1,3,2)+
     &    dgamtw(3)*ptw(1,3,3)+dgamtw(4)*ptw(1,3,4) +
     &    dgamtw(5)*ptw(1,3,5)+dgamtw(6)*ptw(1,3,6)
        dtw(6)=dgamtw(1)*ptw(2,3,1)+dgamtw(2)*ptw(2,3,2)+
     &    dgamtw(3)*ptw(2,3,3)+dgamtw(4)*ptw(2,3,4) +
     &    dgamtw(5)*ptw(2,3,5)+dgamtw(6)*ptw(2,3,6)
        deptw = dsqrt(2.0d+0/3.0d+0*(dtw(1)**2+dtw(2)**2+
     &    dtw(3)**2+2.0d+0*(dtw(4)**2+dtw(5)**2+dtw(6)**2)))
      else
        deptw = 0.0d+0
      end if

      return
      end


c=====================================================================
c=====================================================================
c  Calculate energy and pressure based volumetric strain
c---------------------------------------------------------------------

      subroutine eos(eosflag, dfgrd0, dfgrd1, dtime, temp0, rho0,
     &   gam, b0, dbdp, cv, ysbas0, ystw0, yssl, dgambslip, dgamtw,
     &   depsl, p, uint, tempsv)
      implicit none

c     input
      integer eosflag
      double precision dfgrd0(3,3), dfgrd1(3,3), dtime, temp0, rho0
      double precision gam, b0, dbdp, cv
      double precision ysbas0, ystw0(6), yssl
      double precision dgambslip, dgamtw(6), depsl

c     input/output
      double precision p, uint

c     output
      double precision tempsv

c     util
      double precision jnew, jold, du, rho, cb, q, p0
      double precision determinant
      double precision dtplast

      double precision qc1, qc2, ONE, TWO
      parameter (qc1=0.00d+0, qc2=0.0d+0, ONE=1.0D+0, TWO=2.0D+0)

c     determine jacobian, volume jump, density
      jnew = determinant(dfgrd1)
      jold = determinant(dfgrd0)
      du = (jnew-jold)/dtime
      rho = 2.0d0*rho0/(jnew+jold)
      p0 = p
      cb = dsqrt((b0+dbdp*p0)/rho) !Murnaghan eos

c     artificial viscosity - off in rarefaction
      if ((jnew.lt.jold).and.(eosflag.eq.2)) then
        q = rho*(qc1*cB*dabs(du)+qc2**2*du**2)
      else
```

```
          q = 0.0d+0
        end if

c       Murnaghan eos with constant cv
        p = (TWO*((b0*(dbdp + (jnew**dbdp - ONE)*(gam*jnew + ONE) -
     &  dbdp*jnew**dbdp*ONE*(gam*(jnew - ONE) + ONE)))/(dbdp*
     &  jnew**(dbdp*ONE)*(dbdp - ONE)) + (gam*((jold - jnew*ONE)*p0 -
     &  ((jnew - jold*ONE)*q + cv*gam*(jnew - ONE)*temp0)*TWO +
     &  TWO*uint))/TWO))/(gam*jnew - gam*jold*ONE + TWO)

c       Discretized energy update
        uint = uint - ONE/TWO*(jnew-jold)*(p+p0+q)

c       Alter pressure by art visc
        p = p + q

c       tempsv
        tempsv = (-(b0*jnew*ONE) + jnew**dbdp*(b0*(dbdp + jnew -
     &  dbdp*jnew*ONE) + dbdp*(dbdp - ONE)*(cv*(gam + ONE -
     &  gam*jnew*ONE)*temp0 + uint)))/(cv*dbdp*jnew**(dbdp*ONE)*
     &  (dbdp - ONE))
        dtplast = jnew/(rho0*cv)*(ysbas0*dgambslip+ystw0(1)*dgamtw(1)+
     &  ystw0(2)*dgamtw(2)+ystw0(3)*dgamtw(3)+ystw0(4)*dgamtw(4)+
     &  ystw0(5)*dgamtw(5)+ystw0(6)*dgamtw(6)+yssl*depsl)
        tempsv = tempsv + dtplast

        return
        end


c=====================================================================
c=====================================================================
c  Based on dgamtwr and acttwsys, assigns dgamtw to original systems
c---------------------------------------------------------------------

        subroutine calc_full_dgamtw(acttwsys, dgamtwr, dgamtw)
        implicit none

c       input
        integer acttwsys(6)
        double precision dgamtwr(6)

c       output
        double precision dgamtw(6)

c       util
        integer full, red, nact

        dgamtw(1) = 0.0d+0
        dgamtw(2) = 0.0d+0
        dgamtw(3) = 0.0d+0
        dgamtw(4) = 0.0d+0
        dgamtw(5) = 0.0d+0
        dgamtw(6) = 0.0d+0

        nact = acttwsys(1)+acttwsys(2)+acttwsys(3)+acttwsys(4)+
     &  acttwsys(5)+acttwsys(6)

        if (nact.gt.0) then
          full = 1 !full notation
          red = 1  !reduced notation
   10     if (red.le.nact) then
            if (acttwsys(full).eq.1) then
              dgamtw(full) = dgamtwr(red)
              red = red + 1
            else
```

```
            dgamtw(full) = 0.0d+0
          end if
          full = full + 1
          goto 10
        end if
      end if

      return
      end


c==================================================================
c==================================================================
c  Calculate Re = exp(wedt).Re, where in this case wedt=wpdt-wdt
c  -- note if basal is inactive, or if omega is small, do nothing to re
c------------------------------------------------------------------
      subroutine update_re(actbas, wpdt, wdt, re)
      implicit none

c     input
      logical actbas
      double precision wpdt(3,3), wdt(3,3)

c     intput/output
      double precision re(3,3)

c     util - w is used in place of wedt for shortness
      double precision w(3,3), wdw(3,3), ch(3,3), reo(3,3)
      double precision om, small

      small = 1.0d-12
      if (actbas) then

       w(1,1) = wdt(1,1) - wpdt(1,1)
       w(1,2) = wdt(1,2) - wpdt(1,2)
       w(1,3) = wdt(1,3) - wpdt(1,3)
       w(2,1) = wdt(2,1) - wpdt(2,1)
       w(2,2) = wdt(2,2) - wpdt(2,2)
       w(2,3) = wdt(2,3) - wpdt(2,3)
       w(3,1) = wdt(3,1) - wpdt(3,1)
       w(3,2) = wdt(3,2) - wpdt(3,2)
       w(3,3) = wdt(3,3) - wpdt(3,3)

c      omega = om = sqrt(0.5*w:w)
       om = dsqrt(0.5d+0*(w(1,1)*w(1,1)+w(1,2)*w(1,2) +
     &  w(1,3)*w(1,3)+w(2,1)*w(2,1)+w(2,2)*w(2,2) + w(2,3)*w(2,3) +
     &  w(3,1)*w(3,1)+w(3,2)*w(3,2)+w(3,3)*w(3,3)))
       if (om.ge.small) then
c       wdw = w.w
        wdw(1,1)=w(1,1)*w(1,1)+w(1,2)*w(2,1)+w(1,3)*w(3,1)
        wdw(1,2)=w(1,1)*w(1,2)+w(1,2)*w(2,2)+w(1,3)*w(3,2)
        wdw(1,3)=w(1,1)*w(1,3)+w(1,2)*w(2,3)+w(1,3)*w(3,3)
        wdw(2,1)=w(2,1)*w(1,1)+w(2,2)*w(2,1)+w(2,3)*w(3,1)
        wdw(2,2)=w(2,1)*w(1,2)+w(2,2)*w(2,2)+w(2,3)*w(3,2)
        wdw(2,3)=w(2,1)*w(1,3)+w(2,2)*w(2,3)+w(2,3)*w(3,3)
        wdw(3,1)=w(3,1)*w(1,1)+w(3,2)*w(2,1)+w(3,3)*w(3,1)
        wdw(3,2)=w(3,1)*w(1,2)+w(3,2)*w(2,2)+w(3,3)*w(3,2)
        wdw(3,3)=w(3,1)*w(1,3)+w(3,2)*w(2,3)+w(3,3)*w(3,3)

c       exp(w) = id + sin(om)/om*w+(1-cos(om))/om**2(w.w)
        ch(1,1)=1.0d+0+dsin(om)/om*w(1,1)+
     &   (1.0d+0-dcos(om))/(om*om)*wdw(1,1)
        ch(1,2)=dsin(om)/om*w(1,2)+(1.0d+0-dcos(om))/(om*om)*wdw(1,2)
        ch(1,3)=dsin(om)/om*w(1,3)+(1.0d+0-dcos(om))/(om*om)*wdw(1,3)
        ch(2,1)=dsin(om)/om*w(2,1)+(1.0d+0-dcos(om))/(om*om)*wdw(2,1)
```

```
        ch(2,2)=1.0d+0+dsin(om)/om*w(2,2)+
     &   (1.0d+0-dcos(om))/(om*om)*wdw(2,2)
        ch(2,3)=dsin(om)/om*w(2,3)+(1.0d+0-dcos(om))/(om*om)*wdw(2,3)
        ch(3,1)=dsin(om)/om*w(3,1)+(1.0d+0-dcos(om))/(om*om)*wdw(3,1)
        ch(3,2)=dsin(om)/om*w(3,2)+(1.0d+0-dcos(om))/(om*om)*wdw(3,2)
        ch(3,3)=1.0d+0+dsin(om)/om*w(3,3)+
     &   (1.0d+0-dcos(om))/(om*om)*wdw(3,3)

        reo(1,1) = re(1,1)
        reo(1,2) = re(1,2)
        reo(1,3) = re(1,3)
        reo(2,1) = re(2,1)
        reo(2,2) = re(2,2)
        reo(2,3) = re(2,3)
        reo(3,1) = re(3,1)
        reo(3,2) = re(3,2)
        reo(3,3) = re(3,3)

c       re = ch.reo
        re(1,1)=ch(1,1)*reo(1,1)+ch(1,2)*reo(2,1)+ch(1,3)*reo(3,1)
        re(1,2)=ch(1,1)*reo(1,2)+ch(1,2)*reo(2,2)+ch(1,3)*reo(3,2)
        re(1,3)=ch(1,1)*reo(1,3)+ch(1,2)*reo(2,3)+ch(1,3)*reo(3,3)
        re(2,1)=ch(2,1)*reo(1,1)+ch(2,2)*reo(2,1)+ch(2,3)*reo(3,1)
        re(2,2)=ch(2,1)*reo(1,2)+ch(2,2)*reo(2,2)+ch(2,3)*reo(3,2)
        re(2,3)=ch(2,1)*reo(1,3)+ch(2,2)*reo(2,3)+ch(2,3)*reo(3,3)
        re(3,1)=ch(3,1)*reo(1,1)+ch(3,2)*reo(2,1)+ch(3,3)*reo(3,1)
        re(3,2)=ch(3,1)*reo(1,2)+ch(3,2)*reo(2,2)+ch(3,3)*reo(3,2)
        re(3,3)=ch(3,1)*reo(1,3)+ch(3,2)*reo(2,3)+ch(3,3)*reo(3,3)

        end if
      end if

      return
      end

c=====================================================================
c=====================================================================
c  Calculate W^P*dt from basal slip part of L
c  -- recall mbas = mbslip
c  -- neglects twinning since this is just plain wrong
c---------------------------------------------------------------------

      subroutine calc_wpdt(sbs, mbs, dgambslip, wpdt)
      implicit none

c     input
      double precision sbs(3), mbs(3), dgambslip
c     output
      double precision wpdt(3,3)

      wpdt(1,1) = 0.0d+0
      wpdt(2,2) = 0.0d+0
      wpdt(3,3) = 0.0d+0
      wpdt(1,2) = 0.5d+0*dgambslip*(sbs(1)*mbs(2)-sbs(2)*mbs(1))
      wpdt(2,1) = 0.5d+0*dgambslip*(sbs(2)*mbs(1)-sbs(1)*mbs(2))
      wpdt(1,3) = 0.5d+0*dgambslip*(sbs(1)*mbs(3)-sbs(3)*mbs(1))
      wpdt(3,1) = 0.5d+0*dgambslip*(sbs(3)*mbs(1)-sbs(1)*mbs(3))
      wpdt(2,3) = 0.5d+0*dgambslip*(sbs(2)*mbs(3)-sbs(3)*mbs(2))
      wpdt(3,2) = 0.5d+0*dgambslip*(sbs(3)*mbs(2)-sbs(2)*mbs(3))

      return
      end

c=====================================================================
c=====================================================================
```

