



This repository Search

Pull requests Issues Gist



zubaexy / MgSpall2

Unwatch 1

Star 0

Fork 0

branch: master

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



```

1
2 c-----
3 c   This umat is called by ale3d and wraps around a regular
4 c   abaqus umat call. Main differences are cmname, and initialization
5 c   is handled differently.
6 c-----
7
8     subroutine umat ( stress, statev, ddsdde, sse,   spd,
9 &                   scd,   rpl,   ddsddt, drplde, drpldt,
10 &                   strain, dstrain, time,   dtime, temp,
11 &                   dtemp, predef, dpred,   cmname, ndi,
12 &                   nshr, ntens, nstatv, props, nprops,
13 &                   coords, drot,   pnwtdt, celent, dfgrd0,
14 &                   dfgrd1, noel,   npt,   layer, kspt,
15 &                   kstep, kinc )
16
17     implicit none
18
19     ! Dimension variables passed into the UMAT sub (not all are used)
20     integer ndi, nshr, ntens, nstatv, nprops, noel
21     integer layer, kspt, kstep, kinc, npt
22     double precision cmname, sse, celent, dtime, temp, dtemp, pnwtdt
23     double precision spd, scd, rpl, drpldt
24     character*7 cmnameale ! Material name
25     double precision coords(3), ddsdde(6,6), ddsddt(ntens)
26     double precision dfgrd1(3,3), dfgrd0(3,3), dpred(1), drplde(ntens)
27     double precision drot(3,3), dstrain(ntens), predef(1)
28     double precision props(nprops), statev(nstatv), strain(ntens)
29     double precision stress(ntens), time(2)
30
31 c   Helper variables
32     integer orient, ngrains
33     double precision seed
34     integer nstatvpg, nstatvreg
35     parameter (nstatvreg=25, nstatvpg=31)
36     cmnameale = 'magnesm'
37
38     if (nstatv.eq.nstatvreg) then
39         ngrains = 1
40     else
41         ngrains = nstatv / nstatvpg
42     end if
43
44 c   if first step perform initialization
45 c   statev 1 is random seed (0-1), 2 is orientation, 3 is num xtals
46     if ((time(1)-dtime).le.(0.5d+0*dtime)) then
47         seed = statev(1)
48         orient = int(statev(2))
49         call sdvinia(statev,nstatv,seed,orient,ngains)

```

```

50      end if
51
52
53      if (ngrains.eq.1) then
54          call cumat(stress, statev, ddsdde, sse,   spd,
55      &   scd,   rpl,   ddsddt, drplde, drpldt,
56      &   strain, dstrain, time,   dtime, temp,
57      &   dtemp, predef, dpred,   cmnameale, ndi,
58      &   nshr, ntens, nstatv, props, nprops,
59      &   coords, drot,   pnnewdt, celent, dfgrd0,
60      &   dfgrd1, noel,   npt,   layer, kspt,
61      &   kstep, kinc)
62      else
63          call pumat(stress, statev, ddsdde, sse,   spd,
64      &   scd,   rpl,   ddsddt, drplde, drpldt,
65      &   strain, dstrain, time,   dtime, temp,
66      &   dtemp, predef, dpred,   cmnameale, ndi,
67      &   nshr, ntens, nstatv, props, nprops,
68      &   coords, drot,   pnnewdt, celent, dfgrd0,
69      &   dfgrd1, noel,   npt,   layer, kspt,
70      &   kstep, kinc)
71      end if
72
73      return
74      end
75
76 C-----
77 c   ABAQUS STRESS - SIG11, SIG22, SIG33, SIG12, SIG13, SIG23
78 c   ABAQUS STRAIN - EPS11, EPS22, EPS33, GAM12, GAM13, GAM23
79 c                               WHERE GAM12 = 2*EPS12
80 C-----
81
82      subroutine cumat ( stress, statev, ddsdde, sse,   spd,
83      &   scd,   rpl,   ddsddt, drplde, drpldt,
84      &   strain, dstrain, time,   dtime, temp,
85      &   dtemp, predef, dpred,   cmname, ndi,
86      &   nshr, ntens, nstatv, props, nprops,
87      &   coords, drot,   pnnewdt, celent, dfgrd0,
88      &   dfgrd1, noel,   npt,   layer, kspt,
89      &   kstep, kinc )
90
91      IMPLICIT NONE
92
93      ! loop variables
94      integer i
95
96      ! Dimension variables passed into the UMAT sub (not all are used)
97      integer ndi      ! Number of direct stress components
98      integer nshr     ! Number of shear stress components
99      integer ntens    ! Size of stress or strain array (ndi + nshr)
100     integer nstatv   ! Number of SDVs
101     integer nprops   ! Number of material constants
102     integer noel     ! Element number
103     integer layer    ! Layer number (for composites)
104     integer kspt     ! Section point number within layer
105     integer kstep    ! Step number
106     integer kinc     ! Increment number
107     integer npt      ! Integration point number
108     character*7 cmname ! Material name
109     double precision sse ! Specific elastic strain energy
110
111     double precision
112     & celent,      ! Characteristic element length
113     & dtime,       ! Time increment
114     & temp,        ! Temperature at start of increment

```

```

115      & dtemp,          ! Temperature increment
116      & pnewdt,        ! Ratio of new time increment to time
117                      ! increment being used
118      & spd,           ! Specific plastic dissipation
119      & scd,           ! Specific creep dissipation
120      & rpl,           ! Volumetric heat generation per unit time
121      & drpldt,        ! Variation of rpl with temperature
122      & coords(3),     ! Coordinates of Gauss pt. being evaluated
123      & ddsde(ntens,ntens), ! Tangent Stiffness Matrix
124      & ddsddt(ntens), ! Change in stress per change in temperature
125      & dfgdr1(3,3),   ! Deformation gradient at end of step
126      & dfgdr0(3,3),   ! Deformation gradient at beginning of step
127      & dpred(1),      ! Change in predefined state variables
128      & drplde(ntens), ! Change in heat generation per change in strain
129      & drot(3,3),     ! Rotation matrix
130      & dstrain(ntens), ! Strain increment tensor stored in vector form
131      & predef(1),     ! Predefined state vars dependent on field
132                      ! variables
133      & props(nprops), ! Material properties passed in
134      & statev(nstatv), ! State Variables
135      & strain(ntens), ! Strain tensor stored in vector form
136      & stress(ntens), ! Cauchy stress tensor stored in vector form
137      & time(2)        ! Step Time and Total Time
138
139      !Variables fed in from props
140      double precision lam      ! Elastic lambda (MPa)
141      double precision mu       ! Shear modulus (MPa)
142      double precision grun     ! Grun coeff of MgO ~1.6 White JAP, 66
143      double precision temp0    ! Initial temperature
144      double precision tempmelt ! Melt temperature
145      double precision rho0     ! Initial density (ng/um^3)
146      double precision qbastw   ! Interaction param for bas and twin
147      double precision qbassl   ! "" bas and slip
148      double precision qtwbas   ! "" twin and bas
149      double precision qtwsl    ! "" twin and slip
150      double precision qslbas   ! "" slip and bas
151      double precision qsltw    ! "" slip and twin
152      integer hbastype          ! Basal slip hardening model used
153      double precision hbas1    ! Basal slip hardening parameter1
154      double precision hbas2    ! Basal slip hardening parameter2
155      double precision hbas3    ! Basal slip hardening parameter3
156      double precision hbas4    ! Basal slip hardening parameter4
157      double precision hbas5    ! Basal slip hardening parameter5
158      double precision hbas6    ! Basal slip hardening parameter6
159      integer htwtype           ! Twinning hardening model used
160      double precision htw1     ! Twinning hardening parameter 1
161      double precision htw2     ! Twinning hardening parameter 2
162      double precision htw3     ! Twinning hardening parameter 3
163      double precision htw4     ! Twinning hardening parameter 4
164      double precision htw5     ! Twinning hardening parameter 5
165      double precision htw6     ! Twinning hardening parameter 6
166      integer hsltype           ! Slip hardening model used
167      double precision hsl1     ! Slip hardening parameter 1
168      double precision hsl2     ! Slip hardening parameter 2
169      double precision hsl3     ! Slip hardening parameter 3
170      double precision hsl4     ! Slip hardening parameter 4
171      double precision hsl5     ! Slip hardening parameter 5
172      double precision hsl6     ! Slip hardening parameter 6
173      double precision hsl7     ! Slip hardening parameter 7
174      double precision hsl8     ! Slip hardening parameter 8
175      integer eosflag           ! 0 lin elast, 1 murn eos cv no art vis
176      c                          2 murn eos cv wart visc
177      double precision b0       ! Bulk modulus, Guinan and Stein, 74
178      double precision dbdp     ! Bulk mod deriv w.r.t p, "" ref
179      double precision cv       ! MPa/K, Lee, Int J Thermophys, 13

```

```

180
181     !Variables related to statev
182     double precision re(3,3)      ! 3x3 rotation matrix
183     double precision gambslip     ! Shear basal slip (notwin)
184     double precision gamtw(6)     ! Shear strain on twin systems
185     double precision epsl0        ! Nonbas effplastic strain prev step
186     double precision epdsl        ! " " rate
187     double precision epdsl0       ! " " from prev step step
188     double precision energy       ! Int energy / ref vol
189     double precision tempsv       ! Temperature stored as state var
190     double precision depbas, deptw !change in epeff due to bas, tw
191
192     !Utility
193     integer nexit      ! Determines how calc_epdsl exited
194 c    0=reg, 1=noiter, 2=nobound, 3=maxit
195     logical actbas, acttw, actsl !whether modes are active
196     logical firstsl          !true if first sl iteration, f other
197     double precision gbas, gsl !trial shear on these
198     double precision ysbas0, yssl0, yssl !initial and cur ys
199     double precision dyssldepd          !change in ys wrt epd
200     double precision depsl, dgambas, dgamslip !change in strains
201     double precision p0, dp, p          !initial pressure, and increment
202     double precision bmod              !bulk modulus
203     double precision epdslmax          !estimate of max slip strain rate
204     double precision epdslez          !epdsl calculation w/o bas or tw
205
206     double precision
207     & stw(3,6),          ! Slip dir vector for twin systems
208     & mtw(3,6),          ! Slip norm vector for twin systems
209     & mbas(3),           ! Slip norm vector for basal system
210     & ptw(3,3,6),        ! P for twinning
211     & phtw(3,3,6),       ! P hat for twinning
212     & ystw0(6),          ! All 6 twin taus from last step
213     & stressd0(3,3),     ! Deviatoric trial stress
214     & stressd(3,3),      ! Deviatoric stress
215     & sbas(3),           ! Slip dir vector for basal system
216     & pbas(3,3),         ! P for basal slip
217     & gtw(6),            ! Trial stress on twin systems
218     & hmix0(6),          ! Interaction b/w twin and bas - doesnt change
219     & dgamtw(6),         ! Delta gamma for twin in original order
220     & dgamtwr(6),        ! Delta Gamma for twin in reduced order
221     & sbslip(3),         ! Direction of bslip
222     & wpdt(3,3),         ! Plastic spin times dt
223     & wdt(3,3)          ! Spin times dt (from abaqus drot)
224
225     integer
226     & acttwsys(6)        ! List of active twin sys, 1 = act, 0 = inact
227
228 c    very solution oriented
229     logical reactivate ! If true dont delete defm modes in slip step
230     logical deactivatesl !If slip needs to be deactivated at end
231     logical captw
232     double precision reactTOL, deactTOL, actTOL
233     integer itNum, itMax
234     double precision twcap, gamtwtot
235
236 c-----
237 c    Solution related parameters
238 c-----
239     actTOL = 1.0d-8
240     deactTOL = 1.0d-4
241     reactTOL = 1.0d-8
242     itNum = 0
243     itMax = 10
244     reactivate = .false.
245     deactivatesl = .false.

```

```

246      firstsl = .true.
247
248      c-----
249      c    Read in material properties
250      c-----
251
252      lam = props(1)
253      mu = props(2)
254      grun = props(3)
255      temp0 = props(4)
256      tempmelt = props(5)
257      rho0 = props(6)
258      qbastw = props(7)
259      qbassl = props(8)
260      qtwbas = props(9)
261      qtwsl = props(10)
262      qslbas = props(11)
263      qsltw = props(12)
264      hbastype = props(13)
265      hbas1 = props(14)
266      hbas2 = props(15)
267      hbas3 = props(16)
268      hbas4 = props(17)
269      hbas5 = props(18)
270      hbas6 = props(19)
271      htwtype = props(20)
272      htw1 = props(21)
273      htw2 = props(22)
274      htw3 = props(23)
275      htw4 = props(24)
276      htw5 = props(25)
277      htw6 = props(26)
278      hsltype = props(27)
279      hsl1 = props(28)
280      hsl2 = props(29)
281      hsl3 = props(30)
282      hsl4 = props(31)
283      hsl5 = props(32)
284      hsl6 = props(33)
285      hsl7 = props(34)
286      hsl8 = props(35)
287      eosflag = props(36)
288      b0 = props(37)
289      dbdp = props(38)
290      cv = props(39)
291
292      c-----
293      c    Read in state variables
294      c-----
295
296      !Read in statev
297      call init_statevs(statev, re, gambslip, gamtw, epsl0,
298      & epdsl0, energy, tempsv)
299      temp = temp0
300      gamwtot = gamtw(1)+gamtw(2)+gamtw(3)+gamtw(4)+gamtw(5)+gamtw(6)
301
302      c-----
303      c    Store things before big loop
304      c-----
305
306      c    Store initial pressure
307      c    - could also solve hydrostatic part at beginning of time step
308      p0 = - (stress(1) + stress(2) + stress(3))/3.0d+0
309
310      c    Based on initial re from the previous step,

```

```

311 c      calculate stw, mtw, and mbas, as well as ptw, and phat tw
312 call ensurerot(re)
313 call calc_stw_mtw_mbas(re, stw, mtw, mbas)
314 call calc_ptw_phtw(stw, mtw, mbas, ptw, phtw)
315
316 c      Calculate trial stress
317 call calc_trial_devstress(stress, dstrain, mu, stressd0)
318
319 c      Calculate sbas, and pbas based on trial stress
320 call calc_spbas(stressd0, mbas, sbas, pbas)
321
322 c      Calculate trial stresses on each mode
323 call calc_taus(stressd0, pbas, ptw, gbas, gtw, gsl)
324
325 c      Calculate strengths for basal, twin, and nb slip
326 call calc_str_bas(gambslip, gamtw, epsl0, temp, hbastype, hbas1,
327 & hbas2, hbas3, hbas4, hbas5, hbas6, qbastw, qbass1, ysbas0)
328 call calc_str_tw(gambslip, gamtw, epsl0, temp, htwtype, htw1, htw2,
329 & htw3, htw4, htw5, htw6, qtwbas, qtwsl, ystw0, twcap, captw)
330 call calc_str_sl(.true., gambslip, gamtw, epsl0, epdsl0, temp,
331 & tempmelt, hsltype, hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8,
332 & qslbas, qsltw, yssl0, dyssldepd)
333
334 C=====
335 c      Start the solve for increment in plastic strain of each mech
336 C=====
337
338 c      Determine what deformation modes may potentially activate based
339 c      solely on the trial stress
340 call calc_potactive_modes(gbas, gtw, gsl, ysbas0, ystw0, captw,
341 & yssl0, actTOL, mu, actbas, acttw, actsl)
342
343 c      Do necessary pre-calcs and initializations
344 if (acttw) then
345     call init_hmix(ptw, pbas, hmix0)
346 end if
347 depl = 0.0d+0
348 dgambas = 0.0d+0
349 epdsl = 0.0d+0
350 yssl = yssl0
351
352 c      -- Basal slip routine --
353 c      actbas = .false. !Manually deactivate basal slip
354 if (actbas) then
355     call calc_dgambas(gbas, ysbas0, mu, depl, yssl0, dgambas)
356 end if
357
358 c      -- Twinning routine --
359 c      acttw = .false. !Manually deactivate twinning
360 if (acttw) then
361     call calc_dgamtw(gtw, hmix0, ystw0, yssl0, mu, dgambas,
362 & depl, deactTOL, acttwsys, dgamtwr)
363 else
364     do i=1,6
365         dgamtwr(i) = 0.0d+0
366         acttwsys(i) = 0
367     end do
368 end if
369
370 c      -- Non-basal slip routine --
371 c      actsl = .false. !Manually deactivate nb slip
372 22 if (actsl) then
373 c      Check if twinning or basal supressing nb slip - OBSOLETE
374 c      call suppress_slip_query(actbas, acttw, acttwsys, dgambas,
375 c      & pbas, dgamtwr, phtw, stressd0, mu, yssl0, actsl)

```

```

376         if (actsl) then
377 c           Only calculate epdslez once, on first sl iteration
378             if (firstsl) then
379 c               Calculate epdsl as if it is the only mechanism is active
380                 call calc_max_epdsl(dstrain,gsl,lam,mu,dtime,hsl8,epdslmax)
381                 call calc_epdsl_ez(gambslip, gamtw, epsl0, gsl,
382 &                   hsltype, hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8,
383 &                   tempmelt, qslbas, qsltw, stressd0, mu, temp, dtime, epdslmax,
384 &                   epdslez)
385                 epdsl = epdslez
386                 firstsl = .false.
387             else
388                 epdsl = epdslez
389             end if
390
391 c           If basal slip or twin active, do full solve
392             if (acttw.or.actbas) then
393                 epdslmax = epdsl
394                 call calc_epdsl(actbas, acttw, acttwsys, hmix0, yssl0, gbas,
395 &                   ysbas0, pbas, gambslip, gtw, ystw0, gamtw, phtw, epsl0, gsl,
396 &                   hsltype, hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8,
397 &                   tempmelt, qslbas, qsltw, stressd0, mu, temp, dtime, epdsl0,
398 &                   epdslmax, reactivate, epdsl, nexit)
399             end if
400 c           if nexit = 2, slip stop by twin/bas, nexit=3 did not converge
401             else
402                 epdsl = 0.0d+0
403             end if
404             depl = epdsl * dtime
405
406             call calc_str_sl(.true., gambslip, gamtw, epsl0, epdsl, temp,
407 &                   tempmelt, hsltype, hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8,
408 &                   qslbas, qsltw, yssl, dyssldepl)
409             if (epdsl.lt.hsl8) then
410                 epdsl = 0.0d+0
411                 actsl = .false.
412             end if
413         end if
414
415 c       Slip is converged. Recalculate basal and twin if slip occurred.
416 c       -- Basal slip routine --
417 23 if (actbas) then
418     if ((actsl).or.(deactivatesl)) then
419         call calc_dgambas(gbas, ysbas0, mu, depl, yssl, dgambas)
420     end if
421 else
422     dgambas = 0.0d+0
423 end if
424
425 c       -- Twinning routine --
426 if (acttw) then
427     if ((actsl).or.(deactivatesl)) then
428         call calc_dgamtw(gtw, hmix0, ystw0, yssl, mu, dgambas,
429 &                   depl, deactTOL, acttwsys, dgamtwr)
430     end if
431 else
432     do i=1,6
433         dgamtwr(i) = 0.0d+0
434         acttwsys(i) = 0
435     end do
436 end if
437
438 c       At the end: calculate the deviatoric part of the stress
439 call calc_return_dev_stress(stressd0, acttwsys, mu, dgamtwr,
440 & dgambas, depl, yssl, phtw, pbas, stressd)

```

```

441
442 c   Based on new stress, check if mechanisms should be reactivated
443   call reactivate_mechanisms(actbas, acttw, actsl, acttwsys, captw,
444   & stressd, pbas, ptw, ysbas0, ystw0, yssl, epdsl,
445   & reactTOL, deactTOL, mu, reactivate, deactivatesl)
446
447   itNum = itNum + 1
448
449   !slip needs to be deactivated, dont change anything else
450   if ((deactivatesl).and.(itNum.le.itMax).and.(reactivate)) go to 23
451   !twinning or basal slip needs to be reactivated
452   if ((reactivate).and.(itNum.le.itMax)) go to 22
453
454 c=====
455 c   End of large solve, solution has converged, calc state vars
456 c=====
457
458 c   Calculate dgambslip, sbslip
459   call calc_dgamsbslip(dgambas, dgamtwr, sbas, mbas, ptw, phtw,
460   & dgambslip, sbslip)
461
462 c   Update re by calculating plastic spin
463   call calc_wpdt(sbslip, mbas, dgambslip, wpdt)
464   call calc_wdt_abq(drot, wdt)
465   call update_re(actbas, wpdt, wdt, re)
466
467 c   Transform dgamtwr from the reduced to full frame
468   call calc_full_dgamtwr(acttwsys, dgamtwr, dgamtwr)
469   call update_dep(actbas, acttw, dgambslip, sbslip, mbas, dgamtwr,
470   & ptw, depbas, deptw)
471
472 c   Calculate the pressure contribution and return it to the stress
473   if (eosflag.eq.0) then
474     bmod = lam+2.0d+0*mu/3.0d+0
475     dp = - bmod*(dstrain(1) + dstrain(2) + dstrain(3))
476     p = p0+dp
477     tempsv = temp0
478   else
479     p = p0
480     call eos(eosflag, dfgrd0, dfgrd1, dtime, temp0, rho0, grun, b0,
481   & dbdp, cv, ysbas0, ystw0, yssl, dgambslip, dgamtwr, depsl,
482   & p, energy, tempsv)
483   end if
484
485   stress(1) = stressd(1,1) - p
486   stress(2) = stressd(2,2) - p
487   stress(3) = stressd(3,3) - p
488   stress(4) = stressd(1,2)
489   stress(5) = stressd(1,3)
490   stress(6) = stressd(2,3)
491
492   call update_statevs(re, dgambslip, dgamtwr, depsl, epdsl,
493   & energy, tempsv, depbas, deptw, dtime, statev)
494 c   Return ddsdde for abaqus implicit
495   call elastddsdde(lam, mu, ntens, ndi, ddsdde)
496
497   return
498 end
499
500 c=====
501 c=====
502 c   Construct state variable array from other variables
503 c   -- NOTE: epdeff is approx as epeff/(dtime+TOL) so no NaN if dtime=0
504 c-----
505   subroutine update_statevs(re, dgambslip, dgamtwr, depsl, epdsl,
   & energy, tempsv, depbas, deptw, dtime, statev)

```



```

506
507     implicit none
508
509 c     input
510     double precision re(3,3), dgambslip, dgamtw(6), depl, epdpl
511     double precision energy, tempsv, depbas, deptw, dtime
512
513 c     output
514     double precision statev(25)
515
516 c     util
517     double precision GAMTW, TOL
518     parameter (GAMTW=0.128917d+0, TOL=1.0d-12)
519
520     !Update state variables
521     statev(1) = re(1,1)
522     statev(2) = re(1,2)
523     statev(3) = re(1,3)
524     statev(4) = re(2,1)
525     statev(5) = re(2,2)
526     statev(6) = re(2,3)
527     statev(7) = re(3,1)
528     statev(8) = re(3,2)
529     statev(9) = re(3,3)
530     statev(10) = statev(10) + dgambslip
531     statev(11) = statev(11) + dgamtw(1)
532     statev(12) = statev(12) + dgamtw(2)
533     statev(13) = statev(13) + dgamtw(3)
534     statev(14) = statev(14) + dgamtw(4)
535     statev(15) = statev(15) + dgamtw(5)
536     statev(16) = statev(16) + dgamtw(6)
537     statev(17) = statev(17) + depl
538     statev(18) = epdpl
539     statev(19) = energy
540     statev(20) = tempsv
541     statev(21) = statev(21) + depbas
542     statev(22) = statev(22) + deptw
543     statev(23) = (statev(11) + statev(12) + statev(13) + statev(14) +
544 & statev(15) + statev(16))/GAMTW !vf twin
545     statev(24) = statev(24) + (depbas+deptw+depl)
546     statev(25) = (depbas+deptw+depl) / (dtime+TOL)
547     return
548     end
549
550 c=====
551 c=====
552 c Read in state variables array to other variables
553 c-----
554
555     subroutine init_statevs(statev, re, gambslip, gamtw, epsl0,
556 & epdpl0, energy, tempsv)
557     implicit none
558
559 c     input
560     double precision statev(25)
561
562 c     output
563     double precision re(3,3), gambslip, gamtw(6), epsl0, epdpl0
564     double precision energy, tempsv
565
566     !Read in statev
567     re(1,1) = statev(1)
568     re(1,2) = statev(2)
569     re(1,3) = statev(3)
570     re(2,1) = statev(4)
571     re(2,2) = statev(5)

```

```

571
572     re(2,3) = statev(6)
573     re(3,1) = statev(7)
574     re(3,2) = statev(8)
575     re(3,3) = statev(9)
576     gambslip = statev(10)
577     gamtw(1) = statev(11)
578     gamtw(2) = statev(12)
579     gamtw(3) = statev(13)
580     gamtw(4) = statev(14)
581     gamtw(5) = statev(15)
582     gamtw(6) = statev(16)
583     epsl0 = statev(17)
584     epdsl0 = statev(18)
585     energy = statev(19)
586     tempsv = statev(20)
587     !svs 21-25 do not need to be read in, they are just for output
588
589     return
590 end
591
592 c=====
593 c=====
594 c Calculate depbas, deptw.
595
596 c-----
597
598     subroutine update_dep(actbas, acttw, dgambslip, sbslip, mbas,
599 & dgamtw, ptw, depbas, deptw)
600     implicit none
601
602 c     input
603     logical actbas, acttw
604     double precision dgambslip, sbslip(3), mbas(3)
605     double precision dgamtw(6), ptw(3,3,6)
606
607 c     output
608     double precision depbas, deptw
609
610 c     util
611     double precision dbas(6), dtw(6)
612
613 c     depeff = dsqrt(2/3*dp:dp)*dt = dsqrt(2/3*deltadp:deltadp)
614
615 c     calculate deltaDP for basal slip
616     if (actbas) then
617         dbas(1)=dgambslip*sbslip(1)*mbas(1)
618         dbas(2)=dgambslip*sbslip(2)*mbas(2)
619         dbas(3)=dgambslip*sbslip(3)*mbas(3)
620         dbas(4)=0.5d+0*dgambslip*(sbslip(1)*mbas(2)+sbslip(2)*mbas(1))
621         dbas(5)=0.5d+0*dgambslip*(sbslip(1)*mbas(3)+sbslip(3)*mbas(1))
622         dbas(6)=0.5d+0*dgambslip*(sbslip(2)*mbas(3)+sbslip(3)*mbas(2))
623         depbas = dsqrt(2.0d+0/3.0d+0*(dbas(1)**2+dbas(2)**2+
624 & dbas(3)**2+2.0d+0*(dbas(4)**2+dbas(5)**2+dbas(6)**2)))
625     else
626         depbas = 0.0d+0
627     end if
628
629 c     calculate deltaDP for twinning
630     if (acttw) then
631         dtw(1)=dgamtw(1)*ptw(1,1,1)+dgamtw(2)*ptw(1,1,2)+
632 & dgamtw(3)*ptw(1,1,3)+dgamtw(4)*ptw(1,1,4) +
633 & dgamtw(5)*ptw(1,1,5)+dgamtw(6)*ptw(1,1,6)
634         dtw(2)=dgamtw(1)*ptw(2,2,1)+dgamtw(2)*ptw(2,2,2)+
635 & dgamtw(3)*ptw(2,2,3)+dgamtw(4)*ptw(2,2,4) +
636 & dgamtw(5)*ptw(2,2,5)+dgamtw(6)*ptw(2,2,6)

```

```

636
637      dtw(3)=dgamtw(1)*ptw(3,3,1)+dgamtw(2)*ptw(3,3,2)+
638      & dgamtw(3)*ptw(3,3,3)+dgamtw(4)*ptw(3,3,4) +
639      & dgamtw(5)*ptw(3,3,5)+dgamtw(6)*ptw(3,3,6)
640      dtw(4)=dgamtw(1)*ptw(1,2,1)+dgamtw(2)*ptw(1,2,2)+
641      & dgamtw(3)*ptw(1,2,3)+dgamtw(4)*ptw(1,2,4) +
642      & dgamtw(5)*ptw(1,2,5)+dgamtw(6)*ptw(1,2,6)
643      dtw(5)=dgamtw(1)*ptw(1,3,1)+dgamtw(2)*ptw(1,3,2)+
644      & dgamtw(3)*ptw(1,3,3)+dgamtw(4)*ptw(1,3,4) +
645      & dgamtw(5)*ptw(1,3,5)+dgamtw(6)*ptw(1,3,6)
646      dtw(6)=dgamtw(1)*ptw(2,3,1)+dgamtw(2)*ptw(2,3,2)+
647      & dgamtw(3)*ptw(2,3,3)+dgamtw(4)*ptw(2,3,4) +
648      & dgamtw(5)*ptw(2,3,5)+dgamtw(6)*ptw(2,3,6)
649      deptw = dsqrt(2.0d+0/3.0d+0*(dtw(1)**2+dtw(2)**2+
650      & dtw(3)**2+2.0d+0*(dtw(4)**2+dtw(5)**2+dtw(6)**2)))
651      else
652          deptw = 0.0d+0
653      end if
654
655      return
656  end
657
658
659  C=====
660  C=====
661  c Calculate energy and pressure based volumetric strain
662  C-----
663
664      subroutine eos(eosflag, dfgrd0, dfgrd1, dtime, temp0, rho0,
665      & gam, b0, dbdp, cv, ysbas0, ystw0, yssl, dgambslip, dgamtw,
666      & depl, p, uint, tempsv)
667      implicit none
668
669  c      input
670      integer eosflag
671      double precision dfgrd0(3,3), dfgrd1(3,3), dtime, temp0, rho0
672      double precision gam, b0, dbdp, cv
673      double precision ysbas0, ystw0(6), yssl
674      double precision dgambslip, dgamtw(6), depl
675
676  c      input/output
677      double precision p, uint
678
679  c      output
680      double precision tempsv
681
682  c      util
683      double precision jnew, jold, du, rho, cb, q, p0
684      double precision determinant
685      double precision dtplast
686
687      double precision qc1, qc2, ONE, TWO
688      parameter (qc1=0.00d+0, qc2=0.0d+0, ONE=1.0D+0, TWO=2.0D+0)
689
690  c      determine jacobian, volume jump, density
691      jnew = determinant(dfgrd1)
692      jold = determinant(dfgrd0)
693      du = (jnew-jold)/dtime
694      rho = 2.0d0*rho0/(jnew+jold)
695      p0 = p
696      cb = dsqrt((b0+dbdp*p0)/rho) !Murnaghan eos
697
698  c      artificial viscosity - off in rarefaction
699      if ((jnew.lt.jold).and.(eosflag.eq.2)) then
700          q = rho*(qc1*cB*dabs(du)+qc2**2*du**2)
701      else

```

```

702     q = 0.0d+0
703     end if
704
705     c    Murnaghan eos with constant cv
706     p = (TWO*((b0*(dbdp + (jnew**dbdp - ONE)*(gam*jnew + ONE) -
707 & dbdp*jnew**dbdp*ONE*(gam*(jnew - ONE) + ONE)))/(dbdp*
708 & jnew*(dbdp*ONE)*(dbdp - ONE)) + (gam*((jold - jnew*ONE)*p0 -
709 & ((jnew - jold*ONE)*q + cv*gam*(jnew - ONE)*temp0)*TWO +
710 & TWO*uint))/(TWO))/(gam*jnew - gam*jold*ONE + TWO)
711
712     c    Discretized energy update
713     uint = uint - ONE/TWO*(jnew-jold)*(p+p0+q)
714
715     c    Alter pressure by art visc
716     p = p + q
717
718     c    tempstv
719     tempstv = (-b0*jnew*ONE) + jnew**dbdp*(b0*(dbdp + jnew -
720 & dbdp*jnew*ONE) + dbdp*(dbdp - ONE)*(cv*(gam + ONE -
721 & gam*jnew*ONE)*temp0 + uint)))/(cv*dbdp*jnew*(dbdp*ONE)*
722 & (dbdp - ONE))
723     dtplast = jnew/(rho0*cv)*(ysbas0*dgambslip+ystw0(1)*dgamtw(1)+
724 & ystw0(2)*dgamtw(2)+ystw0(3)*dgamtw(3)+ystw0(4)*dgamtw(4)+
725 & ystw0(5)*dgamtw(5)+ystw0(6)*dgamtw(6)+ysl1*depsl)
726     tempstv = tempstv + dtplast
727
728     return
729     end
730
731
732     C=====
733     C=====
734     c    Based on dgamtwr and acttwsys, assigns dgamtw to original systems
735     C-----
736
737     subroutine calc_full_dgamtw(acttwsys, dgamtwr, dgamtw)
738     implicit none
739
740     c    input
741     integer acttwsys(6)
742     double precision dgamtwr(6)
743
744     c    output
745     double precision dgamtw(6)
746
747     c    util
748     integer full, red, nact
749
750     dgamtw(1) = 0.0d+0
751     dgamtw(2) = 0.0d+0
752     dgamtw(3) = 0.0d+0
753     dgamtw(4) = 0.0d+0
754     dgamtw(5) = 0.0d+0
755     dgamtw(6) = 0.0d+0
756
757     nact = acttwsys(1)+acttwsys(2)+acttwsys(3)+acttwsys(4)+
758 & acttwsys(5)+acttwsys(6)
759
760     if (nact.gt.0) then
761         full = 1 !full notation
762         red = 1 !reduced notation
763     10  if (red.le.nact) then
764         if (acttwsys(full).eq.1) then
765             dgamtw(full) = dgamtwr(red)
766             red = red + 1
767         else