```
c  Given drot in abaqus, calculate dW
c  -- FROM HUGHES AND WINGET,  W=2*(R-1)*(R+1)^(-1)
c-------------------------------------------------------------------

      SUBROUTINE calc_wdt_abq(DROT, W)
      IMPLICIT NONE

c     input
      DOUBLE PRECISION DROT(3,3)
c     output
      DOUBLE PRECISION W(3,3)
c     util
      DOUBLE PRECISION R1(3,3), R2(3,3)

      R1(1,1)=DROT(1,1)+1.0D+0
      R1(1,2)=DROT(1,2)
      R1(1,3)=DROT(1,3)
      R1(2,1)=DROT(2,1)
      R1(2,2)=DROT(2,2)+1.0D+0
      R1(2,3)=DROT(2,3)
      R1(3,1)=DROT(3,1)
      R1(3,2)=DROT(3,2)
      R1(3,3)=DROT(3,3)+1.0D+0
      CALL calc_inverse_3x3(R1,R2)
      R1(1,1)=DROT(1,1)-1.0D+0
      R1(1,2)=DROT(1,2)
      R1(1,3)=DROT(1,3)
      R1(2,1)=DROT(2,1)
      R1(2,2)=DROT(2,2)-1.0D+0
      R1(2,3)=DROT(2,3)
      R1(3,1)=DROT(3,1)
      R1(3,2)=DROT(3,2)
      R1(3,3)=DROT(3,3)-1.0D+0
C
      W(1,1)=R1(1,1)*R2(1,1)+R1(1,2)*R2(2,1)+R1(1,3)*R2(3,1)
      W(1,2)=R1(1,1)*R2(1,2)+R1(1,2)*R2(2,2)+R1(1,3)*R2(3,2)
      W(1,3)=R1(1,1)*R2(1,3)+R1(1,2)*R2(2,3)+R1(1,3)*R2(3,3)
      W(2,1)=R1(2,1)*R2(1,1)+R1(2,2)*R2(2,1)+R1(2,3)*R2(3,1)
      W(2,2)=R1(2,1)*R2(1,2)+R1(2,2)*R2(2,2)+R1(2,3)*R2(3,2)
      W(2,3)=R1(2,1)*R2(1,3)+R1(2,2)*R2(2,3)+R1(2,3)*R2(3,3)
      W(3,1)=R1(3,1)*R2(1,1)+R1(3,2)*R2(2,1)+R1(3,3)*R2(3,1)
      W(3,2)=R1(3,1)*R2(1,2)+R1(3,2)*R2(2,2)+R1(3,3)*R2(3,2)
      W(3,3)=R1(3,1)*R2(1,3)+R1(3,2)*R2(2,3)+R1(3,3)*R2(3,3)

      return
      end

c===================================================================
c===================================================================
c  Calculates inverse of a 3x3 matrix
c-------------------------------------------------------------------

      subroutine calc_inverse_3x3(a,b)
      ! Calculate the inverse of a 3 x 3 matrix.

      implicit none

      double precision a(3,3), b(3,3)
      double precision d, small
      integer i, j

      small = 1d-12


      b(1,1) = a(2,2) * a(3,3) - a(3,2) * a(2,3)
      b(1,2) = a(3,2) * a(1,3) - a(1,2) * a(3,3)
```

```
      b(1,3) = a(1,2) * a(2,3) - a(2,2) * a(1,3)
      b(2,1) = a(3,1) * a(2,3) - a(2,1) * a(3,3)
      b(2,2) = a(1,1) * a(3,3) - a(3,1) * a(1,3)
      b(2,3) = a(2,1) * a(1,3) - a(1,1) * a(2,3)
      b(3,1) = a(2,1) * a(3,2) - a(3,1) * a(2,2)
      b(3,2) = a(3,1) * a(1,2) - a(1,1) * a(3,2)
      b(3,3) = a(1,1) * a(2,2) - a(2,1) * a(1,2)

      d = a(1,1) * b(1,1) + a(1,2) * b(2,1) + a(1,3) * b(3,1)

      if (abs(d).le.small) then
        print*, 'Took determinant in inverse, smaller than 1e-12'
      end if

      DO i = 1,3
          DO j = 1,3
              b(i,j) = b(i,j) / d
          END DO
      END DO

      RETURN
      END

c=====================================================================
c=====================================================================
c  Find out if any mechanisms need to be reactivated. If so,
c    reactivate mechanisms according to which has the highest
c    overstress (tau - ys), and reactivate that one. Return
c    reactivate = .true. if any need to be, .false. otherwise
c---------------------------------------------------------------------

      subroutine reactivate_mechanisms(actbas, acttw, actsl, acttwsys,
     &  captw, sigd, pbas, ptw, ysbas, ystw, yssl, epdsl, reactTOL,
     &  deactTOL, mu, reactivate, deactivatesl)

      implicit none

c     input/output
      logical actbas, acttw, actsl
      integer acttwsys(6)
      logical captw

c     input
      double precision sigd(3,3), pbas(3,3), ptw(3,6,6)
      double precision ysbas, ystw(6), yssl, epdsl
      double precision reactTOL, deactTOL, mu

c     output
      logical reactivate, deactivatesl

c     util
      double precision taubas, tautw(6), sigvm, maxdiff
      double precision stressdiff
      integer i, maxdiffint

      call calc_taus(sigd, pbas, ptw, taubas, tautw, sigvm)

c     maxdiffint is 1-6 for tw, 7 for bas, 8 for sl
c     if the system would be active based on stress but is inactive,
c       record the difference, and if it's the largest diff, record
c       its maxdiffint. also, throw a warning if the system is active
c       but shouldn't be

c     twinning
      maxdiff = 0.0d+0
```

```
      maxdiffint = 0
      reactivate = .false.

      if (.not.captw) then
        do i=1,6
          stressdiff = tautw(i)-ystw(i)
c         reactivation
          if ((stressdiff/mu.ge.reactTOL).and.(acttwsys(i).eq.0)) then
            if (stressdiff.gt.maxdiff) then
              maxdiff = stressdiff
              maxdiffint = i
            end if
          end if
c         decactivation identification
          if ((stressdiff/mu.le.-deactTOL).and.(acttwsys(i).eq.1)) then
            print*, 'ERROR: TWIN SYSTEM ', i, ' IS ACTV BUT SHOULDNT BE'
            print*, '-- TAU: ', tautw(i), '  YS: ', ystw(i)
            print*, '-- ACTTWSYS, ', acttwsys
            print*, 'allTau: ', tautw
          end if
        end do
      end if

c     basal slip
      stressdiff = taubas - ysbas
c     reactivation
      if ((stressdiff/mu.ge.reactTOL).and.(.not.actbas)) then
        if (stressdiff.gt.maxdiff) then
          maxdiff = stressdiff
          maxdiffint = 7
        end if
      end if
c     decactivation identification
      if ((stressdiff/mu.le.-deactTOL).and.(actbas)) then
        print*, 'ERROR: BASAL SLIP IS ACTIVE BUT SHOULDNT BE'
        print*, '-- TAU: ', taubas, '  YS: ', ysbas
      end if

c     nonbasal slip
      stressdiff = sigvm - yssl
c     reactivation
      if ((stressdiff/mu.ge.reactTOL).and.(.not.actsl)) then
        if (stressdiff.gt.maxdiff) then
          maxdiff = stressdiff
          maxdiffint = 8
        end if
      end if
c     decactivation
      if ((stressdiff/mu.le.-deactTOL).and.(actsl).and.(epdsl.gt.1d-8))
     &  then
c        print*, 'ERROR: NONBASAL SLIP IS ACTIVE BUT SHOULDNT BE'
c        print*, '-- SIGVM: ', sigvm, '  YS: ', yssl
        actsl = .false.
        epdsl = 0.0d+0
        deactivatesl = .true.
        reactivate = .true.
      end if

c     reactivation
      if (maxdiffint.gt.0) then
        reactivate = .true.
        if (maxdiffint.le.6) then !twinning
          acttw = .true.
          acttwsys(maxdiffint) = 1
        elseif (maxdiffint.eq.7) then !basal
```

```
          actbas = .true.
        else !nonbasal
          reactivate = .false.
c          actsl = .true.
          print*,'WARNING: NONBASAL SLIP SHOULD BE REACTIVATED BUT WONT'
          print*,'sigvm: ', sigvm, '  ys: ', yssl
        end if
      end if

      return
      end

c=====================================================================
c=====================================================================
c  Calculate the amount of non-basal slip that would cause a mode
c    to become inactive. Note that full representation of gtw, ystw
c    are fed in, and not reduced form, but depftw is returned for
c    the reduced form.
c---------------------------------------------------------------------

      subroutine calc_epsflip(actbas, acttw, hmix0, gbas, ysbas, gtw,
     &     ystw, yssl, mu, acttwsys, epfbas, depftw)

c     input
      logical actbas, acttw
      double precision hmix0(6)
      double precision gbas, ysbas
      double precision gtw(6), ystw(6)
      double precision yssl
      double precision mu
      integer acttwsys(6)

c     output
      double precision epfbas, depftw(6)

c     util
      integer i,j,nact
      double precision num, denom
      double precision mhinv(6,6), hmix(6), gtwr(6), ystwr(6)

c     evaluate basal slip component
      if (actbas) then
        epfbas = yssl/(3.0d+0*mu*ysbas)*(gbas-ysbas)
      else
        epfbas = 1.0d+5
      end if

c     evaluate twining component - note different if basal is active
      if (acttw) then
c       define mhinv, gtwr, ystwr based on acttwsys
        call calc_reduced_hmix(hmix0, acttwsys, hmix)
        call calc_reduced_gys(gtw, ystw, acttwsys, gtwr, ystwr)
        nact = acttwsys(1) + acttwsys(2) + acttwsys(3) + acttwsys(4) +
     &    acttwsys(5) + acttwsys(6)

        if (actbas) then
c         define hmix based on acttwsys
          call calc_minv(acttwsys, mhinv)
          do i=1,nact
            num = 0.0d+0
            denom = 0.0d+0
            do j=1,nact
              num = num + mhinv(i,j)*(gtwr(j)-hmix(j)*gbas
     &         - (ystwr(j) - hmix(j)*ysbas))
              denom = denom+mhinv(i,j)*(ystwr(j)-hmix(j)*ysbas)
```