```

```

768         dgamtw(full) = 0.0d+0
769     end if
770     full = full + 1
771     goto 10
772 end if
773 end if
774
775 return
776 end
777
778
779 c=====
780 c=====
781 c Calculate Re = exp(wedt).Re, where in this case wedt=wpdt-wdt
782 c -- note if basal is inactive, or if omega is small, do nothing to re
783 c-----
784     subroutine update_re(actbas, wpdt, wdt, re)
785     implicit none
786
787 c     input
788     logical actbas
789     double precision wpdt(3,3), wdt(3,3)
790
791 c     input/output
792     double precision re(3,3)
793
794 c     util - w is used in place of wedt for shortness
795     double precision w(3,3), wdw(3,3), ch(3,3), reo(3,3)
796     double precision om, small
797
798     small = 1.0d-12
799     if (actbas) then
800
801         w(1,1) = wdt(1,1) - wpdt(1,1)
802         w(1,2) = wdt(1,2) - wpdt(1,2)
803         w(1,3) = wdt(1,3) - wpdt(1,3)
804         w(2,1) = wdt(2,1) - wpdt(2,1)
805         w(2,2) = wdt(2,2) - wpdt(2,2)
806         w(2,3) = wdt(2,3) - wpdt(2,3)
807         w(3,1) = wdt(3,1) - wpdt(3,1)
808         w(3,2) = wdt(3,2) - wpdt(3,2)
809         w(3,3) = wdt(3,3) - wpdt(3,3)
810
811 c     omega = om = sqrt(0.5*w:w)
812     om = dsqrt(0.5d+0*(w(1,1)*w(1,1)+w(1,2)*w(1,2) +
813 & w(1,3)*w(1,3)+w(2,1)*w(2,1)+w(2,2)*w(2,2) + w(2,3)*w(2,3) +
814 & w(3,1)*w(3,1)+w(3,2)*w(3,2)+w(3,3)*w(3,3)))
815     if (om.ge.small) then
816 c     wdw = w.w
817     wdw(1,1)=w(1,1)*w(1,1)+w(1,2)*w(2,1)+w(1,3)*w(3,1)
818     wdw(1,2)=w(1,1)*w(1,2)+w(1,2)*w(2,2)+w(1,3)*w(3,2)
819     wdw(1,3)=w(1,1)*w(1,3)+w(1,2)*w(2,3)+w(1,3)*w(3,3)
820     wdw(2,1)=w(2,1)*w(1,1)+w(2,2)*w(2,1)+w(2,3)*w(3,1)
821     wdw(2,2)=w(2,1)*w(1,2)+w(2,2)*w(2,2)+w(2,3)*w(3,2)
822     wdw(2,3)=w(2,1)*w(1,3)+w(2,2)*w(2,3)+w(2,3)*w(3,3)
823     wdw(3,1)=w(3,1)*w(1,1)+w(3,2)*w(2,1)+w(3,3)*w(3,1)
824     wdw(3,2)=w(3,1)*w(1,2)+w(3,2)*w(2,2)+w(3,3)*w(3,2)
825     wdw(3,3)=w(3,1)*w(1,3)+w(3,2)*w(2,3)+w(3,3)*w(3,3)
826
827 c     exp(w) = id + sin(om)/om*w+(1-cos(om))/om**2(w.w)
828     ch(1,1)=1.0d+0+dsin(om)/om*w(1,1)+
829 & (1.0d+0-dcos(om))/(om*om)*wdw(1,1)
830     ch(1,2)=dsin(om)/om*w(1,2)+(1.0d+0-dcos(om))/(om*om)*wdw(1,2)
831
832     ch(1,3)=dsin(om)/om*w(1,3)+(1.0d+0-dcos(om))/(om*om)*wdw(1,3)
833     ch(2,1)=dsin(om)/om*w(2,1)+(1.0d+0-dcos(om))/(om*om)*wdw(2,1)

```

```

833      ch(2,2)=1.0d+0+dsin(om)/om*w(2,2)+
834      & (1.0d+0-dcos(om))/(om*om)*wdw(2,2)
835      ch(2,3)=dsin(om)/om*w(2,3)+(1.0d+0-dcos(om))/(om*om)*wdw(2,3)
836      ch(3,1)=dsin(om)/om*w(3,1)+(1.0d+0-dcos(om))/(om*om)*wdw(3,1)
837      ch(3,2)=dsin(om)/om*w(3,2)+(1.0d+0-dcos(om))/(om*om)*wdw(3,2)
838      ch(3,3)=1.0d+0+dsin(om)/om*w(3,3)+
839      & (1.0d+0-dcos(om))/(om*om)*wdw(3,3)
840
841      reo(1,1) = re(1,1)
842      reo(1,2) = re(1,2)
843      reo(1,3) = re(1,3)
844      reo(2,1) = re(2,1)
845      reo(2,2) = re(2,2)
846      reo(2,3) = re(2,3)
847      reo(3,1) = re(3,1)
848      reo(3,2) = re(3,2)
849      reo(3,3) = re(3,3)
850
851      c      re = ch.reo
852      re(1,1)=ch(1,1)*reo(1,1)+ch(1,2)*reo(2,1)+ch(1,3)*reo(3,1)
853      re(1,2)=ch(1,1)*reo(1,2)+ch(1,2)*reo(2,2)+ch(1,3)*reo(3,2)
854      re(1,3)=ch(1,1)*reo(1,3)+ch(1,2)*reo(2,3)+ch(1,3)*reo(3,3)
855      re(2,1)=ch(2,1)*reo(1,1)+ch(2,2)*reo(2,1)+ch(2,3)*reo(3,1)
856      re(2,2)=ch(2,1)*reo(1,2)+ch(2,2)*reo(2,2)+ch(2,3)*reo(3,2)
857      re(2,3)=ch(2,1)*reo(1,3)+ch(2,2)*reo(2,3)+ch(2,3)*reo(3,3)
858      re(3,1)=ch(3,1)*reo(1,1)+ch(3,2)*reo(2,1)+ch(3,3)*reo(3,1)
859      re(3,2)=ch(3,1)*reo(1,2)+ch(3,2)*reo(2,2)+ch(3,3)*reo(3,2)
860      re(3,3)=ch(3,1)*reo(1,3)+ch(3,2)*reo(2,3)+ch(3,3)*reo(3,3)
861
862      end if
863  end if
864
865      return
866  end
867
868  C=====
869  C=====
870  c Calculate W^P*dt from basal slip part of L
871  c -- recall mbas = mbslip
872  c -- neglects twinning since this is just plain wrong
873  c-----
874
875      subroutine calc_wpdt(sbs, mbs, dgambslip, wpdt)
876      implicit none
877
878      c      input
879      double precision sbs(3), mbs(3), dgambslip
880      c      output
881      double precision wpdt(3,3)
882
883      wpdt(1,1) = 0.0d+0
884      wpdt(2,2) = 0.0d+0
885      wpdt(3,3) = 0.0d+0
886      wpdt(1,2) = 0.5d+0*dgambslip*(sbs(1)*mbs(2)-sbs(2)*mbs(1))
887      wpdt(2,1) = 0.5d+0*dgambslip*(sbs(2)*mbs(1)-sbs(1)*mbs(2))
888      wpdt(1,3) = 0.5d+0*dgambslip*(sbs(1)*mbs(3)-sbs(3)*mbs(1))
889      wpdt(3,1) = 0.5d+0*dgambslip*(sbs(3)*mbs(1)-sbs(1)*mbs(3))
890      wpdt(2,3) = 0.5d+0*dgambslip*(sbs(2)*mbs(3)-sbs(3)*mbs(2))
891      wpdt(3,2) = 0.5d+0*dgambslip*(sbs(3)*mbs(2)-sbs(2)*mbs(3))
892
893      return
894  end
895
896  C=====
897  C=====

```

```

898 c Given drot in abaqus, calculate dW
899 c -- FROM HUGHES AND WINGET, W=2*(R-1)*(R+1)^(-1)
900 c-----
901
902 SUBROUTINE calc_wdt_abq(DROT, W)
903 IMPLICIT NONE
904
905 c input
906 DOUBLE PRECISION DROT(3,3)
907 c output
908 DOUBLE PRECISION W(3,3)
909 c util
910 DOUBLE PRECISION R1(3,3), R2(3,3)
911
912 R1(1,1)=DROT(1,1)+1.0D+0
913 R1(1,2)=DROT(1,2)
914 R1(1,3)=DROT(1,3)
915 R1(2,1)=DROT(2,1)
916 R1(2,2)=DROT(2,2)+1.0D+0
917 R1(2,3)=DROT(2,3)
918 R1(3,1)=DROT(3,1)
919 R1(3,2)=DROT(3,2)
920 R1(3,3)=DROT(3,3)+1.0D+0
921 CALL calc_inverse_3x3(R1,R2)
922 R1(1,1)=DROT(1,1)-1.0D+0
923 R1(1,2)=DROT(1,2)
924 R1(1,3)=DROT(1,3)
925 R1(2,1)=DROT(2,1)
926 R1(2,2)=DROT(2,2)-1.0D+0
927 R1(2,3)=DROT(2,3)
928 R1(3,1)=DROT(3,1)
929 R1(3,2)=DROT(3,2)
930 R1(3,3)=DROT(3,3)-1.0D+0
931 C
932 W(1,1)=R1(1,1)*R2(1,1)+R1(1,2)*R2(2,1)+R1(1,3)*R2(3,1)
933 W(1,2)=R1(1,1)*R2(1,2)+R1(1,2)*R2(2,2)+R1(1,3)*R2(3,2)
934 W(1,3)=R1(1,1)*R2(1,3)+R1(1,2)*R2(2,3)+R1(1,3)*R2(3,3)
935 W(2,1)=R1(2,1)*R2(1,1)+R1(2,2)*R2(2,1)+R1(2,3)*R2(3,1)
936 W(2,2)=R1(2,1)*R2(1,2)+R1(2,2)*R2(2,2)+R1(2,3)*R2(3,2)
937 W(2,3)=R1(2,1)*R2(1,3)+R1(2,2)*R2(2,3)+R1(2,3)*R2(3,3)
938 W(3,1)=R1(3,1)*R2(1,1)+R1(3,2)*R2(2,1)+R1(3,3)*R2(3,1)
939 W(3,2)=R1(3,1)*R2(1,2)+R1(3,2)*R2(2,2)+R1(3,3)*R2(3,2)
940 W(3,3)=R1(3,1)*R2(1,3)+R1(3,2)*R2(2,3)+R1(3,3)*R2(3,3)
941
942 return
943 end
944
945 C=====
946 C=====
947 c Calculates inverse of a 3x3 matrix
948 c-----
949
950 subroutine calc_inverse_3x3(a,b)
951 ! Calculate the inverse of a 3 x 3 matrix.
952
953 implicit none
954
955 double precision a(3,3), b(3,3)
956 double precision d, small
957 integer i, j
958
959 small = 1d-12
960
961 b(1,1) = a(2,2) * a(3,3) - a(3,2) * a(2,3)
962 b(1,2) = a(3,2) * a(1,3) - a(1,2) * a(3,3)

```

```

963      b(1,3) = a(1,2) * a(2,3) - a(2,2) * a(1,3)
964      b(2,1) = a(3,1) * a(2,3) - a(2,1) * a(3,3)
965      b(2,2) = a(1,1) * a(3,3) - a(3,1) * a(1,3)
966      b(2,3) = a(2,1) * a(1,3) - a(1,1) * a(2,3)
967      b(3,1) = a(2,1) * a(3,2) - a(3,1) * a(2,2)
968      b(3,2) = a(3,1) * a(1,2) - a(1,1) * a(3,2)
969      b(3,3) = a(1,1) * a(2,2) - a(2,1) * a(1,2)
970
971      d = a(1,1) * b(1,1) + a(1,2) * b(2,1) + a(1,3) * b(3,1)
972
973      if (abs(d).le.small) then
974          print*, 'Took determinant in inverse, smaller than 1e-12'
975      end if
976
977      DO i = 1,3
978          DO j = 1,3
979              b(i,j) = b(i,j) / d
980          END DO
981      END DO
982
983      RETURN
984      END
985
986      C=====
987      C=====
988      c Find out if any mechanisms need to be reactivated. If so,
989      c   reactivate mechanisms according to which has the highest
990      c   overstress (tau - ys), and reactivate that one. Return
991      c   reactivate = .true. if any need to be, .false. otherwise
992      c-----
993
994      subroutine reactivate_mechanisms(actbas, acttw, actsl, acttwsys,
995      & captw, sigd, pbas, ptw, ysbas, ystw, yssl, epdsl, reactTOL,
996      & deactTOL, mu, reactivate, deactivatesl)
997
998      implicit none
999
1000     c input/output
1001     logical actbas, acttw, actsl
1002     integer acttwsys(6)
1003     logical captw
1004
1005     c input
1006     double precision sigd(3,3), pbas(3,3), ptw(3,6,6)
1007     double precision ysbas, ystw(6), yssl, epdsl
1008     double precision reactTOL, deactTOL, mu
1009
1010     c output
1011     logical reactivate, deactivatesl
1012
1013     c util
1014     double precision taubas, tautw(6), sigvm, maxdiff
1015     double precision stressdiff
1016     integer i, maxdiffint
1017
1018     call calc_taus(sigd, pbas, ptw, taubas, tautw, sigvm)
1019
1020     c maxdiffint is 1-6 for tw, 7 for bas, 8 for sl
1021     c if the system would be active based on stress but is inactive,
1022     c   record the difference, and if it's the largest diff, record
1023     c   its maxdiffint. also, throw a warning if the system is active
1024     c   but shouldn't be
1025
1026     c twinning
1027     maxdiff = 0.0d+0

```



```

1028     maxdiffint = 0
1029     reactivate = .false.
1030
1031     if (.not.captw) then
1032         do i=1,6
1033             stressdiff = tautw(i)-ystw(i)
1034             c      reactivation
1035             if ((stressdiff/mu.ge.reactTOL).and.(acttwsys(i).eq.0)) then
1036                 if (stressdiff.gt.maxdiff) then
1037                     maxdiff = stressdiff
1038                     maxdiffint = i
1039                 end if
1040             end if
1041             c      deactivation identification
1042             if ((stressdiff/mu.le.-deactTOL).and.(acttwsys(i).eq.1)) then
1043                 print*, 'ERROR: TWIN SYSTEM ', i, ' IS ACTV BUT SHOULDNT BE'
1044                 print*, '-- TAU: ', tautw(i), '  YS: ', ystw(i)
1045                 print*, '-- ACTTWSYS, ', acttwsys
1046                 print*, 'allTau: ', tautw
1047             end if
1048         end do
1049     end if
1050
1051     c      basal slip
1052     stressdiff = taubas - ysbas
1053     c      reactivation
1054     if ((stressdiff/mu.ge.reactTOL).and.(.not.actbas)) then
1055         if (stressdiff.gt.maxdiff) then
1056             maxdiff = stressdiff
1057             maxdiffint = 7
1058         end if
1059     end if
1060     c      deactivation identification
1061     if ((stressdiff/mu.le.-deactTOL).and.(actbas)) then
1062         print*, 'ERROR: BASAL SLIP IS ACTIVE BUT SHOULDNT BE'
1063         print*, '-- TAU: ', taubas, '  YS: ', ysbas
1064     end if
1065
1066     c      nonbasal slip
1067     stressdiff = sigvm - yssl
1068     c      reactivation
1069     if ((stressdiff/mu.ge.reactTOL).and.(.not.actsl)) then
1070         if (stressdiff.gt.maxdiff) then
1071             maxdiff = stressdiff
1072             maxdiffint = 8
1073         end if
1074     end if
1075     c      deactivation
1076     if ((stressdiff/mu.le.-deactTOL).and.(actsl).and.(epdsl.gt.1d-8))
1077     & then
1078         c      print*, 'ERROR: NONBASAL SLIP IS ACTIVE BUT SHOULDNT BE'
1079         c      print*, '-- SIGVM: ', sigvm, '  YS: ', yssl
1080         actsl = .false.
1081         epdsl = 0.0d+0
1082         deactivatesl = .true.
1083         reactivate = .true.
1084     end if
1085
1086     c      reactivation
1087     if (maxdiffint.gt.0) then
1088         reactivate = .true.
1089         if (maxdiffint.le.6) then !twinning
1090
1091             acttw = .true.
1092             acttwsys(maxdiffint) = 1
1093         elseif (maxdiffint.eq.7) then !basal

```

```

1093         actbas = .true.
1094     else !nonbasal
1095         reactivate = .false.
1096 c         actsl = .true.
1097         print*, 'WARNING: NONBASAL SLIP SHOULD BE REACTIVATED BUT WONT'
1098         print*, 'sigvm: ', sigvm, ' ys: ', yssl
1099     end if
1100 end if
1101
1102     return
1103 end
1104
1105 C=====
1106 C=====
1107 c Calculate the amount of non-basal slip that would cause a mode
1108 c to become inactive. Note that full representation of gtw, ystw
1109 c are fed in, and not reduced form, but depftw is returned for
1110 c the reduced form.
1111 C-----
1112
1113     subroutine calc_epsflip(actbas, acttw, hmix0, gbas, ysbas, gtw,
1114 &     ystw, yssl, mu, acttwsys, epfbas, depftw)
1115
1116 c     input
1117     logical actbas, acttw
1118     double precision hmix0(6)
1119     double precision gbas, ysbas
1120     double precision gtw(6), ystw(6)
1121     double precision yssl
1122     double precision mu
1123     integer acttwsys(6)
1124
1125 c     output
1126     double precision epfbas, depftw(6)
1127
1128 c     util
1129     integer i,j,nact
1130     double precision num, denom
1131     double precision mhin(6,6), hmix(6), gtwr(6), ystwr(6)
1132
1133 c     evaluate basal slip component
1134     if (actbas) then
1135         epfbas = yssl/(3.0d+0*mu*ysbas)*(gbas-ysbas)
1136     else
1137         epfbas = 1.0d+5
1138     end if
1139
1140 c     evaluate twining component - note different if basal is active
1141     if (acttw) then
1142 c         define mhin, gtwr, ystwr based on acttwsys
1143         call calc_reduced_hmix(hmix0, acttwsys, hmix)
1144         call calc_reduced_gys(gtw, ystw, acttwsys, gtwr, ystwr)
1145         nact = acttwsys(1) + acttwsys(2) + acttwsys(3) + acttwsys(4) +
1146 &         acttwsys(5) + acttwsys(6)
1147
1148         if (actbas) then
1149 c             define hmix based on acttwsys
1150             call calc_minv(acttwsys, mhin)
1151             do i=1,nact
1152                 num = 0.0d+0
1153                 denom = 0.0d+0
1154                 do j=1,nact
1155                     num = num + mhin(i,j)*(gtwr(j)-hmix(j)*gbas
1156 &                     - (ystwr(j) - hmix(j)*ysbas))
1157                     denom = denom+mhin(i,j)*(ystwr(j)-hmix(j)*ysbas)

```

```

1158         end do
1159         depftw(i) = yssl/(3.0d+0*mu)*num/denom
1160
1161 c         note if depftwin is greater than epfbas, above is invalid
1162         if (depftw(i).ge.epfbas) then
1163             num = 0.0d+0
1164             denom = 0.0d+0
1165             do j=1,nact
1166                 num = num + mhinu(i,j)*(gtwr(j)-ystwr(j))
1167                 denom = denom + mhinu(i,j)*(ystwr(j))
1168             end do
1169             depftw(i) = yssl/(3.0d+0*mu)*num/denom
1170         end if
1171     end do
1172
1173 c     evaluate twinning w/o basal slip
1174     else
1175         do i=1,nact
1176             num = 0.0d+0
1177             denom = 0.0d+0
1178             do j=1,nact
1179                 num = num + mhinu(i,j)*(gtwr(j)-ystwr(j))
1180                 denom = denom + mhinu(i,j)*(ystwr(j))
1181             end do
1182             depftw(i) = yssl/(3.0d+0*mu)*num/denom
1183         end do
1184     end if
1185
1186 c     if twinning is inactive
1187     else
1188         do i=1,6
1189             depftw(i) = 1.0d+5
1190         end do
1191     end if
1192
1193     return
1194 end
1195
1196 C=====
1197 C=====
1198 c If stress is not enough to cause slip with epdsl = 0, kill slip. Do
1199 c so by changing actsl from .true. to .false.
1200 C-----
1201
1202     subroutine suppress_slip_query(actbas, acttw, acttwsys, dgambas,
1203 &    pbas, dgamtw, phtw, sigdt, mu, yssl0, actsl)
1204
1205     implicit none
1206 c     input
1207     logical actbas, acttw
1208     integer acttwsys(6)
1209     double precision dgambas, pbas(3,3), dgamtw(6), phtw(3,3,6)
1210     double precision sigdt(3,3) !deviatoric trial stress
1211     double precision mu, yssl0
1212
1213 c     input/output
1214     logical actsl
1215
1216 c     util
1217     integer i,j,k,nact
1218     double precision sigd(3,3), phtwr(3,3,6), sigvm
1219
1220 c     if neither twinning or basal is active, skip this function
1221     if ((actbas).or.(acttw)) then
1222         nact = acttwsys(1) + acttwsys(2) + acttwsys(3) + acttwsys(4) +

```

```

1223      &   acttwsys(5) + acttwsys(6)
1224      call calc_reduced_phtw(phtw, acttwsys, phtwr)
1225      do i=1,3
1226        do j=1,3
1227          sigd(i,j) = sigdt(i,j) - 2.0d+0*mu*dgambas*pbas(i,j)
1228          do k=1,nact
1229            sigd(i,j) = sigd(i,j) - 2.0d+0*mu*dgamtw(k)*phtwr(i,j,k)
1230          end do
1231        end do
1232      end do
1233
1234      sigvm = dsqrt(3.0d+0/2.0d+0*(sigd(1,1)**2+
1235      &   sigd(2,2)**2+sigd(3,3)**2+2.0d+0*(sigd(1,2)**2 +
1236      &   sigd(1,3)**2 + sigd(2,3)**2))
1237
1238      c   if stress with epsl = 0 implies no j2 slip occurs, kill slip
1239      if (sigvm.lt.yssl0) then
1240        actsl = .false.
1241      end if
1242
1243      end if
1244
1245      return
1246      end
1247
1248
1249      C=====
1250      C=====
1251      c   Determines if basal slip or twinning should be deactivated based
1252      c   on depslmax, which is the maximum slip that occurs if it is the
1253      c   only deformation mechanism that is active
1254      C-----
1255
1256      subroutine deactivate_bastw_fromslip(actbas, acttw, acttwsys,
1257      &   hmix0, gbas, ysbas, gtw, ystw, yssl0, mu, depslmax)
1258
1259      implicit none
1260
1261      c   input/output
1262      logical actbas, acttw
1263      integer acttwsys(6)
1264
1265      c   input
1266      double precision hmix0(6), gbas, ysbas, gtw(6), ystw(6), yssl0
1267      double precision mu
1268      double precision depslmax !max plast strain for slip only
1269
1270      c   util
1271      logical elimsys
1272      double precision depfbas, depftw(6), depfmin
1273      integer i,j,nact
1274
1275      c   see if anything needs to be eliminated by finding the smallest
1276      c   ep that switches signs, and see if its less than depslmax
1277      elimsys = .true.
1278      21 if ((elimsys).and.(actbas.or.acttw)) then
1279
1280      c   calculate deps so that sign flips, twin is reduced form
1281      call calc_epsflip(actbas, acttw, hmix0, gbas, ysbas, gtw,
1282      &   ystw, yssl0, mu, acttwsys, depfbas, depftw)
1283
1284      c   common initializations
1285      depfmin = 1.0d+5
1286      nact = acttwsys(1) + acttwsys(2) + acttwsys(3) + acttwsys(4)
1287      &   + acttwsys(5) + acttwsys(6)

```

```

1288         if (nact.eq.0) then
1289             acttw = .false.
1290         end if
1291
1292 c       determine reduced index of minimum epf, 0 is for basal slip
1293         do i=1,nact
1294             if (depftw(i).le.depftwmin) then
1295                 j=i
1296                 depftwmin = depftw(i)
1297             end if
1298         end do
1299         if (depftbas.le.depftwmin) then
1300             j=0
1301             depftwmin = depftbas
1302         end if
1303
1304 c       if true, eliminate a system and start over
1305         if (depftwmin.le.depslmax) then
1306             if (j.eq.0) then !basal slip
1307                 actbas = .false.
1308             else !twinning
1309                 call remove_acttwsys_entry(acttwsys, j)
1310                 nact = acttwsys(1) + acttwsys(2) + acttwsys(3) + acttwsys(4)
1311 &               + acttwsys(5) + acttwsys(6)
1312 c             if (nact.eq.0) then
1313 c                 print*, 'deleted all twin modes in slip step'
1314 c             end if
1315             end if
1316         else
1317             elimsys = .false.
1318         end if
1319         go to 21
1320     end if
1321
1322     return
1323 end
1324
1325
1326 C=====
1327 C=====
1328 c Calculate epd, von mises stress, and a logical of activity for slip
1329 C-----
1330
1331     subroutine calc_epdsl(actbas, acttw, acttwsys, hmix0, yssl0,
1332 &   gbas, ysbas, pbas, gambslip, gtw, ystw, gamtw, phtw, epsl0, gsl,
1333 &   hsltype, hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8,
1334 &   tempmelt, qslbas, qsltw, stressd0, mu, temp, dt, epdslprev,
1335 &   epdslmax, reactivate, epdsl, nexit)
1336
1337     implicit none
1338
1339 c   input/output
1340     logical actbas, acttw
1341     integer acttwsys(6)
1342
1343 c   input
1344 c   - basal slip and twinning
1345     double precision hmix0(6), yssl0, gbas, ysbas, pbas(3,3), gambslip
1346
1347     double precision gtw(6), ystw(6), gamtw(6)
1348     double precision phtw(3,3,6)
1349 c   - slip - params and isvs
1350     double precision epsl0, gsl
1351     integer hsltype
1352     double precision hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8

```

```

1353     double precision qslbas, qsltw, tempmelt
1354 c     - general
1355     double precision stressd0(3,3), mu, temp
1356 c     - solution related
1357     double precision dt, epdslprev, epdslmax
1358     logical reactivate
1359
1360 c     output
1361     double precision epdsl
1362     integer nexit
1363
1364 c     util - solution related things
1365     logical bis
1366     double precision depslmax
1367     double precision FTOL, RTOL
1368     double precision X1, X2, F, FL, FH, DF, XL, XH, DX, DXOLD
1369     integer MAXIT, J
1370 c     - initial parameters
1371     double precision a,b,c, tempvar
1372     double precision amat(3,3), bmat(3,3)
1373
1374     DATA FTOL,RTOL/1.D-10,1.D-10/
1375     MAXIT = 100
1376     nexit = 0
1377
1378 c     changes actbas, acttw, and acttwsys based on if slip will cause
1379 c     any of the deformation modes to deactivate
1380 c     don't do this if reactivate is true
1381     depslmax = epdslmax*dt
1382     if (.not.reactivate) then
1383         call deactivate_bastw_fromslip(actbas, acttw, acttwsys,
1384 &     hmix0, gbas, ysbas, gtw, ystw, yssl0, mu, depslmax)
1385     end if
1386
1387 c     check if the step should be slip only, and if so, exit
1388     if ((.not.actbas).and(.not.acttw)) then
1389         epdsl = epdslmax
1390         nexit = 0
1391         return
1392     end if
1393
1394 c     do initializations that will be used throughout
1395     call form_abc(stressd0, actbas, gbas, pbas, ysbas, acttw,
1396 &     acttwsys, gtw, ystw, hmix0, phtw, amat, bmat, a, b, c)
1397
1398 c     initial bisection check over strain rates of interest
1399     X1=hs18
1400     X2=epdslmax
1401
1402     bis=.true.
1403     call ksr(bis, X1, a, b, c, gsl, mu, dt,gambslip,gamtw,epsl0, temp,
1404 &     tempmelt, hsltype, hsl1,hs12,hs13,hs14, hsl5, hsl6, hsl7, hsl8,
1405 &     qslbas, qsltw, FL, DF)
1406
1407 c     if FL is positive, no slip will happen
1408     if (FL.ge.0.0D+0) then
1409         epdsl = 0.0d+0
1410         return
1411     end if
1412
1413     call ksr(bis, X2, a, b, c, gsl, mu, dt,gambslip,gamtw,epsl0, temp,
1414 &     tempmelt, hsltype, hsl1,hs12,hs13,hs14, hsl5, hsl6, hsl7, hsl8,
1415 &     qslbas, qsltw, FH, DF)
1416
1417     IF(DABS(FL) .LT. FTOL)THEN
1418         epdsl=X1

```

```

1419         nexit = 1
1420         RETURN
1421     END IF
1422     IF(DABS(FH) .LT. FTOL)THEN
1423         epdsl=X2
1424         nexit = 1
1425         RETURN
1426     END IF
1427
1428 C     SLIP WAS SUPPRESSED BY TWINNING AND BASAL SLIP
1429     IF((FL.GT.0.d0.AND.FH.GT.0.d0).OR.
1430 & (FL.LT.0.0.AND.FH.LT.0.d0)) THEN
1431 C         WRITE(6,19)X1,FL,X2,FH
1432 C 19    FORMAT(' SOLUTION NOT BOUNDED',4G12.5)
1433 C         print*, 'EPDSLPREV: ', epdslprev
1434 C         print*, 'EPDSLMAX: ', epdslmax
1435 C         print*, 'EPDSLMIN: ', hsl8
1436 C         print*, 'acttw: ', acttw
1437 C         print*, 'actbas: ', actbas
1438 C         print*, 'A: ', a
1439 C         print*, 'B: ', b
1440 C         print*, 'C: ', c
1441         nexit = 2
1442         epdsl = 0.0d+0
1443         RETURN
1444     END IF
1445
1446 C     associate high and low of strain rate to high and low of function
1447     IF(FL .LT. 0.0d+0)THEN
1448         XL=X1
1449         XH=X2
1450     ELSE
1451         XH=X1
1452         XL=X2
1453     ENDIF
1454
1455     epdsl = epdslprev
1456     DXOLD=DABS(X2-X1)
1457     DX=DXOLD
1458     bis = .false.
1459
1460     call ksr(bis, epdsl,a,b,c,gsl,mu,dt,gambslip,gamtw,eps10, temp,
1461 & tempmelt, hsltype, hsl1,hsl2,hsl3,hsl4, hsl5, hsl6, hsl7, hsl8,
1462 & qslbas, qsltw, F, DF)
1463 C
1464 C --- BISECT IF SOLUTION EXCEEDS LIMIT OR IF SLOW CONVERGENCE
1465 C     OTHERWISE USE NEWTON ITERATION
1466 C --- CONVERGENCE CHECKS ON BOTH STRAIN RATE AND NORMALIZED FUNCTION
1467 C
1468     DO 10 J=1,MAXIT
1469 C
1470         IF(((epdsl-XH)*DF-F)*((epdsl-XL)*DF-F) .GE. 0.0d+0 .OR.
1471 * DABS(2.0d+0*F) .GT. DABS(DXOLD*DF) ) THEN
1472             DXOLD=DX
1473             DX=0.5d+0*(XH-XL)
1474             epdsl=XL+2.0d+0/3.0d+0*DX
1475             IF(DABS(XL-epdsl)*dt .LT. RTOL .AND.
1476 & DABS(F) .LT. FTOL)RETURN
1477         ELSE
1478             DXOLD=DX
1479             DX=F/DF
1480             tempvar=epdsl
1481             epdsl=epdsl-DX
1482             IF(DABS(tempvar-epdsl)*dt .LT. RTOL .AND.
1483 & DABS(F) .LT. FTOL)RETURN
1484         ENDIF