```
          end do
          depftw(i) = yssl/(3.0d+0*mu)*num/denom

c         note if depftwin is greater than epfbas, above is invalid
          if (depftw(i).ge.epfbas) then
            num = 0.0d+0
            denom = 0.0d+0
            do j=1,nact
              num = num + mhinv(i,j)*(gtwr(j)-ystwr(j))
              denom = denom + mhinv(i,j)*(ystwr(j))
            end do
            depftw(i) = yssl/(3.0d+0*mu)*num/denom
          end if
        end do

c     evaluate twinning w/o basal slip
      else
        do i=1,nact
          num = 0.0d+0
          denom = 0.0d+0
          do j=1,nact
            num = num + mhinv(i,j)*(gtwr(j)-ystwr(j))
            denom = denom + mhinv(i,j)*(ystwr(j))
          end do
          depftw(i) = yssl/(3.0d+0*mu)*num/denom
        end do
      end if

c     if twinning is inactive
      else
        do i=1,6
          depftw(i) = 1.0d+5
        end do
      end if

      return
      end

c==================================================================
c==================================================================
c  If stress is not enough to cause slip with epdsl = 0, kill slip. Do
c    so by changing actsl from .true. to .false.
c------------------------------------------------------------------

      subroutine suppress_slip_query(actbas, acttw, acttwsys, dgambas,
     &    pbas, dgamtw, phtw, sigdt, mu, yssl0, actsl)

      implicit none
c     input
      logical actbas, acttw
      integer acttwsys(6)
      double precision dgambas, pbas(3,3), dgamtw(6), phtw(3,3,6)
      double precision sigdt(3,3) !deviatoric trial stress
      double precision mu, yssl0

c     input/output
      logical actsl

c     util
      integer i,j,k,nact
      double precision sigd(3,3), phtwr(3,3,6), sigvm

c     if neither twinning or basal is active, skip this function
      if ((actbas).or.(acttw)) then
        nact = acttwsys(1) + acttwsys(2) + acttwsys(3) + acttwsys(4) +
```

```
     &    acttwsys(5) + acttwsys(6)
        call calc_reduced_phtw(phtw, acttwsys, phtwr)
        do i=1,3
          do j=1,3
            sigd(i,j) = sigdt(i,j) - 2.0d+0*mu*dgambas*pbas(i,j)
            do k=1,nact
              sigd(i,j) = sigd(i,j) - 2.0d+0*mu*dgamtw(k)*phtwr(i,j,k)
            end do
          end do
        end do

        sigvm = dsqrt(3.0d+0/2.0d+0*(sigd(1,1)**2+
     &    sigd(2,2)**2+sigd(3,3)**2+2.0d+0*(sigd(1,2)**2 +
     &    sigd(1,3)**2 + sigd(2,3)**2)))

c       if stress with epdsl = 0 implies no j2 slip occurs, kill slip
        if (sigvm.lt.yssl0) then
          actsl = .false.
        end if

      end if

      return
      end


c=====================================================================
c=====================================================================
c  Determines if basal slip or twinning should be deactivated based
c    on depslmax, which is the maximum slip that occurs if it is the
c    only deformation mechanism that is active
c---------------------------------------------------------------------

      subroutine deactivate_bastw_fromslip(actbas, acttw, acttwsys,
     &  hmix0, gbas, ysbas, gtw, ystw, yssl0, mu, depslmax)

      implicit none

c     input/output
      logical actbas, acttw
      integer acttwsys(6)

c     input
      double precision hmix0(6), gbas, ysbas, gtw(6), ystw(6), yssl0
      double precision mu
      double precision depslmax !max plast strain for slip only

c     util
      logical elimsys
      double precision depfbas, depftw(6), depfmin
      integer i,j,nact

c     see if anything needs to be eliminated by finding the smallest
c       ep that switches signs, and see if its less than depslmax
      elimsys = .true.
  21  if ((elimsys).and.(actbas.or.acttw)) then

c       calculate deps so that sign flips, twin is reduced form
        call calc_epsflip(actbas, acttw, hmix0, gbas, ysbas, gtw,
     &    ystw, yssl0, mu, acttwsys, depfbas, depftw)

c       common initializations
        depfmin = 1.0d+5
        nact = acttwsys(1) + acttwsys(2) + acttwsys(3) + acttwsys(4)
     &       + acttwsys(5) + acttwsys(6)
```

```
        if (nact.eq.0) then
          acttw = .false.
        end if

c       determine reduced index of minimum epf, 0 is for basal slip
        do i=1,nact
          if (depftw(i).le.depfmin) then
            j=i
            depfmin = depftw(i)
          end if
        end do
        if (depfbas.le.depfmin) then
          j=0
          depfmin = depfbas
        end if

c       if true, eliminate a system and start over
        if (depfmin.le.depslmax) then
          if (j.eq.0) then !basal slip
            actbas = .false.
          else !twinning
            call remove_acttwsys_entry(acttwsys, j)
            nact = acttwsys(1) + acttwsys(2) + acttwsys(3) + acttwsys(4)
     &       + acttwsys(5) + acttwsys(6)
c            if (nact.eq.0) then
c              print*, 'deleted all twin modes in slip step'
c            end if
          end if
        else
          elimsys = .false.
        end if
        go to 21
      end if

      return
      end


c=====================================================================
c=====================================================================
c  Calculate epd, von mises stress, and a logical of activity for slip
c---------------------------------------------------------------------

      subroutine calc_epdsl(actbas, acttw, acttwsys, hmix0, yssl0,
     &  gbas, ysbas, pbas, gambslip, gtw, ystw, gamtw, phtw, epsl0, gsl,
     &  hsltype, hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8,
     &  tempmelt, qslbas, qsltw, stressd0, mu, temp, dt, epdslprev,
     &  epdslmax, reactivate, epdsl, nexit)

      implicit none


c     intput/output
      logical actbas, acttw
      integer acttwsys(6)

c     input
c     - basal slip and twinning
      double precision hmix0(6), yssl0, gbas, ysbas, pbas(3,3), gambslip
      double precision gtw(6), ystw(6), gamtw(6)
      double precision phtw(3,3,6)
c     - slip - params and isvs
      double precision epsl0, gsl
      integer hsltype
      double precision hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8
```

```
      double precision qslbas, qsltw, tempmelt
c     - general
      double precision stressd0(3,3), mu, temp
c     - solution related
      double precision dt, epdslprev, epdslmax
      logical reactivate

c     output
      double precision epdsl
      integer nexit

c     util - solution related things
      logical bis
      double precision depslmax
      double precision FTOL, RTOL
      double precision X1, X2, F, FL, FH, DF, XL, XH, DX, DXOLD
      integer MAXIT, J
c     - initial parameters
      double precision a,b,c, tempvar
      double precision amat(3,3), bmat(3,3)

      DATA FTOL,RTOL/1.D-10,1.D-10/
      MAXIT = 100
      nexit = 0

c     changes actbas, acttw, and acttwsys based on if slip will cause
c       any of the deformation modes to deactivate
c       don't do this if reactivate is true
      depslmax = epdslmax*dt
      if (.not.reactivate) then
        call deactivate_bastw_fromslip(actbas, acttw, acttwsys,
     &    hmix0, gbas, ysbas, gtw, ystw, yssl0, mu, depslmax)
      end if

c     check if the step should be slip only, and if so, exit
      if ((.not.actbas).and.(.not.acttw)) then
        epdsl = epdslmax
        nexit = 0
        return
        end if

c     do initializations that will be used throughout
      call form_abc(stressd0, actbas, gbas, pbas, ysbas, acttw,
     &  acttwsys, gtw, ystw, hmix0, phtw, amat, bmat, a, b, c)

c     initial bisection check over strain rates of interest
      X1=hsl8
      X2=epdslmax

      bis=.true.
      call ksr(bis, X1, a, b, c, gsl, mu, dt,gambslip,gamtw,epsl0, temp,
     &  tempmelt, hsltype, hsl1,hsl2,hsl3,hsl4, hsl5, hsl6, hsl7, hsl8,
     &  qslbas, qsltw, FL, DF)

c     if FL is positive, no slip will happen
      if (FL.ge.0.0D+0) then
        epdsl = 0.0d+0
        return
      end if

      call ksr(bis, X2, a, b, c, gsl, mu, dt,gambslip,gamtw,epsl0, temp,
     &  tempmelt, hsltype, hsl1,hsl2,hsl3,hsl4, hsl5, hsl6, hsl7, hsl8,
     &  qslbas, qsltw, FH, DF)

      IF(DABS(FL) .LT. FTOL)THEN
        epdsl=X1
```

```
        nexit = 1
        RETURN
        END IF
      IF(DABS(FH) .LT. FTOL)THEN
        epdsl=X2
        nexit = 1
        RETURN
        END IF

C     SLIP WAS SUPPRESSED BY TWINNING AND BASAL SLIP
      IF((FL.GT.0.d0.AND.FH.GT.0.d0).OR.
     &  (FL.LT.0.0.AND.FH.LT.0.d0)) THEN
c        WRITE(6,19)X1,FL,X2,FH
c   19   FORMAT(' SOLUTION NOT BOUNDED',4G12.5)
c        print*, 'EPDSLPREV: ', epdslprev
c        print*, 'EPDSLMAX: ', epdslmax
c        print*, 'EPDSLMIN: ', hsl8
c        print*, 'acttw: ', acttw
c        print*, 'actbas: ', actbas
c        print*, 'A: ', a
c        print*, 'B: ', b
c        print*, 'C: ', c
        nexit = 2
        epdsl = 0.0d+0
        RETURN
        END IF

c     associate high and low of strain rate to high and low of function
      IF(FL .LT. 0.0d+0)THEN
        XL=X1
        XH=X2
      ELSE
        XH=X1
        XL=X2
      ENDIF

      epdsl = epdslprev
      DXOLD=DABS(X2-X1)
      DX=DXOLD
      bis = .false.

      call ksr(bis, epdsl,a,b,c,gsl,mu,dt,gambslip,gamtw,epsl0, temp,
     &  tempmelt, hsltype, hsl1,hsl2,hsl3,hsl4, hsl5, hsl6, hsl7, hsl8,
     &  qslbas, qsltw, F, DF)
C
C --- BISECT IF SOLUTION EXCEEDS LIMIT OR IF SLOW CONVERGENCE
C       OTHERWISE USE NEWTON ITERATION
C --- CONVERGENCE CHECKS ON BOTH STRAIN RATE AND NORMALIZED FUNCTION
C
      DO 10 J=1,MAXIT
C
        IF(((epdsl-XH)*DF-F)*((epdsl-XL)*DF-F) .GE. 0.0d+0 .OR.
     *    DABS(2.0d+0*F) .GT. DABS(DXOLD*DF) ) THEN
          DXOLD=DX
          DX=0.5d+0*(XH-XL)
          epdsl=XL+2.0d+0/3.0d+0*DX
          IF(DABS(XL-epdsl)*dt .LT. RTOL .AND.
     &       DABS(F) .LT. FTOL)RETURN
        ELSE
          DXOLD=DX
          DX=F/DF
          tempvar=epdsl
          epdsl=epdsl-DX
          IF(DABS(tempvar-epdsl)*dt .LT. RTOL .AND.
     &       DABS(F) .LT. FTOL)RETURN
        ENDIF
```