```

```

1485 C
1486 C --- GET FUNCTION AND SLOPE FOR NEXT ITERATION
1487 C
1488     call ksr(bis, epdsl,a,b,c,gs1,mu,dt,gambslip,gamtw,eps10, temp,
1489 & tempmelt, hsltype, hsl1,hsl2,hsl3,hsl4, hsl5, hsl6, hsl7, hsl8,
1490 & qslbas, qsltw, F, DF)
1491
1492     IF(DABS(DX) .LT. RTOL .AND. DABS(F) .LT. FTOL) RETURN
1493 C
1494     IF(F .LT. 0.0d+0) THEN
1495         XL=epdsl
1496     ELSE
1497         XH=epdsl
1498     ENDIF
1499 10 CONTINUE
1500 C
1501 C --- CUT TIME STEP IF NO CONVERGENCE
1502 C
1503 c     NFAIL=.TRUE.
1504 print*, 'EDOT SOLUTION DID NOT CONVERGE IN 100 STEPS'
1505 nexit = 3
1506
1507 return
1508 end
1509
1510 C=====
1511 C=====
1512 c Form constants to be used in state and deriv equations for epdsl
1513 C-----
1514
1515     subroutine ksr(bis, edot, a, b, c, sigt, mu, dt,
1516 & gambslip, gamtw, eps10, temp, tempmelt, hsltype,
1517 & hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8,
1518 & qslbas, qsltw, f, df)
1519
1520     implicit none
1521
1522 c     input
1523 logical bis
1524 double precision edot
1525 c     - parameters for loading and interaction
1526 double precision a,b,c, sigt, mu, dt
1527 c     - used in strength call only
1528 double precision gambslip, gamtw, eps10, temp, tempmelt
1529 integer hsltype
1530 dimension gamtw(6)
1531 double precision hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8
1532 double precision qslbas, qsltw
1533
1534 c     output
1535 double precision f, df
1536
1537 c     util - temporary var1, 2, von mises stress, dvm / dedot
1538 double precision d, dd, sigvm, dsded
1539 double precision eps1
1540
1541     eps1 = eps10 + edot*dt
1542
1543     call calc_str_sl(bis, gambslip, gamtw, eps1, edot, temp,
1544 & tempmelt,hsltype, hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8,
1545 & qslbas, qsltw, sigvm, dsded)
1546
1547     d = mu*edot*dt
1548     dd = mu*dt
1549 c     check - all units stress**2, then div by stress**2

```



```

1550      f = sigvm**2+6.0d+0*sigvm*d+9.0d+0*d**2-(a+b+c)
1551      & - 3.0d+0*d/sigvm*(b+2.0d+0*c)-9.0d+0*c*d**2/sigvm**2
1552      f = f / sigt**2
1553
1554      if (bis) return
1555
1556      c      check - all units stress**2*time, then div by stress**2
1557      df = 2.0d+0*dsded*sigvm
1558      & + 6.0d+0*dd*(edot*dsded+sigvm)
1559      & + 18.0d+0*d*dd
1560      & - 3.0d+0*dd*(b+2.0d+0*c)*(sigvm-edot*dsded) / sigvm**2
1561      & - 18.0d+0*c*dd**2*(sigvm*edot-edot**2*dsded)/sigvm**3
1562
1563      df = df / sigt**2
1564
1565      return
1566      end
1567
1568      C=====
1569      C=====
1570      c      Calculate epd, von mises stress for nb slip in absence of
1571      c      basal slip and twin increments (still uses init vals to calc sl str)
1572      c-----
1573
1574      subroutine calc_epdsl_ez(gambslip, gamtw, epsl0, gsl,
1575      & hsltype, hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8,
1576      & tempmelt, qslbas, qsltw, stressd0, mu, temp, dt, epdslmax,
1577      & epdsl)
1578
1579      implicit none
1580
1581      c      input
1582      double precision gambslip, gamtw(6)
1583      double precision epsl0, gsl
1584      c      - slip params and thermodynamic vars
1585      integer hsltype
1586      double precision hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8
1587      double precision qslbas, qsltw, tempmelt
1588      c      - general
1589      double precision stressd0(3,3), mu, temp
1590      c      - solution related
1591      double precision dt, epdslmax
1592
1593      c      output
1594      double precision epdsl
1595
1596      c      util - solution related things
1597      logical bis
1598      double precision tempvar
1599      double precision FTOL, RTOL
1600      double precision X1, X2, F, FL, FH, DF, XL, XH, DX, DXOLD, FACT
1601      integer MAXIT, J
1602
1603      DATA FTOL,RTOL/1.D-12,1.D-12/
1604      MAXIT = 100
1605
1606      c      initial bisection check over strain rates of interest
1607      X1=hsl8
1608
1609      X2=epdslmax
1610      FACT=3.0d+0*mu*dt
1611      bis=.true.
1612
1613      call ksr_ez(bis, X1, gsl, FACT, dt, gambslip, gamtw, epsl0, temp,
1614      & tempmelt, hsltype, hsl1,hsl2,hsl3,hsl4, hsl5, hsl6, hsl7, hsl8,
1615      & qslbas, qsltw, FL, DF)

```

```

1615
1616 c    if FL is positive, no slip will happen
1617     if (FL.ge.0.0d+0) then
1618         epdsl = 0.0d+0
1619         return
1620     end if
1621
1622     call ksr_ez(bis, X2, gsl, FACT, dt, gambslip,gamtw,epsl0, temp,
1623 & tempmelt, hsltype, hsl1,hs12,hs13,hs14, hsl5, hsl6, hsl7, hsl8,
1624 & qslbas, qsltw, FH, DF)
1625
1626     IF(DABS(FL) .LT. FTOL)THEN
1627         epdsl=X1
1628         RETURN
1629     END IF
1630     IF(DABS(FH) .LT. FTOL)THEN
1631         epdsl=X2
1632         RETURN
1633     END IF
1634 C
1635     IF((FL.GT.0.d0.AND.FH.GT.0.d0).OR.
1636 & (FL.LT.0.0.AND.FH.LT.0.d0)) THEN
1637         WRITE(6,19)X1,FL,X2,FH
1638 19    FORMAT(' SOLUTION NOT BOUNDED IN EZ EVAL',4G12.5)
1639         epdsl = 0.0d+0
1640         RETURN
1641     END IF
1642
1643 c    associate high and low of strain rate to high and low of function
1644     IF(FL .LT. 0.0d+0)THEN
1645         XL=X1
1646         XH=X2
1647     ELSE
1648         XH=X1
1649         XL=X2
1650     ENDIF
1651
1652     epdsl = 0.5d+0*(X1+X2)
1653     DXOLD=DABS(X2-X1)
1654     DX=DXOLD
1655     bis = .false.
1656
1657     call ksr_ez(bis, epdsl, gsl, FACT, dt, gambslip,gamtw,epsl0, temp,
1658 & tempmelt, hsltype, hsl1,hs12,hs13,hs14, hsl5, hsl6, hsl7, hsl8,
1659 & qslbas, qsltw, F, DF)
1660 C
1661 C --- BISECT IF SOLUTION EXCEEDS LIMIT OR IF SLOW CONVERGENCE
1662 C    OTHERWISE USE NEWTON ITERATION
1663 C --- CONVERGENCE CHECKS ON BOTH STRAIN RATE AND NORMALIZED FUNCTION
1664 C
1665     DO 10 J=1,MAXIT
1666 C
1667     IF(((epdsl-XH)*DF-F)*((epdsl-XL)*DF-F) .GE. 0.0d+0 .OR.
1668 *    DABS(2.0d+0*F) .GT. DABS(DXOLD*DF) ) THEN
1669         DXOLD=DX
1670         DX=0.5d+0*(XH-XL)
1671         epdsl=XL+2.0d+0/3.0d+0*DX
1672         IF(DABS(XL-epdsl)*dt .LT. RTOL .AND.
1673 &    DABS(F) .LT. FTOL)RETURN
1674     ELSE
1675         DXOLD=DX
1676         DX=F/DF
1677         tempvar=epdsl
1678         epdsl=epdsl-DX
1679         IF(DABS(tempvar-epdsl)*dt .LT. RTOL .AND.

```

```

1680      &      DABS(F) .LT. FTOL)RETURN
1681      ENDIF
1682  C
1683  C --- GET FUNCTION AND SLOPE FOR NEXT ITERATION
1684  C
1685      call ksr_ez(bis, epdsl, gsl, FACT, dt,gambslip,gamtw,epsl0,temp,
1686      & tempmelt, hsltype, hsl1,hsl2,hsl3,hsl4, hsl5, hsl6, hsl7, hsl8,
1687      & qslbas, qsltw, F, DF)
1688
1689      IF(DABS(DX) .LT. RTOL .AND. DABS(F) .LT. FTOL) RETURN
1690  C
1691      IF(F .LT. 0.0d+0) THEN
1692          XL=epdsl
1693      ELSE
1694          XH=epdsl
1695      ENDIF
1696      10 CONTINUE
1697  C
1698  C --- CUT TIME STEP IF NO CONVERGENCE
1699  C
1700  C      NFAIL=.TRUE.
1701      print*, 'EDOT SOLUTION DID NOT CONVERGE IN 100 STEPS IN EZ'
1702
1703      return
1704      end
1705
1706  C=====
1707  C=====
1708  c Form constants to be used in state and deriv equations for epsdsl
1709  C-----
1710
1711      subroutine ksr_ez(bis, edot, sigt, fact, dt,
1712      & gambslip, gamtw, epsl0, temp, tempmelt, hsltype,
1713      & hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8,
1714      & qslbas, qsltw, f, df)
1715
1716      implicit none
1717
1718  c      input
1719      logical bis
1720      double precision edot, sigt
1721  c      - parameters for loading and interaction
1722      double precision fact, dt
1723  c      - used in strength call only
1724      double precision gambslip, gamtw, epsl0, temp, tempmelt
1725      integer hsltype
1726      dimension gamtw(6)
1727      double precision hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8
1728      double precision qslbas, qsltw
1729
1730  c      output
1731      double precision f, df
1732
1733  c      util - temporary var1, 2, von mises stress, dvm / dedot
1734      double precision sigvm, dsded
1735      double precision epsl
1736
1737      epsl = epsl0 + edot*dt
1738
1739      call calc_str_sl(bis, gambslip, gamtw, epsl, edot, temp,
1740      & tempmelt,hsltype, hsl1, hsl2, hsl3, hsl4, hsl5, hsl6, hsl7, hsl8,
1741      & qslbas, qsltw, sigvm, dsded)
1742
1743      f = (sigvm + fact*edot-sigt)/sigt
1744

```

```

1745     if (bis) return
1746
1747     df = (dsded + fact - sigt) / sigt
1748
1749     return
1750 end
1751
1752 c=====
1753 c=====
1754 c Form constants to be used in state and deriv equations for epsdsl
1755 c-----
1756
1757     subroutine form_abc(sigdt, actbas, gbas, pbas, ysbas, acttw,
1758 & acttwsys, gtw, ystw, hmix0, phtw, amat, bmat, a, b, c)
1759
1760     implicit none
1761
1762 c input
1763     double precision sigdt(3,3), gbas, ysbas, pbas(3,3)
1764     double precision gtw(6), ystw(6), hmix0(6), phtw(3,3,6)
1765     logical actbas, acttw
1766     integer acttwsys(6)
1767
1768 c output
1769     double precision amat(3,3), bmat(3,3)
1770     double precision a,b,c
1771
1772 c util
1773     integer i,j,k,l,ntw
1774     double precision hmixr(6),mhinv(6,6),gtwr(6),ystwr(6),phtwr(3,3,6)
1775
1776 c use reduced representation
1777     call calc_reduced_hmix(hmix0, acttwsys, hmixr)
1778     call calc_minv(acttwsys, mhinv)
1779     call calc_reduced_gys(gtw, ystw, acttwsys, gtwr, ystwr)
1780     call calc_reduced_phtw(phtw, acttwsys, phtwr)
1781
1782     ntw = acttwsys(1) + acttwsys(2) + acttwsys(3) + acttwsys(4) +
1783 & acttwsys(5) + acttwsys(6)
1784
1785     do i=1,3
1786         do j=1,3
1787             amat(i,j) = sigdt(i,j)
1788             bmat(i,j) = 0.0d+0
1789 c         if basal slip is active
1790             if (actbas) then
1791                 amat(i,j) = amat(i,j) - 2.0d+0*gbas*pbas(i,j)
1792                 bmat(i,j) = 2.0d+0*ysbas*pbas(i,j)
1793             end if
1794 c         if basal slip and twinning are active
1795             if ((actbas).and.(acttw)) then
1796                 do k=1,ntw
1797                     do l=1,ntw
1798                         amat(i,j) = amat(i,j) - phtwr(i,j,k)*mhinv(k,l)*(gtwr(l)
1799 & - hmixr(l)*gbas)
1800                         bmat(i,j) = bmat(i,j) + phtwr(i,j,k)*mhinv(k,l)*(ystwr(l)
1801 & - hmixr(l)*ysbas)
1802                     end do
1803                 end do
1804 c             otherwise, if just twinning is active
1805             else if (acttw) then
1806                 do k=1,ntw
1807                     do l=1,ntw
1808                         amat(i,j) = amat(i,j) - phtwr(i,j,k)*mhinv(k,l)*gtwr(l)
1809                         bmat(i,j) = bmat(i,j) + phtwr(i,j,k)*mhinv(k,l)*ystwr(l)

```

```

1810         end do
1811     end do
1812 end if
1813 end do
1814 end do
1815
1816 a = amat(1,1)*amat(1,1)+amat(1,2)*amat(1,2)+amat(1,3)*amat(1,3)+
1817 & amat(2,1)*amat(2,1)+amat(2,2)*amat(2,2)+amat(2,3)*amat(2,3)+
1818 & amat(3,1)*amat(3,1)+amat(3,2)*amat(3,2)+amat(3,3)*amat(3,3)
1819
1820 b = amat(1,1)*bmat(1,1)+amat(1,2)*bmat(1,2)+amat(1,3)*bmat(1,3)+
1821 & amat(2,1)*bmat(2,1)+amat(2,2)*bmat(2,2)+amat(2,3)*bmat(2,3)+
1822 & amat(3,1)*bmat(3,1)+amat(3,2)*bmat(3,2)+amat(3,3)*bmat(3,3)
1823
1824 c = bmat(1,1)*bmat(1,1)+bmat(1,2)*bmat(1,2)+bmat(1,3)*bmat(1,3)+
1825 & bmat(2,1)*bmat(2,1)+bmat(2,2)*bmat(2,2)+bmat(2,3)*bmat(2,3)+
1826 & bmat(3,1)*bmat(3,1)+bmat(3,2)*bmat(3,2)+bmat(3,3)*bmat(3,3)
1827
1828 a = 1.5d+0*a
1829 b = 3.0d+0*b
1830 c = 1.5d+0*c
1831
1832 return
1833 end
1834
1835
1836 C=====
1837 C=====
1838 c Calculate conservative estimate of maximum von mises strain rate
1839 C-----
1840
1841     subroutine calc_max_epdsl(DSTRAN, SIGT, LAM, MU, DTIME,
1842 & EPSDCUTOFF, EPSDMAX)
1843
1844     implicit none
1845
1846 c     input
1847     double precision DSTRAN(6), SIGT, LAM, MU, DTIME, EPSDCUTOFF
1848
1849 c     output
1850     double precision EPSDMAX
1851
1852
1853 c     util
1854     double precision DEPSD1, DEPSD2, DEPSD3, DEPSD4, DEPSD5, DEPSD6
1855     double precision DEPSH, DEPSE
1856     double precision EMOD
1857
1858     EMOD = MU*(3.0D+0*LAM+2.0D+0*MU)/(LAM+MU)
1859
1860     DEPSH=(DSTRAN(1)+DSTRAN(2)+DSTRAN(3))/3.0d+0
1861     DEPSD1=DSTRAN(1)-DEPSH
1862     DEPSD2=DSTRAN(2)-DEPSH
1863     DEPSD3=DSTRAN(3)-DEPSH
1864     DEPSD4=DSTRAN(4)/2.0d+0
1865     DEPSD5=DSTRAN(5)/2.0d+0
1866     DEPSD6=DSTRAN(6)/2.0d+0
1867
1868     DEPSE=DEPSD1**2+DEPSD2**2+DEPSD3**2+2.0d+0*(DEPSD4**2+DEPSD5**2
1869 & + DEPSD6**2)
1870     DEPSE=DSQRT(2.0d+0/3.0d+0*DEPSE+EPSDCUTOFF)
1871     EPSDMAX=2.0d+0*(DEPSE+SIGT/EMOD)/DTIME
1872
1873     return
1874     end

```

```

1875
1876 C=====
1877 C=====
1878 c Calculate dgambas based
1879 C-----
1880
1881     subroutine calc_dgambas(gbas, ysbas, mu, depl, yssl, dgambas)
1882
1883     implicit none
1884
1885 c     input
1886     double precision gbas, ysbas, mu, depl, yssl
1887
1888 c     output
1889     double precision dgambas
1890
1891     dgambas = (gbas - ysbas*(1.d+0+3.0d+0*mu*depl/yssl))/mu
1892
1893     return
1894     end
1895
1896
1897 C=====
1898 C=====
1899 c Calculate dgambslip based on dgambas, dgamtw, sbas, mbas, ptw, phtw
1900 C-----
1901
1902     subroutine calc_dgambslip(dgambas, dgamtw, sbas, mbas, ptw,
1903 & phtw, dgambslip, sbslip)
1904
1905     implicit none
1906
1907 c     input
1908     double precision dgambas
1909     double precision dgamtw(6), sbas(3), mbas(3)
1910     double precision ptw(3,3,6), phtw(3,3,6)
1911
1912 c     output
1913     double precision dgambslip, sbslip(3)
1914
1915 c     util
1916     integer i
1917
1918     double precision rvec(3)
1919
1920     rvec(1) = 0.0d+0
1921     rvec(2) = 0.0d+0
1922     rvec(3) = 0.0d+0
1923
1924     do i=1,6
1925         rvec(1) = rvec(1)+2.0d+0*dgamtw(i)*
1926 &         (mbas(1)*(ptw(1,1,i)-phtw(1,1,i))+
1927 &         mbas(2)*(ptw(1,2,i)-phtw(1,2,i))+
1928 &         mbas(3)*(ptw(1,3,i)-phtw(1,3,i)))
1929         rvec(2) = rvec(2)+2.0d+0*dgamtw(i)*
1930 &         (mbas(1)*(ptw(2,1,i)-phtw(2,1,i))+
1931 &         mbas(2)*(ptw(2,2,i)-phtw(2,2,i))+
1932 &         mbas(3)*(ptw(2,3,i)-phtw(2,3,i)))
1933         rvec(3) = rvec(3)+2.0d+0*dgamtw(i)*
1934 &         (mbas(1)*(ptw(3,1,i)-phtw(3,1,i))+
1935 &         mbas(2)*(ptw(3,2,i)-phtw(3,2,i))+
1936 &         mbas(3)*(ptw(3,3,i)-phtw(3,3,i)))
1937     end do
1938
1939     dgambslip = dsqrt((dgambas*sbas(1)-rvec(1))**2 +

```

```

1940      & (dgambas*sbas(2)-rvec(2))**2 + (dgambas*sbas(3)-rvec(3))**2)
1941
1942 c   calculate sb slip, with null value if basal slip doesnt occur
1943 if (dgambslip.le.1d-12) then
1944     sb slip(1) = 1.0d+0
1945     sb slip(2) = 1.0d+0
1946     sb slip(3) = 1.0d+0
1947 else
1948     sb slip(1) = (dgambas*sbas(1)-rvec(1))/dgambslip
1949     sb slip(2) = (dgambas*sbas(2)-rvec(2))/dgambslip
1950     sb slip(3) = (dgambas*sbas(3)-rvec(3))/dgambslip
1951 end if
1952
1953 return
1954 end
1955
1956 C=====
1957 C=====
1958 c   Calculate deviatoric stress based on trial stress and amount
1959 c   of plastic deformation
1960 c   - This uses phattw, and pbas. Could also use ptwin and pbslip, just
1961 c   have to make sure to use the correct variables
1962 C-----
1963
1964     subroutine calc_return_dev_stress(sigdT, acttwsys, mu, dgamtw,
1965 & dgambas, depl, yssl, phtw, pbas, sigd)
1966
1967     implicit none
1968
1969 c   input
1970 double precision sigdT(3,3), dgamtw(6), phtw(3,3,6), pbas(3,3)
1971 integer acttwsys(6)
1972 double precision mu, dgambas, depl, yssl
1973
1974 c   output
1975 double precision sigd(3,3)
1976
1977 c   util
1978 integer i,j
1979 double precision phtwr(3,3,6)
1980
1981 call calc_reduced_phtw(phtw, acttwsys, phtwr)
1982
1983 do i=1,3
1984   do j=1,3
1985     sigd(i,j) = (sigdT(i,j)-2.0d0*mu*(phtwr(i,j,1)*dgamtw(1)
1986 & + phtwr(i,j,2)*dgamtw(2) + phtwr(i,j,3)*dgamtw(3)
1987 & + phtwr(i,j,4)*dgamtw(4) + phtwr(i,j,5)*dgamtw(5)
1988 & + phtwr(i,j,6)*dgamtw(6) + pbas(i,j)*dgambas))
1989 & / (1.0d+0+3.0d+0*mu*depl/ysl)
1990   end do
1991 end do
1992
1993 return
1994 end
1995
1996 C=====
1997 C=====
1998 c   Read in stress in indicial notation, output abaqus notation
1999 C-----
2000     subroutine ind_to_voigtbq(sigi, sigabq)
2001
2002     implicit none
2003
2004 c   input
2005 double precision sigi(3,3)

```

```

2005
2006
2007 c      output
2008     double precision sigabq(6)
2009
2010     sigabq(1) = sigi(1,1)
2011     sigabq(2) = sigi(2,2)
2012     sigabq(3) = sigi(3,3)
2013     sigabq(4) = sigi(1,2)
2014     sigabq(5) = sigi(1,3)
2015     sigabq(6) = sigi(2,3)
2016
2017     return
2018 end
2019
2020
2021 c=====
2022 c=====
2023 c Calculate dgamtwt and record active twin systems. dgamtwt is stored
2024 c in reduced format
2025 c-----
2026
2027     subroutine calc_dgamtwt(gtw, hmix0, ystw, yssl, mu, dgambas,
2028 &     deplsl, deactTOL, acttwsys, dgamtwt)
2029
2030     implicit none
2031
2032 c      input
2033     double precision gtw(6), hmix0(6), ystw(6)
2034     double precision yssl, mu, dgambas, deplsl, deactTOL
2035
2036 c      output
2037     integer acttwsys(6)
2038     double precision dgamtwt(6)
2039
2040 c      util
2041     double precision minslip
2042     integer i,j, nact, minslipindex
2043     double precision mhinvt(6,6), hmixr(6), gtwr(6), ystwr(6)
2044     logical negslip
2045
2046 c      test
2047     double precision gtwmod(6)
2048
2049 c      calc act tw sys based on trial stress less basal slip
2050     do i=1,6
2051         gtwmod(i) = gtw(i) - mu*dgambas*hmix0(i)
2052     end do
2053     call calc_acttwsys_stress(gtwmod, ystw, acttwsys)
2054
2055     negslip = .true.
2056     nact = 0
2057     do i=1,6
2058         nact = nact + acttwsys(i)
2059     end do
2060
2061     20 if ((negslip).and.(nact.gt.0)) then
2062
2063 c      zero everything out
2064     do i=1,6
2065         hmixr(i) = 0.0d+0
2066         dgamtwt(i) = 0.0d+0
2067     do j=1,6
2068         mhinvt(i,j) = 0.0d+0
2069     end do
2070     end do

```



```

2070
2071     minslip = 0.0d+0
2072
2073 c     determine mhinu and hmix according to number of active systems
2074 call calc_minu(actwsys, mhinu)
2075 call calc_reduced_hmix(hmix0, actwsys, hmixr)
2076 call calc_reduced_gys(gtw, ystw, actwsys, gtwr, ystwr)
2077
2078 c     sum over active systems
2079 do i=1,nact
2080     do j=1,nact
2081         dgamtw(i) = dgamtw(i) + mhinu(i,j)/(2.0d+0*mu)*
2082 &         (gtwr(j)-mu*dgambas*hmixr(j)
2083 &         - ystwr(j)*(1.0d+0+3.0d+0*mu*depsl/yssl))
2084     end do
2085
2086 c     if slip is most negative, record index with most negative slip
2087 if (dgamtw(i).lt.minslip) then
2088     minslip = dgamtw(i)
2089     minslipindex = i
2090 end if
2091 end do
2092
2093 c     if neg slip occurred, figure out if most neg entry has an
2094 c     active pair with equal slip. if so, check which resolved
2095 c     shear stress is higher and eliminate that
2096 if (minslip.lt.0.0d+0) then
2097     call identify_moreneg_twinpair(minslipindex, actwsys,
2098 &     dgamtw, gtwr, dgambas, hmixr, mu)
2099     call remove_actwsys_entry(actwsys, minslipindex)
2100     dgamtw(minslipindex) = 0.0d+0
2101     nact = nact - 1
2102 else
2103 c     set negslip = false if everything is in equilibrium
2104 c     otherwise, set negslip = true and eliminate a deformation mode
2105 c     -- only eliminate twinning (not bas slip)
2106
2107     call twin_equilm_check(actwsys, nact, dgamtw, negslip,
2108 &     dgambas, hmixr, gtwr, ystw, mu, depsl, yssl, deactTOL)
2109
2110 end if
2111
2112 c     go back through loop (only happens if neg slip occurs)
2113 goto 20
2114 end if
2115
2116 return
2117 end
2118
2119 C=====
2120 C=====
2121 c The input to this is minslipindex, which identifies the most neg slip
2122 c -- This may not select the right system to delete since slip pairs
2123 c are identical (due to SVD) if both are active. Therefore, make
2124 c sure to eliminate the other with the lowest modified shear stress
2125 c
2126 c The output is minslipindex, modified if necessary to reflect the
2127 c twin pair with the lowest crss if they are both active
2128 C-----
2129
2130 subroutine identify_moreneg_twinpair(minslipindex, actwsys,
2131 & dgamtw, gtwr, dgambas, hmixr, mu)
2132
2133 implicit none
2134
2135 c input/output

```

```

2135
2136     integer minslipindex
2137
2138 c   input
2139     integer acttwsys(6)
2140     double precision dgamtw(6), gtwr(6), dgambas, hmixr(6), mu
2141
2142 c   util
2143     integer nact, j, minslipindexfull, di
2144     double precision tempvar, tempvar2
2145
2146 c   find what index minslipindex corresponds to in unreduced notation
2147     nact = 0
2148     do j=1,6
2149         if (acttwsys(j).eq.1) then
2150             nact = nact + 1
2151             if (nact.eq.minslipindex) then
2152                 minslipindexfull = j
2153             end if
2154         end if
2155     end do
2156
2157 c   determine if pair index is 1 higher or lower. set di accordingly
2158     if ((minslipindexfull.eq.1).or.(minslipindexfull.eq.3).or.
2159 & (minslipindexfull.eq.5)) then
2160         di = 1
2161     else
2162         di = -1
2163     end if
2164
2165 c   determine if pair is active and has the same negative slip amount
2166     tempvar = dabs(dgamtw(minslipindex)-dgamtw(minslipindex+di))
2167     if ((acttwsys(minslipindexfull+di).eq.1).and.
2168 & (tempvar.le.1d-6)) then
2169         CONTINUE
2170     else
2171         RETURN
2172     end if
2173
2174 c   if the pair is active, determine which has the lowest modified
2175 c   crss, and select that one as the one to eliminate
2176     tempvar = gtwr(minslipindex) - mu*hmixr(minslipindex)*
2177 & dgambas
2178     tempvar2=gtwr(minslipindex+di)- mu*hmixr(minslipindex+di)*
2179 & dgambas
2180     if (tempvar.gt.tempvar2) then
2181         minslipindex = minslipindex + di
2182     end if
2183
2184     return
2185 end
2186
2187 C=====
2188 C=====
2189 c   For twinning, after it is determined there is no negative slip, do
2190 c   this check to make sure tau>ys. If not, eliminate the most neg
2191 c   twin system by using tau - ys = taurel.
2192 c   - If everything is in equilm, set negslip = false.
2193 c   - If sys eliminated, change nact, acttwsys, and set dgamtw(i) = 0
2194 c-----
2195     subroutine twin_equilm_check(acttwsys, nact, dgamtw, negslip,
2196 & dgambas, hmixr, gtwr, ystwr, mu, depl, yssl, deactTOL)
2197
2198     implicit none
2199
2200 c   input/output

```

```

2201     integer acttwsys(6), nact
2202     double precision dgamtw(6)
2203     logical negslip
2204
2205 c     input
2206     double precision dgambas, hmixr(6), gtwr(6), ystwr(6), mu
2207     double precision depl, yssl, deactTOL
2208
2209 c     util
2210     integer i,j
2211     double precision mhatr(6,6), taur(6), maxrel
2212     integer maxreindex
2213
2214     maxrel = 100.0d0
2215     maxreindex = 0
2216
2217     call calc_mhat(acttwsys, mhatr)
2218     do i=1,nact
2219         taur(i) = gtwr(i) - hmixr(i)*dgambas/2.0d+0
2220         do j=1,nact
2221             taur(i) = taur(i) - 2.0d+0*mu*mhatr(i,j)*dgamtw(j)
2222         end do
2223         taur(i) = taur(i) / (1.0d+0+3.0d+0*mu*depl/ysl)
2224
2225 c         print*, 'tau - ys, red sys ', i, ': ', taur(i)-ystwr(i)
2226         if (taur(i)-ystwr(i).le.maxrel) then
2227             maxrel = taur(i)-ystwr(i)
2228 c         print*, 'maxrel: ', maxrel
2229             maxreindex = i
2230         end if
2231     end do
2232
2233 c     if everything is equilibrium, set negslip = false
2234     if (maxrel/mu.le.-deactTOL) then
2235         dgamtw(maxreindex) = 0.0d+0
2236         call remove_acttwsys_entry(acttwsys, maxreindex)
2237         nact = nact - 1
2238     else
2239         negslip = .false.
2240     end if
2241
2242     return
2243 end
2244
2245 c=====
2246 c=====
2247 c Given an index from the reduced representation, deactivates this
2248 c currently active entry from the full representation of acttwsys
2249 c - ex: given acttwsys = (1,0,1,0,1,0) and i = 2
2250 c     acttwsys becomes (1,0,0,0,1,0) because the second active
2251 c     entry has been deleted
2252 c-----
2253     subroutine remove_acttwsys_entry(acttwsys, i)
2254
2255     implicit none
2256 c     input/output
2257     integer acttwsys(6)
2258
2259 c     input
2260     integer i
2261
2262 c     util
2263     integer nact, j
2264
2265     nact = 0
2266     do j=1,6