```
C
C ---   GET FUNCTION AND SLOPE FOR NEXT ITERATION
C
        call ksr(bis, epdsl,a,b,c,gsl,mu,dt,gambslip,gamtw,epsl0, temp,
     &   tempmelt, hsltype, hsl1,hsl2,hsl3,hsl4, hsl5, hsl6, hsl7, hsl8,
     &   qslbas, qsltw, F, DF)

        IF(DABS(DX) .LT. RTOL .AND. DABS(F) .LT. FTOL) RETURN
C
        IF(F .LT. 0.0d+0) THEN
          XL=epdsl
        ELSE
          XH=epdsl
        ENDIF
   10 CONTINUE
C
C --- CUT TIME STEP IF NO CONVERGENCE
C
c      NFAIL=.TRUE.
      print*, 'EDOT SOLUTION DID NOT CONVERGE IN 100 STEPS'
      nexit = 3

      return
      end

c==================================================================
c==================================================================
c  Form constants to be used in state and deriv equations for epsdsl
c------------------------------------------------------------------

      subroutine ksr(bis, edot, a, b, c, sigt, mu,  dt,
     &  gambslip, gamtw, epsl0, temp, tempmelt, hsltype,
     &  hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8,
     &  qslbas, qsltw, f, df)

      implicit none

c     input
      logical bis
      double precision edot
c     - parameters for loading and interaction
      double precision a,b,c, sigt, mu, dt
c     - used in strength call only
      double precision gambslip, gamtw, epsl0, temp, tempmelt
      integer hsltype
      dimension gamtw(6)
      double precision hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8
      double precision qslbas, qsltw

c     output
      double precision f, df

c     util - temporary var1, 2, von mises stress, dvm / dedot
      double precision d, dd, sigvm, dsded
      double precision epsl

      epsl = epsl0 + edot*dt


      call calc_str_sl(bis, gambslip, gamtw, epsl, edot, temp,
     & tempmelt,hsltype, hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8,
     & qslbas, qsltw, sigvm, dsded)

      d = mu*edot*dt
      dd = mu*dt
c     check - all units stress**2, then div by stress**2
```

```
      f = sigvm**2+6.0d+0*sigvm*d+9.0d+0*d**2-(a+b+c)
     & - 3.0d+0*d/sigvm*(b+2.0d+0*c)-9.0d+0*c*d**2/sigvm**2
      f = f / sigt**2

      if (bis) return

c     check - all units stress**2*time, then div by stress**2
      df = 2.0d+0*dsded*sigvm
     & + 6.0d+0*dd*(edot*dsded+sigvm)
     & + 18.0d+0*d*dd
     & - 3.0d+0*dd*(b+2.0d+0*c)*(sigvm-edot*dsded) / sigvm**2
     & - 18.0d+0*c*dd**2*(sigvm*edot-edot**2*dsded)/sigvm**3

      df = df / sigt**2

      return
      end

c=====================================================================
c=====================================================================
c  Calculate epd, von mises stress for nb slip in absence of
c  basal slip and twin increments (still uses init vals to calc sl str)
c---------------------------------------------------------------------

      subroutine calc_epdsl_ez(gambslip, gamtw, epsl0, gsl,
     &  hsltype, hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8,
     &  tempmelt, qslbas, qsltw, stressd0, mu, temp, dt, epdslmax,
     &  epdsl)

      implicit none

c     input
      double precision gambslip, gamtw(6)
      double precision epsl0, gsl
c     - slip params and thermodynamic vars
      integer hsltype
      double precision hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8
      double precision qslbas, qsltw, tempmelt
c     - general
      double precision stressd0(3,3), mu, temp
c     - solution related
      double precision dt, epdslmax

c     output
      double precision epdsl

c     util - solution related things
      logical bis
      double precision tempvar
      double precision FTOL, RTOL
      double precision X1, X2, F, FL, FH, DF, XL, XH, DX, DXOLD, FACT
      integer MAXIT, J

      DATA FTOL,RTOL/1.D-12,1.D-12/
      MAXIT = 100

c     initial bisection check over strain rates of interest
      X1=hsl8

      X2=epdslmax
      FACT=3.0d+0*mu*dt
      bis=.true.

      call ksr_ez(bis, X1, gsl, FACT, dt, gambslip, gamtw, epsl0, temp,
     &  tempmelt, hsltype, hsl1,hsl2,hsl3,hsl4, hsl5, hsl6, hsl7, hsl8,
     &  qslbas, qsltw, FL, DF)
```

```

c     if FL is positive, no slip will happen
      if (FL.ge.0.0D+0) then
        epdsl = 0.0d+0
        return
      end if

      call ksr_ez(bis, X2, gsl, FACT, dt, gambslip,gamtw,epsl0, temp,
     &  tempmelt, hsltype, hsl1,hsl2,hsl3,hsl4, hsl5, hsl6, hsl7, hsl8,
     &  qslbas, qsltw, FH, DF)

      IF(DABS(FL) .LT. FTOL)THEN
        epdsl=X1
        RETURN
        END IF
      IF(DABS(FH) .LT. FTOL)THEN
        epdsl=X2
        RETURN
        END IF
C
      IF((FL.GT.0.d0.AND.FH.GT.0.d0).OR.
     &  (FL.LT.0.0.AND.FH.LT.0.d0)) THEN
        WRITE(6,19)X1,FL,X2,FH
   19   FORMAT(' SOLUTION NOT BOUNDED IN EZ EVAL',4G12.5)
        epdsl = 0.0d+0
        RETURN
        END IF

c     associate high and low of strain rate to high and low of function
      IF(FL .LT. 0.0d+0)THEN
        XL=X1
        XH=X2
      ELSE
        XH=X1
        XL=X2
      ENDIF

      epdsl = 0.5d+0*(X1+X2)
      DXOLD=DABS(X2-X1)
      DX=DXOLD
      bis = .false.

      call ksr_ez(bis, epdsl, gsl, FACT, dt, gambslip,gamtw,epsl0, temp,
     &  tempmelt, hsltype, hsl1,hsl2,hsl3,hsl4, hsl5, hsl6, hsl7, hsl8,
     &  qslbas, qsltw, F, DF)
C
C --- BISECT IF SOLUTION EXCEEDS LIMIT OR IF SLOW CONVERGENCE
C       OTHERWISE USE NEWTON ITERATION
C --- CONVERGENCE CHECKS ON BOTH STRAIN RATE AND NORMALIZED FUNCTION
C
      DO 10 J=1,MAXIT
C
        IF(((epdsl-XH)*DF-F)*((epdsl-XL)*DF-F) .GE. 0.0d+0 .OR.
     *    DABS(2.0d+0*F) .GT. DABS(DXOLD*DF) ) THEN
          DXOLD=DX
          DX=0.5d+0*(XH-XL)
          epdsl=XL+2.0d+0/3.0d+0*DX
          IF(DABS(XL-epdsl)*dt .LT. RTOL .AND.

     &       DABS(F) .LT. FTOL)RETURN
        ELSE
          DXOLD=DX
          DX=F/DF
          tempvar=epdsl
          epdsl=epdsl-DX
          IF(DABS(tempvar-epdsl)*dt .LT. RTOL .AND.
```

```
     &       DABS(F) .LT. FTOL)RETURN
        ENDIF
C
C ---   GET FUNCTION AND SLOPE FOR NEXT ITERATION
C
        call ksr_ez(bis, epdsl, gsl, FACT, dt,gambslip,gamtw,epsl0,temp,
     &   tempmelt, hsltype, hsl1,hsl2,hsl3,hsl4, hsl5, hsl6, hsl7, hsl8,
     &   qslbas, qsltw, F, DF)

        IF(DABS(DX) .LT. RTOL .AND. DABS(F) .LT. FTOL) RETURN
C
        IF(F .LT. 0.0d+0) THEN
          XL=epdsl
        ELSE
          XH=epdsl
        ENDIF
   10 CONTINUE
C
C --- CUT TIME STEP IF NO CONVERGENCE
C
c      NFAIL=.TRUE.
      print*, 'EDOT SOLUTION DID NOT CONVERGE IN 100 STEPS IN EZ'

      return
      end

c=====================================================================
c=====================================================================
c  Form constants to be used in state and deriv equations for epsdsl
c---------------------------------------------------------------------

      subroutine ksr_ez(bis, edot, sigt, fact, dt,
     &  gambslip, gamtw, epsl0, temp, tempmelt, hsltype,
     &  hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8,
     &  qslbas, qsltw, f, df)

      implicit none

c     input
      logical bis
      double precision edot, sigt
c     - parameters for loading and interaction
      double precision fact, dt
c     - used in strength call only
      double precision gambslip, gamtw, epsl0, temp, tempmelt
      integer hsltype
      dimension gamtw(6)
      double precision hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8
      double precision qslbas, qsltw

c     output
      double precision f, df

c     util - temporary var1, 2, von mises stress, dvm / dedot
      double precision sigvm, dsded
      double precision epsl

      epsl = epsl0 + edot*dt


      call calc_str_sl(bis, gambslip, gamtw, epsl, edot, temp,
     & tempmelt,hsltype, hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8,
     & qslbas, qsltw, sigvm, dsded)

      f = (sigvm + fact*edot-sigt)/sigt

```

```
      if (bis) return

      df = (dsded + fact - sigt) / sigt

      return
      end

c=================================================================
c=================================================================
c  Form constants to be used in state and deriv equations for epsdsl
c-----------------------------------------------------------------

      subroutine form_abc(sigdt, actbas, gbas, pbas, ysbas, acttw,
     &  acttwsys, gtw, ystw, hmix0, phtw, amat, bmat, a, b, c)

      implicit none

c     input
      double precision sigdt(3,3), gbas, ysbas, pbas(3,3)
      double precision gtw(6), ystw(6), hmix0(6), phtw(3,3,6)
      logical actbas, acttw
      integer acttwsys(6)

c     output
      double precision amat(3,3), bmat(3,3)
      double precision a,b,c

c     util
      integer i,j,k,l,ntw
      double precision hmixr(6),mhinv(6,6),gtwr(6),ystwr(6),phtwr(3,3,6)

c     use reduced representation
      call calc_reduced_hmix(hmix0, acttwsys, hmixr)
      call calc_minv(acttwsys, mhinv)
      call calc_reduced_gys(gtw, ystw, acttwsys, gtwr, ystwr)
      call calc_reduced_phtw(phtw, acttwsys, phtwr)

      ntw = acttwsys(1) + acttwsys(2) + acttwsys(3) + acttwsys(4) +
     &  acttwsys(5) + acttwsys(6)

      do i=1,3
        do j=1,3
          amat(i,j) = sigdt(i,j)
          bmat(i,j) = 0.0d+0
c         if basal slip is active
          if (actbas) then
            amat(i,j) = amat(i,j) - 2.0d+0*gbas*pbas(i,j)
            bmat(i,j) = 2.0d+0*ysbas*pbas(i,j)
          end if
c         if basal slip and twinning are active
          if ((actbas).and.(acttw)) then
            do k=1,ntw
              do l=1,ntw
                amat(i,j) = amat(i,j)- phtwr(i,j,k)*mhinv(k,l)*(gtwr(l)
     &           - hmixr(l)*gbas)
                bmat(i,j) = bmat(i,j)+ phtwr(i,j,k)*mhinv(k,l)*(ystwr(l)
     &           - hmixr(l)*ysbas)
              end do

            end do
c         otherwise, if just twinning is active
          else if (acttw) then
            do k=1,ntw
              do l=1,ntw
                amat(i,j) = amat(i,j)- phtwr(i,j,k)*mhinv(k,l)*gtwr(l)
                bmat(i,j) = bmat(i,j)+ phtwr(i,j,k)*mhinv(k,l)*ystwr(l)
```

```
              end do
            end do
          end if
        end do
      end do

      a = amat(1,1)*amat(1,1)+amat(1,2)*amat(1,2)+amat(1,3)*amat(1,3)+
     &    amat(2,1)*amat(2,1)+amat(2,2)*amat(2,2)+amat(2,3)*amat(2,3)+
     &    amat(3,1)*amat(3,1)+amat(3,2)*amat(3,2)+amat(3,3)*amat(3,3)

      b = amat(1,1)*bmat(1,1)+amat(1,2)*bmat(1,2)+amat(1,3)*bmat(1,3)+
     &    amat(2,1)*bmat(2,1)+amat(2,2)*bmat(2,2)+amat(2,3)*bmat(2,3)+
     &    amat(3,1)*bmat(3,1)+amat(3,2)*bmat(3,2)+amat(3,3)*bmat(3,3)

      c = bmat(1,1)*bmat(1,1)+bmat(1,2)*bmat(1,2)+bmat(1,3)*bmat(1,3)+
     &    bmat(2,1)*bmat(2,1)+bmat(2,2)*bmat(2,2)+bmat(2,3)*bmat(2,3)+
     &    bmat(3,1)*bmat(3,1)+bmat(3,2)*bmat(3,2)+bmat(3,3)*bmat(3,3)

      a = 1.5d+0*a
      b = 3.0d+0*b
      c = 1.5d+0*c

      return
      end


c=====================================================================
c=====================================================================
c  Calculate conservative estimate of maximum von mises strain rate
c---------------------------------------------------------------------

      subroutine calc_max_epdsl(DSTRAN, SIGT, LAM, MU, DTIME,
     &  EPSDCUTOFF, EPSDMAX)

      implicit none

c     input
      double precision DSTRAN(6), SIGT, LAM, MU, DTIME, EPSDCUTOFF

c     output
      double precision EPSDMAX


c     util
      double precision DEPSD1, DEPSD2, DEPSD3, DEPSD4, DEPSD5, DEPSD6
      double precision DEPSH, DEPSE
      double precision EMOD

      EMOD = MU*(3.0D+0*LAM+2.0D+0*MU)/(LAM+MU)

      DEPSH=(DSTRAN(1)+DSTRAN(2)+DSTRAN(3))/3.0d+0
      DEPSD1=DSTRAN(1)-DEPSH
      DEPSD2=DSTRAN(2)-DEPSH
      DEPSD3=DSTRAN(3)-DEPSH
      DEPSD4=DSTRAN(4)/2.0d+0
      DEPSD5=DSTRAN(5)/2.0d+0
      DEPSD6=DSTRAN(6)/2.0d+0
      DEPSE=DEPSD1**2+DEPSD2**2+DEPSD3**2+2.0d+0*(DEPSD4**2+DEPSD5**2
     & + DEPSD6**2)
      DEPSE=DSQRT(2.0d+0/3.0d+0*DEPSE+EPSDCUTOFF)
      EPSDMAX=2.0d+0*(DEPSE+SIGT/EMOD)/DTIME

      return
      end

```

```
c=====================================================================
c=====================================================================
c  Calculate dgambas based
c---------------------------------------------------------------------

      subroutine calc_dgambas(gbas, ysbas, mu, depsl, yssl, dgambas)

      implicit none

c     input
      double precision gbas, ysbas, mu, depsl, yssl

c     output
      double precision dgambas

      dgambas = (gbas - ysbas*(1.d+0+3.0d+0*mu*depsl/yssl))/mu

      return
      end


c=====================================================================
c=====================================================================
c  Calculate dgambslip based on dgambas, dgamtw, sbas, mbas, ptw, phtw
c---------------------------------------------------------------------

      subroutine calc_dgamsbslip(dgambas, dgamtw, sbas, mbas, ptw,
     &  phtw, dgambslip, sbslip)

      implicit none

c     input
      double precision dgambas
      double precision dgamtw(6), sbas(3), mbas(3)
      double precision ptw(3,3,6), phtw(3,3,6)

c     output
      double precision dgambslip, sbslip(3)

c     util
      integer i

      double precision rvec(3)

      rvec(1) = 0.0d+0
      rvec(2) = 0.0d+0
      rvec(3) = 0.0d+0

      do i=1,6
        rvec(1) = rvec(1)+2.0d+0*dgamtw(i)*
     &    (mbas(1)*(ptw(1,1,i)-phtw(1,1,i))+
     &     mbas(2)*(ptw(1,2,i)-phtw(1,2,i))+
     &     mbas(3)*(ptw(1,3,i)-phtw(1,3,i)))
        rvec(2) = rvec(2)+2.0d+0*dgamtw(i)*
     &    (mbas(1)*(ptw(2,1,i)-phtw(2,1,i))+
     &     mbas(2)*(ptw(2,2,i)-phtw(2,2,i))+
     &     mbas(3)*(ptw(2,3,i)-phtw(2,3,i)))
        rvec(3) = rvec(3)+2.0d+0*dgamtw(i)*
     &    (mbas(1)*(ptw(3,1,i)-phtw(3,1,i))+
     &     mbas(2)*(ptw(3,2,i)-phtw(3,2,i))+
     &     mbas(3)*(ptw(3,3,i)-phtw(3,3,i)))
      end do

      dgambslip = dsqrt((dgambas*sbas(1)-rvec(1))**2 +
```

```
     &   (dgambas*sbas(2)-rvec(2))**2 + (dgambas*sbas(3)-rvec(3))**2)

c     calculate sbslip, with null value if basal slip doesnt occur
      if (dgambslip.le.1d-12) then
        sbslip(1) = 1.0d+0
        sbslip(2) = 1.0d+0
        sbslip(3) = 1.0d+0
      else
        sbslip(1) = (dgambas*sbas(1)-rvec(1))/dgambslip
        sbslip(2) = (dgambas*sbas(2)-rvec(2))/dgambslip
        sbslip(3) = (dgambas*sbas(3)-rvec(3))/dgambslip
      end if

      return
      end

c=====================================================================
c=====================================================================
c  Calculate deviatoric stress based on trial stress and amount
c    of plastic deformation
c  - This uses phattw, and pbas. Could also use ptwin and pbslip, just
c    have to make sure to use the correct variables
c---------------------------------------------------------------------

      subroutine calc_return_dev_stress(sigdT, acttwsys, mu, dgamtw,
     & dgambas, depsl, yssl, phtw, pbas, sigd)

      implicit none

c     input
      double precision sigdT(3,3), dgamtw(6), phtw(3,3,6), pbas(3,3)
      integer acttwsys(6)
      double precision mu, dgambas, depsl, yssl

c     output
      double precision sigd(3,3)

c     util
      integer i,j
      double precision phtwr(3,3,6)

      call calc_reduced_phtw(phtw, acttwsys, phtwr)

      do i=1,3
        do j=1,3
        sigd(i,j) = (sigdT(i,j)-2.0d0*mu*(phtwr(i,j,1)*dgamtw(1)
     &  +  phtwr(i,j,2)*dgamtw(2) + phtwr(i,j,3)*dgamtw(3)
     &  +  phtwr(i,j,4)*dgamtw(4) + phtwr(i,j,5)*dgamtw(5)
     &  +  phtwr(i,j,6)*dgamtw(6) + pbas(i,j)*dgambas))
     &  /  (1.0d+0+3.0d+0*mu*depsl/yssl)
        end do
      end do

      return
      end

c=====================================================================
c=====================================================================
c  Read in stress in indicial notation, output abaqus notation
c---------------------------------------------------------------------
      subroutine ind_to_voigtabq(sigi, sigabq)

      implicit none

c     input
      double precision sigi(3,3)
```

```
c     output
      double precision sigabq(6)

      sigabq(1) = sigi(1,1)
      sigabq(2) = sigi(2,2)
      sigabq(3) = sigi(3,3)
      sigabq(4) = sigi(1,2)
      sigabq(5) = sigi(1,3)
      sigabq(6) = sigi(2,3)

      return
      end


c=====================================================================
c=====================================================================
c  Calculate dgamtw and record active twin systems. dgamtw is stored
c   in reduced format
c---------------------------------------------------------------------

      subroutine calc_dgamtw(gtw, hmix0, ystw, yssl, mu, dgambas,
     &    depsl, deactTOL, acttwsys, dgamtw)

      implicit none

c     input
      double precision gtw(6), hmix0(6), ystw(6)
      double precision yssl, mu, dgambas, depsl, deactTOL

c     output
      integer acttwsys(6)
      double precision dgamtw(6)

c     util
      double precision minslip
      integer i,j, nact, minslipindex
      double precision mhinv(6,6), hmixr(6), gtwr(6), ystwr(6)
      logical negslip

c     test
      double precision gtwmod(6)

c     calc act tw sys based on trial stress less basal slip
      do i=1,6
        gtwmod(i) = gtw(i) - mu*dgambas*hmix0(i)
      end do
      call calc_acttwsys_stress(gtwmod, ystw, acttwsys)

      negslip = .true.
      nact = 0
      do i=1,6
        nact = nact + acttwsys(i)
      end do

   20 if ((negslip).and.(nact.gt.0)) then

c       zero everything out
        do i=1,6
          hmixr(i) = 0.0d+0
          dgamtw(i) = 0.0d+0
          do j=1,6
            mhinv(i,j) = 0.0d+0
          end do
        end do
```