```

```

2267         if (acttwsys(j).eq.1) then
2268             nact = nact + 1
2269             if (nact.eq.i) then
2270                 acttwsys(j) = 0
2271             end if
2272         end if
2273     end do
2274
2275     return
2276 end
2277
2278 c=====
2279 c=====
2280 c Calculate acttwsys based on trial stress and strength
2281 c-----
2282
2283     subroutine calc_acttwsys_stress(gtw, ystw, acttwsys)
2284
2285     implicit none
2286
2287     c input
2288     double precision gtw(6), ystw(6)
2289
2290     c output
2291     integer acttwsys(6)
2292
2293     c util
2294     integer n
2295
2296     do n=1,6
2297         if (gtw(n).ge.ystw(n)) then
2298             acttwsys(n) = 1
2299         else
2300             acttwsys(n) = 0
2301         end if
2302     end do
2303
2304     return
2305 end
2306
2307 c=====
2308 c Calculate phtwr based on phtw
2309 c-----
2310
2311     subroutine calc_reduced_phtw(phtw, acttwsys, phtwr)
2312
2313     implicit none
2314
2315     c input
2316     double precision phtw(3,3,6)
2317     integer acttwsys(6)
2318
2319     c output
2320     double precision phtwr(3,3,6)
2321
2322     c util
2323     integer i, n, nact
2324
2325     nact = 0
2326     do n=1,6
2327         phtwr(1,1,n) = 0.0d+0
2328         phtwr(1,2,n) = 0.0d+0
2329         phtwr(1,3,n) = 0.0d+0
2330         phtwr(2,1,n) = 0.0d+0
2331         phtwr(2,2,n) = 0.0d+0
2332         phtwr(2,3,n) = 0.0d+0

```

```

2333      phtwr(3,1,n) = 0.0d+0
2334      phtwr(3,2,n) = 0.0d+0
2335      phtwr(3,3,n) = 0.0d+0
2336      nact = nact + acttwsys(n)
2337  end do
2338
2339  i = 1
2340  n = 1
2341
2342  10 if (i.le.nact) then
2343      if (acttwsys(n).eq.1) then
2344          phtwr(1,1,i) = phtw(1,1,n)
2345          phtwr(1,2,i) = phtw(1,2,n)
2346          phtwr(1,3,i) = phtw(1,3,n)
2347          phtwr(2,1,i) = phtw(2,1,n)
2348          phtwr(2,2,i) = phtw(2,2,n)
2349          phtwr(2,3,i) = phtw(2,3,n)
2350          phtwr(3,1,i) = phtw(3,1,n)
2351          phtwr(3,2,i) = phtw(3,2,n)
2352          phtwr(3,3,i) = phtw(3,3,n)
2353          i = i + 1
2354      end if
2355      n = n + 1
2356      goto 10
2357  end if
2358
2359  return
2360  end
2361
2362  C=====
2363  C=====
2364  c Calculate gtwr, ystwr based on their original values and acttwsys
2365  c - ex: if gtw0 = (1,2,3,4,5), and acttwsys = (1,0,1,0,1) then
2366  c       gtwr = (1,3,5,0,0), and is used in dgamtw calculations
2367  C-----
2368
2369
2370      subroutine calc_reduced_gys(gtw0, ystw0, acttwsys, gtwr, ystwr)
2371
2372      implicit none
2373
2374  c      input
2375      double precision gtw0(6), ystw0(6)
2376      integer acttwsys(6)
2377
2378  c      output
2379      double precision gtwr(6), ystwr(6)
2380
2381  c      util
2382      integer i, n
2383
2384      do n=1,6
2385
2386          gtwr(n) = 0.0d+0
2387          ystwr(n) = 0.0d+0
2388      end do
2389
2390      n = 1
2391      do i=1,6
2392          if (acttwsys(i).eq.1) then
2393              gtwr(n) = gtw0(i)
2394              ystwr(n) = ystw0(i)
2395              n = n + 1
2396          end if
2397      end do

```

```

2398         return
2399     end
2400
2401
2402 C=====
2403 C=====
2404 c  Calculate hmix based on hmix0 and which systems are active
2405 c  ex - if acttwsys is (0,1,0,1,1,0) then
2406 c          hmix(1) = hmix0(2), hmix(2) = hmix0(4), hmix(3) = hmix0(5)
2407 c          and hmix(4-6) = 0
2408 C-----
2409
2410     subroutine calc_reduced_hmix(hmix0, acttwsys, hmix)
2411
2412     implicit none
2413
2414 c  input
2415     double precision hmix0(6)
2416     integer acttwsys(6)
2417
2418 c  output
2419     double precision hmix(6)
2420
2421 c  util
2422     integer i, n, nact
2423
2424     nact = 0
2425     do n=1,6
2426         hmix(n) = 0.0d+0
2427         nact = nact + acttwsys(n)
2428     end do
2429
2430     i = 1
2431     n = 1
2432
2433 10  if (i.le.nact) then
2434         if (acttwsys(n).eq.1) then
2435             hmix(i) = hmix0(n)
2436             i = i + 1
2437         end if
2438         n = n + 1
2439         goto 10
2440     end if
2441
2442     return
2443     end
2444
2445 C=====
2446 C=====
2447 c  Calculate h^\alpha = ptw_\alpha:pbas
2448 C-----
2449
2450     subroutine init_hmix(ptw, pbas, hmix)
2451
2452     implicit none
2453
2454 c  input
2455     double precision ptw(3,3,6), pbas(3,3)
2456
2457 c  output
2458     double precision hmix(6)
2459
2460 c  util
2461     integer n
2462

```

```

2463     do n=1,6
2464         hmix(n) =      2.0d+0*(ptw(1,1,n)*pbas(1,1) + ptw(1,2,n)*pbas(1,2)
2465     & +ptw(1,3,n)*pbas(1,3)+ptw(2,1,n)*pbas(2,1)+ptw(2,2,n)*pbas(2,2)
2466     & +ptw(2,3,n)*pbas(2,3)+ptw(3,1,n)*pbas(3,1)+ptw(3,2,n)*pbas(3,2)
2467     & +ptw(3,3,n)*pbas(3,3))
2468     end do
2469
2470     return
2471 end
2472
2473 c=====
2474 c=====
2475 c Calculate driving forces for yield based on trial stress for
2476 c basal, twinning, and non-basal slip
2477 c-----
2478
2479     subroutine calc_taus(sigd, pbas, ptw, gbas, gtw, gsl)
2480
2481     implicit none
2482
2483     !input
2484     double precision sigd(3,3), pbas(3,3), ptw(3,3,6)
2485
2486     !output
2487     double precision gbas, gtw(6), gsl
2488
2489     !util
2490     integer i
2491
2492
2493
2494 c     do i=1,6
2495 c         gtw(i) = sigd(1,1)*ptw(1,1,i)+sigd(1,2)*ptw(1,2,i) +
2496 c     & sigd(1,3)*ptw(1,3,i)+sigd(2,1)*ptw(2,1,i)+
2497 c     & sigd(2,2)*ptw(2,2,i)+sigd(2,3)*ptw(2,3,i)+
2498 c     & sigd(3,1)*ptw(3,1,i)+sigd(3,2)*ptw(3,2,i)+
2499 c     & sigd(3,3)*ptw(3,3,i)
2500 c     end do
2501
2502     do i=1,6
2503         gtw(i) = sigd(1,1)*ptw(1,1,i)+sigd(2,2)*ptw(2,2,i)+
2504     & sigd(3,3)*ptw(3,3,i)+2.0d+0*(sigd(1,2)*ptw(1,2,i)+
2505     & sigd(1,3)*ptw(1,3,i)+sigd(2,3)*ptw(2,3,i))
2506     end do
2507
2508     gbas = sigd(1,1)*pbas(1,1)+sigd(2,2)*pbas(2,2)+
2509     & sigd(3,3)*pbas(3,3)+2.0d+0*(sigd(1,2)*pbas(1,2)+
2510     & sigd(1,3)*pbas(1,3)+sigd(2,3)*pbas(2,3))
2511
2512     gsl = dsqrt(3.0d+0/2.0d+0*(sigd(1,1)**2+
2513     & sigd(2,2)**2+sigd(3,3)**2+2.0d+0*(sigd(1,2)**2 +
2514     & sigd(1,3)**2 + sigd(2,3)**2)))
2515
2516     return
2517 end
2518
2519
2520 c=====
2521 c=====
2522 c Calculate which deformation modes are potentially active based on
2523 c their orientation from the last time step and the trial stress
2524 c-----
2525
2526     subroutine calc_potactive_modes(gbas,gtw,gsl,ysbas,ystw,captwin,
2527     & yssl, actTOL, mu, actbas, acttw, actsl)

```

```

2528
2529     implicit none
2530
2531 c   input
2532     double precision gbas, gtw(6), gsl
2533     double precision ysbas, ystw(6), yssl, actTOL, mu
2534     logical captwin
2535
2536 c   output
2537     logical actbas, acttw, actsl
2538
2539 c   util
2540     integer n
2541     double precision diff
2542
2543 c   basal
2544     diff = (dabs(gbas)-ysbas)/mu
2545     if (diff.ge.actTOL) then
2546         actbas = .true.
2547     else
2548         actbas = .false.
2549     endif
2550
2551 c   twin
2552     acttw = .false.
2553     do n=1,6
2554         diff = (gtw(n)-ystw(n))/mu
2555         if (diff.ge.actTOL) then
2556             acttw = .true.
2557         end if
2558     end do
2559     if (captwin) acttw = .false.
2560
2561 c   non-basal slip
2562     diff = dabs(gsl) - yssl
2563     if (diff.ge.actTOL) then
2564         actsl = .true.
2565     else
2566         actsl = .false.
2567     endif
2568
2569     return
2570 end
2571
2572 C=====
2573 C=====
2574 c   Calculate sbasal and Pbasal based on trial stress, mbasal
2575 c   o=sig, m = mbas, s=sbas, p=pbas
2576 C-----
2577
2578     subroutine calc_spbas(o, m, s, p)
2579
2580     implicit none
2581
2582 c   input
2583     double precision o(3,3), m(3)
2584
2585 c   output
2586     double precision s(3), p(3,3)
2587
2588 c   util
2589     double precision num1(3)
2590     double precision den1, num2
2591
2592 c       den1 = den1 + m(i)*o(i,j)*o(j,k)*m(k)

```



```

2593     den1 = m(1)*o(1,1)**2*m(1) + m(2)*o(1,1)*o(1,2)*m(1)
2594     & + m(1)*o(1,2)**2*m(1) + m(3)*o(1,1)*o(1,3)*m(1)
2595     & + m(1)*o(1,3)**2*m(1) + m(2)*o(1,2)*o(2,2)*m(1)
2596     & + m(3)*o(1,2)*o(2,3)*m(1) + m(2)*o(1,3)*o(2,3)*m(1)
2597     & + m(3)*o(1,3)*o(3,3)*m(1) + m(1)*o(1,1)*o(1,2)*m(2)
2598     & + m(2)*o(1,2)**2*m(2) + m(3)*o(1,2)*o(1,3)*m(2)
2599     & + m(1)*o(1,2)*o(2,2)*m(2) + m(2)*o(2,2)**2*m(2)
2600     & + m(1)*o(1,3)*o(2,3)*m(2) + m(3)*o(2,2)*o(2,3)*m(2)
2601     & + m(2)*o(2,3)**2*m(2) + m(3)*o(2,3)*o(3,3)*m(2)
2602     & + m(1)*o(1,1)*o(1,3)*m(3) + m(2)*o(1,2)*o(1,3)*m(3)
2603     & + m(3)*o(1,3)**2*m(3) + m(1)*o(1,2)*o(2,3)*m(3)
2604     & + m(2)*o(2,2)*o(2,3)*m(3) + m(3)*o(2,3)**2*m(3)
2605     & + m(1)*o(1,3)*o(3,3)*m(3) + m(2)*o(2,3)*o(3,3)*m(3)
2606     & + m(3)*o(3,3)**2*m(3)
2607
2608 c     num1(i) = num1(i) + sig(i,j)*mbas(j)
2609 c     num2 = num2 + mbas(i)*sig(i,j)*mbas(j)
2610     num1(1) = o(1,1)*m(1)+o(1,2)*m(2)+o(1,3)*m(3)
2611     num1(2) = o(2,1)*m(1)+o(2,2)*m(2)+o(2,3)*m(3)
2612     num1(3) = o(3,1)*m(1)+o(3,2)*m(2)+o(3,3)*m(3)
2613     num2 = m(1)*(o(1,1)*m(1)+o(1,2)*m(2)+o(1,3)*m(3))
2614     & + m(2)*(o(2,1)*m(1)+o(2,2)*m(2)+o(2,3)*m(3))
2615     & + m(3)*(o(3,1)*m(1)+o(3,2)*m(2)+o(3,3)*m(3))
2616
2617     if ((den1-num2**2).le.1d-10) then
2618 c         there is no shear stress on the basal plane, regardless of sbas
2619 c         so assign sbas an arbitrary direction
2620         s(1) = 1.0d+0
2621         s(2) = 0.0d+0
2622         s(3) = 0.0d+0
2623     else
2624         s(1) = (num1(1) - m(1)*num2)/ dsqrt(den1-num2**2)
2625         s(2) = (num1(2) - m(2)*num2)/ dsqrt(den1-num2**2)
2626         s(3) = (num1(3) - m(3)*num2)/ dsqrt(den1-num2**2)
2627     end if
2628
2629     p(1,1) = s(1)*m(1)
2630     p(2,2) = s(2)*m(2)
2631     p(3,3) = s(3)*m(3)
2632     p(1,2) = 0.5d+0*(s(1)*m(2)+s(2)*m(1))
2633     p(2,1) = p(1,2)
2634     p(1,3) = 0.5d+0*(s(1)*m(3)+s(3)*m(1))
2635     p(3,1) = p(1,3)
2636     p(2,3) = 0.5d+0*(s(2)*m(3)+s(3)*m(2))
2637     p(3,2) = p(2,3)
2638
2639     return
2640 end
2641
2642 C=====
2643 C=====
2644 c Return deviatoric trial stress in indicial notation
2645 C-----
2646
2647     subroutine calc_trial_devstress(STRESS, DSTRAIN, MU, SIGTDEV)
2648
2649     implicit none
2650
2651 c     input
2652     double precision STRESS(6), DSTRAIN(6)
2653     double precision MU
2654
2655 c     output
2656     double precision SIGTDEV(3,3)
2657

```

```

2658 c      util
2659      double precision DEPSH, SIGH0
2660      double precision DEPSD1, DEPSD2, DEPSD3, DEPSD4, DEPSD5, DEPSD6
2661
2662 C DEVIATORIC PART OF THE STRAIN INCREMENT
2663 C
2664      DEPSH=(DSTRAIN(1)+DSTRAIN(2)+DSTRAIN(3))/3.0d+0
2665      DEPSD1=DSTRAIN(1)-DEPSH
2666      DEPSD2=DSTRAIN(2)-DEPSH
2667      DEPSD3=DSTRAIN(3)-DEPSH
2668      DEPSD4=DSTRAIN(4)/2.0d+0
2669      DEPSD5=DSTRAIN(5)/2.0d+0
2670      DEPSD6=DSTRAIN(6)/2.0d+0
2671
2672      SIGH0=(STRESS(1)+STRESS(2)+STRESS(3))/3.0d+0
2673      SIGTDEV(1,1)=STRESS(1)-SIGH0+2.0d+0*MU*DEPSD1
2674      SIGTDEV(2,2)=STRESS(2)-SIGH0+2.0d+0*MU*DEPSD2
2675      SIGTDEV(3,3)=STRESS(3)-SIGH0+2.0d+0*MU*DEPSD3
2676      SIGTDEV(1,2)=STRESS(4)+2.0d+0*MU*DEPSD4
2677      SIGTDEV(2,1)=STRESS(4)+2.0d+0*MU*DEPSD4
2678      SIGTDEV(1,3)=STRESS(5)+2.0d+0*MU*DEPSD5
2679      SIGTDEV(3,1)=STRESS(5)+2.0d+0*MU*DEPSD5
2680      SIGTDEV(2,3)=STRESS(6)+2.0d+0*MU*DEPSD6
2681      SIGTDEV(3,2)=STRESS(6)+2.0d+0*MU*DEPSD6
2682
2683 c      SIGT=SIGDEV(1)**2+SIGDEV(2)**2+SIGDEV(3)**2+2.0d+0*(SIGDEV(4)**2
2684 c      & + SIGDEV(5)**2 + SIGDEV(6)**2)
2685 c      SIGT=DSQRT(3.0d+0/2.0d+0*SIGT)
2686
2687
2688      return
2689      end
2690
2691 C=====
2692 C=====
2693 c Just return ddsdde for lin elast umat
2694 C-----
2695
2696      subroutine elastddsde(lam, mu, ntens, ndi, ddsdde)
2697      implicit none
2698
2699      !input
2700      integer i,j, ntens, ndi
2701      double precision lam, mu
2702
2703      !output
2704      double precision ddsdde(ntens,ntens)
2705
2706      do i=1,ntens
2707          do j=1,ntens
2708              ddsdde(i,j) = 0.0d0
2709          end do
2710      end do
2711
2712      do i=1, ndi
2713          do j=1, ndi
2714              ddsdde(j,i) = lam
2715          end do
2716          ddsdde(i,i) = lam+2.0d0*mu
2717      end do
2718
2719      do i=ndi+1, ntens
2720          ddsdde(i,i) = mu
2721      end do
2722
2723      return