```
        minslip = 0.0d+0

c       determine mhinv and hmix according to number of active systems
        call calc_minv(acttwsys, mhinv)
        call calc_reduced_hmix(hmix0, acttwsys, hmixr)
        call calc_reduced_gys(gtw, ystw, acttwsys, gtwr, ystwr)

c       sum over active systems
        do i=1,nact
          do j=1,nact
            dgamtw(i) = dgamtw(i) + mhinv(i,j)/(2.0d+0*mu)*
     &       (gtwr(j)-mu*dgambas*hmixr(j)
     &      - ystwr(j)*(1.0d+0+3.0d+0*mu*depsl/yssl))
          end do

c         if slip is most negative, record index with most negative slip
          if (dgamtw(i).lt.minslip) then
            minslip = dgamtw(i)
            minslipindex = i
          end if
        end do

c       if neg slip occurred, figure out if most neg entry has an
c         active pair with equal slip. if so, check which resolved
c         shear stress is higher and eliminate that
        if (minslip.lt.0.0d+0) then
          call identify_moreneg_twinpair(minslipindex, acttwsys,
     &       dgamtw, gtwr, dgambas, hmixr, mu)
          call remove_acttwsys_entry(acttwsys, minslipindex)
          dgamtw(minslipindex) = 0.0d+0
          nact = nact - 1
        else
c         set negslip = false if everything is in equilibrium
c         otherwise, set negslip = true and eliminate a deformation mode
c         -- only eliminate twinning (not bas slip)

          call twin_equilm_check(acttwsys, nact, dgamtw, negslip,
     &       dgambas, hmixr, gtwr, ystwr, mu, depsl, yssl, deactTOL)

        end if

c       go back through loop (only happens if neg slip occurs)
        goto 20
      end if

      return
      end

c==================================================================
c==================================================================
c  The input to this is minslipindex, which identifies the most neg slip
c  -- This may not select the right system to delete since slip pairs
c     are identical (due to SVD) if both are active. Therefore, make
c     sure to eliminate the other with the lowest modified shear stress
c
c  The output is minslipindex, modified if necessary to reflect the
c    twin pair with the lowest crss if they are both active
c------------------------------------------------------------------

      subroutine identify_moreneg_twinpair(minslipindex, acttwsys,
     &  dgamtw, gtwr, dgambas, hmixr, mu)

      implicit none

c     input/output
```

```

      integer minslipindex

c     input
      integer acttwsys(6)
      double precision dgamtw(6), gtwr(6), dgambas, hmixr(6), mu

c     util
      integer nact, j, minslipindexfull, di
      double precision tempvar, tempvar2

c     find what index minslipindex corresponds to in unreduced notation
      nact = 0
      do j=1,6
        if (acttwsys(j).eq.1) then
          nact = nact + 1
          if (nact.eq.minslipindex) then
            minslipindexfull = j
          end if
        end if
      end do

c     determine if pair index is 1 higher or lower. set di accordingly
      if ((minslipindexfull.eq.1).or.(minslipindexfull.eq.3).or.
     & (minslipindexfull.eq.5)) then
        di = 1
      else
        di = -1
      end if

c     determine if pair is active and has the same negative slip amount
      tempvar = dabs(dgamtw(minslipindex)-dgamtw(minslipindex+di))
      if ((acttwsys(minslipindexfull+di).eq.1).and.
     & (tempvar.le.1d-6)) then
        CONTINUE
      else
        RETURN
      end if

c     if the pair is active, determine which has the lowest modified
c       crss, and select that one as the one to eliminate
      tempvar = gtwr(minslipindex) - mu*hmixr(minslipindex)*
     &  dgambas
      tempvar2=gtwr(minslipindex+di)- mu*hmixr(minslipindex+di)*
     &  dgambas
      if (tempvar.gt.tempvar2) then
        minslipindex = minslipindex + di
      end if

      return
      end

c==================================================================
c==================================================================
c  For twinning, after it is determined there is no negative slip, do
c    this check to make sure tau>ys. If not, eliminate the most neg
c    twin system by using tau - ys = taurel.
c    -  If everything is in equilm, set negslip = false.
c    -  If sys eliminated, change nact, acttwsys, and set dgamtw(i) = 0
c------------------------------------------------------------------
      subroutine twin_equilm_check(acttwsys, nact, dgamtw, negslip,
     &   dgambas, hmixr, gtwr, ystwr, mu, depsl, yssl, deactTOL)

      implicit none

c     input/output
```

```
      integer acttwsys(6), nact
      double precision dgamtw(6)
      logical negslip

c     input
      double precision dgambas, hmixr(6), gtwr(6), ystwr(6), mu
      double precision depsl, yssl, deactTOL

c     util
      integer i,j
      double precision mhatr(6,6), taur(6), maxrel
      integer maxrelindex

      maxrel = 100.0d0
      maxrelindex = 0

      call calc_mhat(acttwsys, mhatr)
      do i=1,nact
        taur(i) = gtwr(i) - hmixr(i)*dgambas/2.0d+0
        do j=1,nact
          taur(i) = taur(i) - 2.0d+0*mu*mhatr(i,j)*dgamtw(j)
        end do
        taur(i) = taur(i) / (1.0d+0+3.0d+0*mu*depsl/yssl)

c        print*, 'tau - ys, red sys ', i, ': ', taur(i)-ystwr(i)
        if (taur(i)-ystwr(i).le.maxrel) then
          maxrel = taur(i)-ystwr(i)
c          print*, 'maxrel: ', maxrel
          maxrelindex = i
        end if
      end do

c     if everything is equilibrium, set negslip = false
      if (maxrel/mu.le.-deactTOL) then
        dgamtw(maxrelindex) = 0.0d+0
        call remove_acttwsys_entry(acttwsys, maxrelindex)
        nact = nact - 1
      else
        negslip = .false.
      end if

      return
      end

c=====================================================================
c=====================================================================
c  Given an index from the reduced representation, deactivates this
c  currently active entry from the full representation of acttwsys
c  -  ex: given acttwsys = (1,0,1,0,1,0) and i = 2
c         acttwsys becomes (1,0,0,0,1,0) because the second active
c         entry has been deleted
c---------------------------------------------------------------------
      subroutine remove_acttwsys_entry(acttwsys, i)

      implicit none
c     input/output
      integer acttwsys(6)

c     input
      integer i

c     util
      integer nact, j

      nact = 0
      do j=1,6
```

```
        if (acttwsys(j).eq.1) then
          nact = nact + 1
          if (nact.eq.i) then
            acttwsys(j) = 0
          end if
        end if
      end do

      return
      end

c==================================================================
c==================================================================
c  Calculate acttwsys based on trial stress and strength
c------------------------------------------------------------------

      subroutine calc_acttwsys_stress(gtw, ystw, acttwsys)

      implicit none

c     input
      double precision gtw(6), ystw(6)

c     output
      integer acttwsys(6)

c     util
      integer n

      do n=1,6
        if (gtw(n).ge.ystw(n)) then
          acttwsys(n) = 1
        else
          acttwsys(n) = 0
        end if
      end do

      return
      end

c==================================================================
c  Calculate phtwr based on phtw
c------------------------------------------------------------------

      subroutine calc_reduced_phtw(phtw, acttwsys, phtwr)

      implicit none

c     input
      double precision phtw(3,3,6)
      integer acttwsys(6)

c     output
      double precision phtwr(3,3,6)

c     util
      integer i, n, nact

      nact = 0
      do n=1,6
        phtwr(1,1,n) = 0.0d+0
        phtwr(1,2,n) = 0.0d+0
        phtwr(1,3,n) = 0.0d+0
        phtwr(2,1,n) = 0.0d+0
        phtwr(2,2,n) = 0.0d+0
        phtwr(2,3,n) = 0.0d+0
```

```
        phtwr(3,1,n) = 0.0d+0
        phtwr(3,2,n) = 0.0d+0
        phtwr(3,3,n) = 0.0d+0
        nact = nact + acttwsys(n)
      end do

      i = 1
      n = 1

   10 if (i.le.nact) then
        if (acttwsys(n).eq.1) then
          phtwr(1,1,i) = phtw(1,1,n)
          phtwr(1,2,i) = phtw(1,2,n)
          phtwr(1,3,i) = phtw(1,3,n)
          phtwr(2,1,i) = phtw(2,1,n)
          phtwr(2,2,i) = phtw(2,2,n)
          phtwr(2,3,i) = phtw(2,3,n)
          phtwr(3,1,i) = phtw(3,1,n)
          phtwr(3,2,i) = phtw(3,2,n)
          phtwr(3,3,i) = phtw(3,3,n)
          i = i + 1
        end if
        n = n + 1
        goto 10
      end if

      return
      end

c=====================================================================
c=====================================================================
c  Calculate gtwr, ystwr based on their original values and acttwsys
c  - ex: if gtw0 = (1,2,3,4,5), and acttwsys = (1,0,1,0,1) then
c            gtwr = (1,3,5,0,0), and is used in dgamtw calculations
c---------------------------------------------------------------------


      subroutine calc_reduced_gys(gtw0, ystw0, acttwsys, gtwr, ystwr)

      implicit none

c     input
      double precision gtw0(6), ystw0(6)
      integer acttwsys(6)

c     output
      double precision gtwr(6), ystwr(6)

c     util
      integer i, n

      do n=1,6

        gtwr(n) = 0.0d+0
        ystwr(n) = 0.0d+0
      end do

      n = 1
      do i=1,6
        if (acttwsys(i).eq.1) then
          gtwr(n) = gtw0(i)
          ystwr(n) = ystw0(i)
          n = n + 1
        end if
      end do

```

```
      return
      end


c=====================================================================
c=====================================================================
c  Calculate hmix based on hmix0 and which systems are active
c    ex - if acttwsys is (0,1,0,1,1,0) then
c         hmix(1) = hmix0(2), hmix(2) = hmix0(4), hmix(3) = hmix0(5)
c         and hmix(4-6) = 0
c---------------------------------------------------------------------

      subroutine calc_reduced_hmix(hmix0, acttwsys, hmix)

      implicit none

c     input
      double precision hmix0(6)
      integer acttwsys(6)

c     output
      double precision hmix(6)

c     util
      integer i, n, nact

      nact = 0
      do n=1,6
        hmix(n) = 0.0d+0
        nact = nact + acttwsys(n)
      end do

      i = 1
      n = 1

   10 if (i.le.nact) then
        if (acttwsys(n).eq.1) then
          hmix(i) = hmix0(n)
          i = i + 1
        end if
        n = n + 1
        goto 10
      end if

      return
      end

c=====================================================================
c=====================================================================
c  Calculate h^\alpha = ptw_^\alpha:pbas
c---------------------------------------------------------------------


      subroutine init_hmix(ptw, pbas, hmix)

      implicit none

c     input
      double precision ptw(3,3,6), pbas(3,3)

c     output
      double precision hmix(6)

c     util
      integer n

```

```
      do n=1,6
        hmix(n) =    2.0d+0*(ptw(1,1,n)*pbas(1,1) + ptw(1,2,n)*pbas(1,2)
     &   +ptw(1,3,n)*pbas(1,3)+ptw(2,1,n)*pbas(2,1)+ptw(2,2,n)*pbas(2,2)
     &   +ptw(2,3,n)*pbas(2,3)+ptw(3,1,n)*pbas(3,1)+ptw(3,2,n)*pbas(3,2)
     &   +ptw(3,3,n)*pbas(3,3))
      end do

      return
      end

c=====================================================================
c=====================================================================
c  Calculate driving forces for yield based on trial stress for
c    basal, twinning, and non-basal slip
c---------------------------------------------------------------------

      subroutine calc_taus(sigd, pbas, ptw, gbas, gtw, gsl)

      implicit none

      !input
      double precision sigd(3,3), pbas(3,3), ptw(3,3,6)

      !output
      double precision gbas, gtw(6), gsl

      !util
      integer i



c      do i=1,6
c        gtw(i) = sigd(1,1)*ptw(1,1,i)+sigd(1,2)*ptw(1,2,i) +
c     &    sigd(1,3)*ptw(1,3,i)+sigd(2,1)*ptw(2,1,i)+
c     &    sigd(2,2)*ptw(2,2,i)+sigd(2,3)*ptw(2,3,i)+
c     &    sigd(3,1)*ptw(3,1,i)+sigd(3,2)*ptw(3,2,i)+
c     &    sigd(3,3)*ptw(3,3,i)
c      end do

      do i=1,6
        gtw(i) = sigd(1,1)*ptw(1,1,i)+sigd(2,2)*ptw(2,2,i)+
     &    sigd(3,3)*ptw(3,3,i)+2.0d+0*(sigd(1,2)*ptw(1,2,i)+
     &    sigd(1,3)*ptw(1,3,i)+sigd(2,3)*ptw(2,3,i))
      end do

      gbas = sigd(1,1)*pbas(1,1)+sigd(2,2)*pbas(2,2)+
     &    sigd(3,3)*pbas(3,3)+2.0d+0*(sigd(1,2)*pbas(1,2)+
     &    sigd(1,3)*pbas(1,3)+sigd(2,3)*pbas(2,3))

      gsl = dsqrt(3.0d+0/2.0d+0*(sigd(1,1)**2+
     & sigd(2,2)**2+sigd(3,3)**2+2.0d+0*(sigd(1,2)**2 +
     & sigd(1,3)**2 + sigd(2,3)**2)))

      return
      end


c=====================================================================
c=====================================================================
c  Calculate which deformation modes are potentially active based on
c    their orientation from the last time step and the trial stress
c---------------------------------------------------------------------

      subroutine calc_potactive_modes(gbas,gtw,gsl,ysbas,ystw,captwin,
     &  yssl, actTOL, mu, actbas, acttw, actsl)
```

```

      implicit none

c     input
      double precision gbas, gtw(6), gsl
      double precision ysbas, ystw(6), yssl, actTOL, mu
      logical captwin

c     output
      logical actbas, acttw, actsl

c     util
      integer n
      double precision diff

c     basal
      diff = (dabs(gbas)-ysbas)/mu
      if (diff.ge.actTOL) then
        actbas = .true.
      else
        actbas = .false.
      endif

c     twin
      acttw = .false.
      do n=1,6
        diff = (gtw(n)-ystw(n))/mu
        if (diff.ge.actTOL) then
          acttw = .true.
        end if
      end do
      if (captwin) acttw = .false.

c     non-basal slip
      diff = dabs(gsl) - yssl
      if (diff.ge.actTOL) then
        actsl = .true.
      else
        actsl = .false.
      endif

      return
      end

c=====================================================================
c=====================================================================
c  Calculate sbasal and Pbasal based on trial stress, mbasal
c  o=sig, m = mbas, s=sbas, p=pbas
c---------------------------------------------------------------------

      subroutine calc_spbas(o, m, s, p)

      implicit none

c     input
      double precision o(3,3), m(3)

c     output
      double precision s(3), p(3,3)

c     util
      double precision num1(3)
      double precision den1, num2

c       den1 = den1 + m(i)*o(i,j)*o(j,k)*m(k)
```

```
       den1 = m(1)*o(1,1)**2*m(1) + m(2)*o(1,1)*o(1,2)*m(1)
      &  + m(1)*o(1,2)**2*m(1) + m(3)*o(1,1)*o(1,3)*m(1)
      &  + m(1)*o(1,3)**2*m(1) + m(2)*o(1,2)*o(2,2)*m(1)
      &  + m(3)*o(1,2)*o(2,3)*m(1) + m(2)*o(1,3)*o(2,3)*m(1)
      &  + m(3)*o(1,3)*o(3,3)*m(1) +  m(1)*o(1,1)*o(1,2)*m(2)
      &  + m(2)*o(1,2)**2*m(2) + m(3)*o(1,2)*o(1,3)*m(2)
      &  + m(1)*o(1,2)*o(2,2)*m(2) + m(2)*o(2,2)**2*m(2)
      &  + m(1)*o(1,3)*o(2,3)*m(2) +  m(3)*o(2,2)*o(2,3)*m(2)
      &  + m(2)*o(2,3)**2*m(2) + m(3)*o(2,3)*o(3,3)*m(2)
      &  + m(1)*o(1,1)*o(1,3)*m(3) + m(2)*o(1,2)*o(1,3)*m(3)
      &  + m(3)*o(1,3)**2*m(3) + m(1)*o(1,2)*o(2,3)*m(3)
      &  + m(2)*o(2,2)*o(2,3)*m(3) + m(3)*o(2,3)**2*m(3)
      &  + m(1)*o(1,3)*o(3,3)*m(3) + m(2)*o(2,3)*o(3,3)*m(3)
      &  + m(3)*o(3,3)**2*m(3)

c      num1(i) = num1(i) + sig(i,j)*mbas(j)
c      num2 = num2 + mbas(i)*sig(i,j)*mbas(j)
       num1(1) = o(1,1)*m(1)+o(1,2)*m(2)+o(1,3)*m(3)
       num1(2) = o(2,1)*m(1)+o(2,2)*m(2)+o(2,3)*m(3)
       num1(3) = o(3,1)*m(1)+o(3,2)*m(2)+o(3,3)*m(3)
       num2 = m(1)*(o(1,1)*m(1)+o(1,2)*m(2)+o(1,3)*m(3))
      &     + m(2)*(o(2,1)*m(1)+o(2,2)*m(2)+o(2,3)*m(3))
      &     + m(3)*(o(3,1)*m(1)+o(3,2)*m(2)+o(3,3)*m(3))

       if ((den1-num2**2).le.1d-10) then
c        there is no shear stress on the basal plane, regardless of sbas
c          so assign sbas an arbitrary direction
         s(1) = 1.0d+0
         s(2) = 0.0d+0
         s(3) = 0.0d+0
       else
         s(1) = (num1(1) - m(1)*num2)/ dsqrt(den1-num2**2)
         s(2) = (num1(2) - m(2)*num2)/ dsqrt(den1-num2**2)
         s(3) = (num1(3) - m(3)*num2)/ dsqrt(den1-num2**2)
       end if

       p(1,1) = s(1)*m(1)
       p(2,2) = s(2)*m(2)
       p(3,3) = s(3)*m(3)
       p(1,2) = 0.5d+0*(s(1)*m(2)+s(2)*m(1))
       p(2,1) = p(1,2)
       p(1,3) = 0.5d+0*(s(1)*m(3)+s(3)*m(1))
       p(3,1) = p(1,3)
       p(2,3) = 0.5d+0*(s(2)*m(3)+s(3)*m(2))
       p(3,2) = p(2,3)

       return
       end

c=====================================================================
c=====================================================================
c  Return deviatoric trial stress in indicial notation

c---------------------------------------------------------------------

       subroutine calc_trial_devstress(STRESS, DSTRAIN, MU, SIGTDEV)

       implicit none

c      input
       double precision STRESS(6), DSTRAIN(6)
       double precision MU

c      output
       double precision SIGTDEV(3,3)

```

```
c     util
      double precision DEPSH, SIGH0
      double precision DEPSD1, DEPSD2, DEPSD3, DEPSD4, DEPSD5, DEPSD6

C DEVIATORIC PART OF THE STRAIN INCREMENT
C
      DEPSH=(DSTRAIN(1)+DSTRAIN(2)+DSTRAIN(3))/3.0d+0
      DEPSD1=DSTRAIN(1)-DEPSH
      DEPSD2=DSTRAIN(2)-DEPSH
      DEPSD3=DSTRAIN(3)-DEPSH
      DEPSD4=DSTRAIN(4)/2.0d+0
      DEPSD5=DSTRAIN(5)/2.0d+0
      DEPSD6=DSTRAIN(6)/2.0d+0

      SIGH0=(STRESS(1)+STRESS(2)+STRESS(3))/3.0d+0
      SIGTDEV(1,1)=STRESS(1)-SIGH0+2.0d+0*MU*DEPSD1
      SIGTDEV(2,2)=STRESS(2)-SIGH0+2.0d+0*MU*DEPSD2
      SIGTDEV(3,3)=STRESS(3)-SIGH0+2.0d+0*MU*DEPSD3
      SIGTDEV(1,2)=STRESS(4)+2.0d+0*MU*DEPSD4
      SIGTDEV(2,1)=STRESS(4)+2.0d+0*MU*DEPSD4
      SIGTDEV(1,3)=STRESS(5)+2.0d+0*MU*DEPSD5
      SIGTDEV(3,1)=STRESS(5)+2.0d+0*MU*DEPSD5
      SIGTDEV(2,3)=STRESS(6)+2.0d+0*MU*DEPSD6
      SIGTDEV(3,2)=STRESS(6)+2.0d+0*MU*DEPSD6

c      SIGT=SIGDEV(1)**2+SIGDEV(2)**2+SIGDEV(3)**2+2.0d+0*(SIGDEV(4)**2
c     & + SIGDEV(5)**2 + SIGDEV(6)**2)
c      SIGT=DSQRT(3.0d+0/2.0d+0*SIGT)


      return
      end

c=====================================================================
c=====================================================================
c  Just return ddsdde for lin elast umat
c---------------------------------------------------------------------

      subroutine elastddsdde(lam, mu, ntens, ndi, ddsdde)
      implicit none

      !input
      integer i,j, ntens, ndi
      double precision lam, mu

      !output
      double precision ddsdde(ntens,ntens)

      do i=1,ntens
        do j=1,ntens
          ddsdde(i,j) = 0.0d0
        end do
      end do

      do i=1, ndi
        do j=1, ndi
          ddsdde(j,i) = lam
        end do
        ddsdde(i,i) = lam+2.0d0*mu
      end do

      do i=ndi+1, ntens
        ddsdde(i,i) = mu
      end do

      return
```

```
      end

c==================================================================
c==================================================================
c   Calculate stwin, mtwin, and mbasal based on re, where re has been
c     initialized to include the initial rotations in uinit
c   Depenencies - init_stw_mtw
c------------------------------------------------------------------

      subroutine calc_stw_mtw_mbas(re, stw, mtw, mbas)
      implicit none

      !input
      double precision re(3,3)

      !output
      double precision stw(3,6), mtw(3,6), mbas(3)

      !util
      integer n
      double precision stw0(3,6), mtw0(3,6), mbas0(3)

      !initialize all the variables in the reference frame
      call init_stw_mtw(stw0, mtw0)
      mbas0(1) = 0.0d+0
      mbas0(2) = 0.0d+0
      mbas0(3) = 1.0d+0

c     rotate twin system
      do n=1,6
        stw(1,n)=re(1,1)*stw0(1,n)+re(1,2)*stw0(2,n)+re(1,3)*stw0(3,n)
        stw(2,n)=re(2,1)*stw0(1,n)+re(2,2)*stw0(2,n)+re(2,3)*stw0(3,n)
        stw(3,n)=re(3,1)*stw0(1,n)+re(3,2)*stw0(2,n)+re(3,3)*stw0(3,n)
        mtw(1,n)=re(1,1)*mtw0(1,n)+re(1,2)*mtw0(2,n)+re(1,3)*mtw0(3,n)
        mtw(2,n)=re(2,1)*mtw0(1,n)+re(2,2)*mtw0(2,n)+re(2,3)*mtw0(3,n)
        mtw(3,n)=re(3,1)*mtw0(1,n)+re(3,2)*mtw0(2,n)+re(3,3)*mtw0(3,n)
      end do

      mbas(1)=re(1,1)*mbas0(1)+re(1,2)*mbas0(2)+re(1,3)*mbas0(3)
      mbas(2)=re(2,1)*mbas0(1)+re(2,2)*mbas0(2)+re(2,3)*mbas0(3)
      mbas(3)=re(3,1)*mbas0(1)+re(3,2)*mbas0(2)+re(3,3)*mbas0(3)

      return
      end

c==================================================================
c==================================================================
c  Calculate ptw and phtw based on stw, mtw, and mbas (c)
c------------------------------------------------------------------

      subroutine calc_ptw_phtw(stw, mtw, c, ptw, phtw)
      !recall mbas = c
      ! to test, form pcirc from subtracted term on phtw and then
      !   ensure that phtw:pcirc = 0 for all 6 systems

      implicit none

      !input
      double precision stw(3,6), mtw(3,6), c(3)