```

```

2724         end
2725
2726 C=====
2727 C=====
2728 c   Calculate stwin, mtwin, and mbasal based on re, where re has been
2729 c   initialized to include the initial rotations in uinit
2730 c   Dependencies - init_stw_mtw
2731 c-----
2732
2733     subroutine calc_stw_mtw_mbas(re, stw, mtw, mbas)
2734     implicit none
2735
2736     !input
2737     double precision re(3,3)
2738
2739     !output
2740     double precision stw(3,6), mtw(3,6), mbas(3)
2741
2742     !util
2743     integer n
2744     double precision stw0(3,6), mtw0(3,6), mbas0(3)
2745
2746     !initialize all the variables in the reference frame
2747     call init_stw_mtw(stw0, mtw0)
2748     mbas0(1) = 0.0d+0
2749     mbas0(2) = 0.0d+0
2750     mbas0(3) = 1.0d+0
2751
2752 c   rotate twin system
2753     do n=1,6
2754         stw(1,n)=re(1,1)*stw0(1,n)+re(1,2)*stw0(2,n)+re(1,3)*stw0(3,n)
2755         stw(2,n)=re(2,1)*stw0(1,n)+re(2,2)*stw0(2,n)+re(2,3)*stw0(3,n)
2756         stw(3,n)=re(3,1)*stw0(1,n)+re(3,2)*stw0(2,n)+re(3,3)*stw0(3,n)
2757         mtw(1,n)=re(1,1)*mtw0(1,n)+re(1,2)*mtw0(2,n)+re(1,3)*mtw0(3,n)
2758         mtw(2,n)=re(2,1)*mtw0(1,n)+re(2,2)*mtw0(2,n)+re(2,3)*mtw0(3,n)
2759         mtw(3,n)=re(3,1)*mtw0(1,n)+re(3,2)*mtw0(2,n)+re(3,3)*mtw0(3,n)
2760     end do
2761
2762     mbas(1)=re(1,1)*mbas0(1)+re(1,2)*mbas0(2)+re(1,3)*mbas0(3)
2763     mbas(2)=re(2,1)*mbas0(1)+re(2,2)*mbas0(2)+re(2,3)*mbas0(3)
2764     mbas(3)=re(3,1)*mbas0(1)+re(3,2)*mbas0(2)+re(3,3)*mbas0(3)
2765
2766     return
2767     end
2768
2769 C=====
2770 C=====
2771 c   Calculate ptw and phtw based on stw, mtw, and mbas (c)
2772 c-----
2773
2774     subroutine calc_ptw_phtw(stw, mtw, c, ptw, phtw)
2775     !recall mbas = c
2776     ! to test, form pcirc from subtracted term on phtw and then
2777     !   ensure that phtw:pcirc = 0 for all 6 systems
2778
2779     implicit none
2780
2781     !input
2782     double precision stw(3,6), mtw(3,6), c(3)
2783
2784     !output
2785     double precision ptw(3,3,6), phtw(3,3,6)
2786
2787     !util
2788     integer n

```

```

2789     double precision a(3), b
2790
2791     !calculate ptwin
2792     do n=1,6
2793         ptw(1,1,n) = 0.5d+0*(stw(1,n)*mtw(1,n)+stw(1,n)*mtw(1,n))
2794         ptw(1,2,n) = 0.5d+0*(stw(1,n)*mtw(2,n)+stw(2,n)*mtw(1,n))
2795         ptw(1,3,n) = 0.5d+0*(stw(1,n)*mtw(3,n)+stw(3,n)*mtw(1,n))
2796         ptw(2,1,n) = ptw(1,2,n)
2797         ptw(2,2,n) = 0.5d+0*(stw(2,n)*mtw(2,n)+stw(2,n)*mtw(2,n))
2798         ptw(2,3,n) = 0.5d+0*(stw(2,n)*mtw(3,n)+stw(3,n)*mtw(2,n))
2799         ptw(3,1,n) = ptw(1,3,n)
2800         ptw(3,2,n) = ptw(2,3,n)
2801         ptw(3,3,n) = 0.5d+0*(stw(3,n)*mtw(3,n)+stw(3,n)*mtw(3,n))
2802
2803     c      a = ptw.c
2804     a(1)=ptw(1,1,n)*c(1)+ptw(1,2,n)*c(2)+ptw(1,3,n)*c(3)
2805     a(2)=ptw(2,1,n)*c(1)+ptw(2,2,n)*c(2)+ptw(2,3,n)*c(3)
2806     a(3)=ptw(3,1,n)*c(1)+ptw(3,2,n)*c(2)+ptw(3,3,n)*c(3)
2807
2808     c      b = c.ptw.c = c.a
2809     b = a(1)*c(1)+a(2)*c(2)+a(3)*c(3)
2810
2811     c      pcirc
2812     c      pc(i,j,n) = a(i,n)*c(j)+c(i)*a(j,n)-c(i)*c(j)*b(n)
2813
2814     c      phtw = ptw - pc
2815     phtw(1,1,n)=ptw(1,1,n)-
2816     &      (a(1)*c(1)+c(1)*a(1)-2.0d+0*c(1)*c(1)*b)
2817     phtw(1,2,n) = ptw(1,2,n)-
2818     &      (a(1)*c(2)+c(1)*a(2)-2.0d+0*c(1)*c(2)*b)
2819     phtw(1,3,n) = ptw(1,3,n)-
2820     &      (a(1)*c(3)+c(1)*a(3)-2.0d+0*c(1)*c(3)*b)
2821     phtw(2,1,n) = ptw(2,1,n)-
2822     &      (a(2)*c(1)+c(2)*a(1)-2.0d+0*c(2)*c(1)*b)
2823     phtw(2,2,n) = ptw(2,2,n)-
2824     &      (a(2)*c(2)+c(2)*a(2)-2.0d+0*c(2)*c(2)*b)
2825     phtw(2,3,n) = ptw(2,3,n)-
2826     &      (a(2)*c(3)+c(2)*a(3)-2.0d+0*c(2)*c(3)*b)
2827     phtw(3,1,n) = ptw(3,1,n)-
2828     &      (a(3)*c(1)+c(3)*a(1)-2.0d+0*c(3)*c(1)*b)
2829     phtw(3,2,n) = ptw(3,2,n)-
2830     &      (a(3)*c(2)+c(3)*a(2)-2.0d+0*c(3)*c(2)*b)
2831     phtw(3,3,n) = ptw(3,3,n)-
2832     &      (a(3)*c(3)+c(3)*a(3)-2.0d+0*c(3)*c(3)*b)
2833
2834     end do
2835
2836     return
2837     end
2838
2839     C=====
2840     C=====
2841     c Initialize a reference s and m for twinning in hcp metals
2842     c the value of H is specific to Magnesium.
2843     c Notation is same as Staroselsky thesis pp. 110, i.e., x->a2
2844     c note M0_3-6 THE C TERM SHOULD NOT BE DIVIDED BY 2
2845     C-----
2846
2847     subroutine init_stw_mtw(s,m)
2848     implicit none
2849
2850
2851     double precision s(3,6), m(3,6)
2852     double precision H, TWO, THREE, ZERO, DEN1, DEN2, S3
2853     parameter (H=1.624D+0,TWO=2.0D+0,THREE=3.0D+0,ZERO=0.0D+0)

```

```

2854
2855      S3 = dsqrt(3.0d+0)
2856      DEN1 = dsqrt(3.0d+0+H**2)
2857      DEN2 = 2.0d+0*DEN1
2858
2859      c                                TOP HEXAGON
2860      c
2861      c      1_
2862      c      / \      / \      / \      / \      / \
2863      c      \ /      \ /      \ /      \ /      \ /
2864      c      2
2865      c
2866      c      2_
2867      c      / \      / \      / \      / \      / \
2868      c      \ /      \ /      \ /      \ /      \ /
2869      c      1
2870      c                                BOTTOM HEXAGON
2871      c
2872      c      y
2873      c      |
2874      c      |___ x
2875      c      /
2876      c      /
2877      c      z
2878
2879      s(1,1) = ZERO
2880      s(2,1) = S3/DEN1
2881      s(3,1) = H/DEN1
2882      m(1,1) = ZERO
2883      m(2,1) = -H/DEN1
2884      m(3,1) = S3/DEN1
2885      s(1,2) = ZERO
2886      s(2,2) = -S3/DEN1
2887      s(3,2) = H/DEN1
2888      m(1,2) = ZERO
2889      m(2,2) = H/DEN1
2890      m(3,2) = S3/DEN1
2891      s(1,3) = THREE/DEN2
2892      s(2,3) = S3/DEN2
2893      s(3,3) = TWO*H/DEN2
2894      m(1,3) = -S3*H/DEN2
2895      m(2,3) = -H/DEN2
2896      m(3,3) = TWO*S3/DEN2
2897      s(1,4) = -THREE/DEN2
2898      s(2,4) = -S3/DEN2
2899      s(3,4) = TWO*H/DEN2
2900      m(1,4) = S3*H/DEN2
2901      m(2,4) = H/DEN2
2902      m(3,4) = TWO*S3/DEN2
2903      s(1,5) = -THREE/DEN2
2904      s(2,5) = S3/DEN2
2905      s(3,5) = TWO*H/DEN2
2906      m(1,5) = S3*H/DEN2
2907      m(2,5) = -H/DEN2
2908      m(3,5) = TWO*S3/DEN2
2909      s(1,6) = THREE/DEN2
2910      s(2,6) = -S3/DEN2
2911      s(3,6) = TWO*H/DEN2
2912      m(1,6) = -S3*H/DEN2
2913      m(2,6) = H/DEN2
2914      m(3,6) = TWO*S3/DEN2
2915
2916      return
2917      end
2918
2919      C=====

```

```

2920 C=====
2921 C  Make sure R is a pure rotation matrix by iteratively determing
2922 C    R from the input, considering R may have stretch and rotation
2923 C  NOTE: This is because advection may alter R aphysically
2924 C  AUTHOR: Rich Becker
2925 C-----
2926
2927     SUBROUTINE ensurerot(R)
2928     IMPLICIT NONE
2929 C    Parameter variables
2930     DOUBLE PRECISION TWO
2931     PARAMETER (TWO = 2.D0)
2932 C
2933 C    Argument variables
2934     DOUBLE PRECISION R(3,3)
2935 C
2936 C    Local variables
2937     INTEGER I, J, K
2938 C
2939     DOUBLE PRECISION A(3,3), AI(3,3), DET, ERR, TEST, TOL
2940
2941     DATA TOL/1.D-8/
2942 C
2943 C RETURNS A PURE ROTATION FROM THE POLAR DECOMPOSITION OF THE
2944 C INPUT MATRIX
2945 C
2946 C LIMIT NUMBER OF ITERATIONS TO 20. IF CONVERGENCE IS NOT
2947 C REACHED, THE PROGRAM WILL STOP.
2948 C
2949     DO 40 K=1,20
2950     DO 10 I=1,3
2951     DO 10 J=1,3
2952 10 A(I,J)=R(I,J)
2953 C
2954 C
2955 C COMPUTE INVERSE OF A
2956 C
2957     DET=A(1,1)*(A(2,2)*A(3,3)-A(3,2)*A(2,3))
2958 1  -A(1,2)*(A(2,1)*A(3,3)-A(3,1)*A(2,3))
2959 1  +A(1,3)*(A(2,1)*A(3,2)-A(3,1)*A(2,2))
2960 C
2961     AI(1,1)=(A(2,2)*A(3,3)-A(3,2)*A(2,3))/DET
2962     AI(1,2)=(A(3,2)*A(1,3)-A(1,2)*A(3,3))/DET
2963     AI(1,3)=(A(1,2)*A(2,3)-A(1,3)*A(2,2))/DET
2964     AI(2,1)=(A(3,1)*A(2,3)-A(2,1)*A(3,3))/DET
2965     AI(2,2)=(A(1,1)*A(3,3)-A(3,1)*A(1,3))/DET
2966     AI(2,3)=(A(1,3)*A(2,1)-A(1,1)*A(2,3))/DET
2967     AI(3,1)=(A(2,1)*A(3,2)-A(2,2)*A(3,1))/DET
2968     AI(3,2)=(A(1,2)*A(3,1)-A(1,1)*A(3,2))/DET
2969     AI(3,3)=(A(1,1)*A(2,2)-A(1,2)*A(2,1))/DET
2970 C
2971 C AVERAGE THE MATRIX AND THE TRANSPOSE OF ITS INVERSE.
2972 C
2973     DO 20 I=1,3
2974     DO 20 J=1,3
2975 20 R(I,J)=(A(I,J)+AI(J,I))/TWO
2976 C
2977 C DETERMINE LARGEST DEVIATION AND CHECK AGAINST TOLERANCE
2978 C
2979     ERR=0.0
2980     DO 30 I=1,3
2981     DO 30 J=1,3
2982     TEST=ABS(R(I,J)-A(I,J))
2983 30 IF(TEST.GT.ERR)ERR=TEST
2984     IF(ERR.LT.TOL) RETURN
2985 C

```

```

2986 C IF ERROR CONDITION IS MET BEFORE 20 ITERATIONS, CONTROL IS
2987 C RETURNED TO THE MAIN PROGRAM. OTHERWISE,
2988 C
2989     40 CONTINUE
2990     WRITE(6,100)
2991     100 FORMAT(' NOT A PURE ROTATION')
2992     STOP
2993     END
2994 C
2995
2996 C=====
2997 C=====
2998 c   Create rotation matrix from Euler angles in bunge notation
2999 C-----
3000
3001     subroutine create_rmat_bunge(a1, ap, a2, rmat)
3002     implicit none
3003
3004     double precision a1, ap, a2, s1, c1, sp, cp, s2, c2
3005     double precision rmat(3,3)
3006
3007     s1 = sin(a1)
3008     c1 = cos(a1)
3009     sp = sin(ap)
3010     cp = cos(ap)
3011     s2 = sin(a2)
3012     c2 = cos(a2)
3013
3014     rmat(1,1) = c1*c2 - s1*s2*cp
3015     rmat(2,1) = s1*c2 + c1*s2*cp
3016     rmat(3,1) = s2*sp
3017     rmat(1,2) = -c1*s2-s1*c2*cp
3018     rmat(2,2) = -s1*s2+c1*c2*cp
3019     rmat(3,2) = c2*sp
3020     rmat(1,3) = s1*sp
3021     rmat(2,3) = -c1*sp
3022     rmat(3,3) = cp
3023
3024     end
3025
3026     double precision function determinant(a)
3027     ! Calculate the determinant of a 3 x 3 matrix.
3028
3029     implicit none
3030
3031     double precision a(3,3)
3032     double precision b1, b2, b3
3033
3034     b1 = a(2,2) * a(3,3) - a(3,2) * a(2,3)
3035     b2 = a(3,1) * a(2,3) - a(2,1) * a(3,3)
3036     b3 = a(2,1) * a(3,2) - a(3,1) * a(2,2)
3037
3038     determinant = a(1,1) * b1 + a(1,2) * b2 + a(1,3) * b3
3039     if (dabs(determinant).le.1d-10) then
3040         print*, 'WARNING: JUST TOOK DET LESS THAN 1E-10'
3041     end if
3042
3043     return
3044     end

```