      !output
      double precision ptw(3,3,6), phtw(3,3,6)

      !util
      integer n
```

```
      double precision a(3), b

      !calculate ptwin
      do n=1,6
        ptw(1,1,n) = 0.5d+0*(stw(1,n)*mtw(1,n)+stw(1,n)*mtw(1,n))
        ptw(1,2,n) = 0.5d+0*(stw(1,n)*mtw(2,n)+stw(2,n)*mtw(1,n))
        ptw(1,3,n) = 0.5d+0*(stw(1,n)*mtw(3,n)+stw(3,n)*mtw(1,n))
        ptw(2,1,n) = ptw(1,2,n)
        ptw(2,2,n) = 0.5d+0*(stw(2,n)*mtw(2,n)+stw(2,n)*mtw(2,n))
        ptw(2,3,n) = 0.5d+0*(stw(2,n)*mtw(3,n)+stw(3,n)*mtw(2,n))
        ptw(3,1,n) = ptw(1,3,n)
        ptw(3,2,n) = ptw(2,3,n)
        ptw(3,3,n) = 0.5d+0*(stw(3,n)*mtw(3,n)+stw(3,n)*mtw(3,n))

c       a = ptw.c
        a(1)=ptw(1,1,n)*c(1)+ptw(1,2,n)*c(2)+ptw(1,3,n)*c(3)
        a(2)=ptw(2,1,n)*c(1)+ptw(2,2,n)*c(2)+ptw(2,3,n)*c(3)
        a(3)=ptw(3,1,n)*c(1)+ptw(3,2,n)*c(2)+ptw(3,3,n)*c(3)

c       b = c.ptw.c = c.a
        b = a(1)*c(1)+a(2)*c(2)+a(3)*c(3)

c       pcirc
c       pc(i,j,n) = a(i,n)*c(j)+c(i)*a(j,n)-c(i)*c(j)*b(n)

c       phtw = ptw - pc
        phtw(1,1,n)=ptw(1,1,n)-
     &    (a(1)*c(1)+c(1)*a(1)-2.0d+0*c(1)*c(1)*b)
        phtw(1,2,n) = ptw(1,2,n)-
     &    (a(1)*c(2)+c(1)*a(2)-2.0d+0*c(1)*c(2)*b)
        phtw(1,3,n) = ptw(1,3,n)-
     &    (a(1)*c(3)+c(1)*a(3)-2.0d+0*c(1)*c(3)*b)
        phtw(2,1,n) = ptw(2,1,n)-
     &    (a(2)*c(1)+c(2)*a(1)-2.0d+0*c(2)*c(1)*b)
        phtw(2,2,n) = ptw(2,2,n)-
     &    (a(2)*c(2)+c(2)*a(2)-2.0d+0*c(2)*c(2)*b)
        phtw(2,3,n) = ptw(2,3,n)-
     &    (a(2)*c(3)+c(2)*a(3)-2.0d+0*c(2)*c(3)*b)
        phtw(3,1,n) = ptw(3,1,n)-
     &    (a(3)*c(1)+c(3)*a(1)-2.0d+0*c(3)*c(1)*b)
        phtw(3,2,n) = ptw(3,2,n)-
     &    (a(3)*c(2)+c(3)*a(2)-2.0d+0*c(3)*c(2)*b)
        phtw(3,3,n) = ptw(3,3,n)-
     &    (a(3)*c(3)+c(3)*a(3)-2.0d+0*c(3)*c(3)*b)

      end do

      return
      end

c=====================================================================
c=====================================================================
c Initialize a reference s and m for twinning in hcp metals
c the value of H is specific to Magnesium.
c Notation is same as Staroselsky thesis pp. 110, i.e., x->a2
c note M0_3-6 THE C TERM SHOULD NOT BE DIVIDED BY 2
c---------------------------------------------------------------------

      subroutine init_stw_mtw(s,m)
      implicit none


      double precision s(3,6), m(3,6)
      double precision H, TWO, THREE, ZERO, DEN1, DEN2, S3
      parameter (H=1.624D+0,TWO=2.0D+0,THREE=3.0D+0,ZERO=0.0D+0)
```

```

      S3 = dsqrt(3.0d+0)
      DEN1 = dsqrt(3.0d+0+H**2)
      DEN2 = 2.0d+0*DEN1

c                         TOP HEXAGON
c       _1_      ___      ___      ___      ___      ___
c      /   \    /   \    /   \3   /   \   5/   \    /   \
c      \___/    \___/    \___/   4\___/    \___/    \___/6
c                 2
c
c       ___      _2_      ___      ___      ___      ___
c      /   \    /   \    /   \    /   \4   /   \   6/   \
c      \___/    \___/   3\___/    \___/    \___/5   \___/
c        1
c
c                       BOTTOM HEXAGON

c              y
c              |
c              |____ x
c             /
c            /
c           z

      s(1,1) = ZERO
      s(2,1) = S3/DEN1
      s(3,1) = H/DEN1
      m(1,1) = ZERO
      m(2,1) = -H/DEN1
      m(3,1) = S3/DEN1
      s(1,2) = ZERO
      s(2,2) = -S3/DEN1
      s(3,2) = H/DEN1
      m(1,2) = ZERO
      m(2,2) = H/DEN1
      m(3,2) = S3/DEN1
      s(1,3) = THREE/DEN2
      s(2,3) = S3/DEN2
      s(3,3) = TWO*H/DEN2
      m(1,3) = -S3*H/DEN2
      m(2,3) = -H/DEN2
      m(3,3) = TWO*S3/DEN2
      s(1,4) = -THREE/DEN2
      s(2,4) = -S3/DEN2
      s(3,4) = TWO*H/DEN2
      m(1,4) = S3*H/DEN2
      m(2,4) = H/DEN2
      m(3,4) = TWO*S3/DEN2
      s(1,5) = -THREE/DEN2
      s(2,5) = S3/DEN2
      s(3,5) = TWO*H/DEN2
      m(1,5) = S3*H/DEN2
      m(2,5) = -H/DEN2
      m(3,5) = TWO*S3/DEN2
      s(1,6) = THREE/DEN2
      s(2,6) = -S3/DEN2
      s(3,6) = TWO*H/DEN2
      m(1,6) = -S3*H/DEN2
      m(2,6) = H/DEN2
      m(3,6) = TWO*S3/DEN2

      return
      end

c=====================================================================
```

```
c===================================================================
c   Make sure R is a pure rotation matrix by iteratively determing
c      R from the input, considering R may have stretch and rotation
c   NOTE: This is because advection may alter R aphysically
c   AUTHOR: Rich Becker
c-------------------------------------------------------------------

      SUBROUTINE ensurerot(R)
      IMPLICIT NONE
C     Parameter variables
      DOUBLE PRECISION TWO
      PARAMETER (TWO = 2.D0)
C
C     Argument variables
      DOUBLE PRECISION R(3,3)
C
C     Local variables
      INTEGER I, J, K
C
      DOUBLE PRECISION A(3,3), AI(3,3), DET, ERR, TEST, TOL

      DATA TOL/1.D-8/
C
C RETURNS A PURE ROTATION FROM THE POLAR DECOMPOSITION OF THE
C INPUT MATRIX
C
C LIMIT NUMBER OF ITERATIONS TO 20. IF CONVERGENCE IS NOT
C REACHED, THE PROGRAM WILL STOP.
C
      DO 40 K=1,20
      DO 10 I=1,3
      DO 10 J=1,3
   10 A(I,J)=R(I,J)
C
C
C COMPUTE INVERSE OF A
C
      DET=A(1,1)*(A(2,2)*A(3,3)-A(3,2)*A(2,3))
     1   -A(1,2)*(A(2,1)*A(3,3)-A(3,1)*A(2,3))
     1   +A(1,3)*(A(2,1)*A(3,2)-A(3,1)*A(2,2))
C
      AI(1,1)=(A(2,2)*A(3,3)-A(3,2)*A(2,3))/DET
      AI(1,2)=(A(3,2)*A(1,3)-A(1,2)*A(3,3))/DET
      AI(1,3)=(A(1,2)*A(2,3)-A(1,3)*A(2,2))/DET
      AI(2,1)=(A(3,1)*A(2,3)-A(2,1)*A(3,3))/DET
      AI(2,2)=(A(1,1)*A(3,3)-A(3,1)*A(1,3))/DET
      AI(2,3)=(A(1,3)*A(2,1)-A(1,1)*A(2,3))/DET
      AI(3,1)=(A(2,1)*A(3,2)-A(2,2)*A(3,1))/DET
      AI(3,2)=(A(1,2)*A(3,1)-A(1,1)*A(3,2))/DET
      AI(3,3)=(A(1,1)*A(2,2)-A(1,2)*A(2,1))/DET
C
C AVERAGE THE MATRIX AND THE TRANSPOSE OF ITS INVERSE.
C
      DO 20 I=1,3
      DO 20 J=1,3
   20 R(I,J)=(A(I,J)+AI(J,I))/TWO
C
C DETERMINE LARGEST DEVIATION AND CHECK AGAINST TOLERANCE
C
      ERR=0.0
      DO 30 I=1,3
      DO 30 J=1,3
      TEST=ABS(R(I,J)-A(I,J))
   30 IF(TEST.GT.ERR)ERR=TEST
      IF(ERR.LT.TOL) RETURN
C
```

```
C IF ERROR CONDITION IS MET BEFORE 20 ITERATIONS, CONTROL IS
C RETURNED TO THE MAIN PROGRAM. OTHERWISE,
C
   40 CONTINUE
      WRITE(6,100)
  100 FORMAT(' NOT A PURE ROTATION')
      STOP
      END
C

c=====================================================================
c=====================================================================
c   Create rotation matrix from Euler angles in bunge notation
c---------------------------------------------------------------------

      subroutine create_rmat_bunge(a1, ap, a2, rmat)
      implicit none

      double precision a1, ap, a2, s1, c1, sp, cp, s2, c2
      double precision rmat(3,3)

      s1 = sin(a1)
      c1 = cos(a1)
      sp = sin(ap)
      cp = cos(ap)
      s2 = sin(a2)
      c2 = cos(a2)

      rmat(1,1) = c1*c2 - s1*s2*cp
      rmat(2,1) = s1*c2 + c1*s2*cp
      rmat(3,1) = s2*sp
      rmat(1,2) = -c1*s2-s1*c2*cp
      rmat(2,2) = -s1*s2+c1*c2*cp
      rmat(3,2) = c2*sp
      rmat(1,3) = s1*sp
      rmat(2,3) = -c1*sp
      rmat(3,3) = cp

      end

      double precision function determinant(a)
      ! Calculate the determinant of a 3 x 3 matrix.

      implicit none

      double precision a(3,3)
      double precision b1, b2, b3


      b1 = a(2,2) * a(3,3) - a(3,2) * a(2,3)
      b2 = a(3,1) * a(2,3) - a(2,1) * a(3,3)
      b3 = a(2,1) * a(3,2) - a(3,1) * a(2,2)

      determinant = a(1,1) * b1 + a(1,2) * b2 + a(1,3) * b3
      if (dabs(determinant).le.1d-10) then
        print*, 'WARNING: JUST TOOK DET LESS THAN 1E-10'
      end if

      return
      end
```