branch: master MgSpall2 / init.f

zubaexy 2 minutes ago MgUMAT as obtained from Jeff on 22 June 2015

0 contributors

534 lines (467 sloc) 18.772 kB Raw Blame History

```
c-----------------------------------------------------------------------
c  ALE3D STATE VARIABLE INITIALIZATION SUBROUTINE
c-----------------------------------------------------------------------

      subroutine sdvinia(statev,nstatv,seed,orient,ngrains)
      implicit none
c     input
      integer nstatv, orient, ngrains
      double precision seed
c     output
      double precision statev(nstatv)
c     static params
      integer nstatvsingle, nstatvpoly, nmax, bigint
      parameter(nstatvsingle=25,nstatvpoly=31,nmax=1000)
      parameter(bigint=1790989824) !arbitrary
c     util
      integer i, itemp, iseed, i8_uniform
      double precision phi1, theta, phi2, rmat(3,3)
      double precision angs(nmax, 3)

c     Sanity checks for number of state variables declared
      if (ngrains.eq.1) then
        if(nstatv.ne.nstatvsingle) then
          print*, '=== WARNING IN SDVINIA === '
          print*,'nstatv is: ', nstatv, ' but should be ',
     &      nstatvsingle, ' for a single crystal'
        end if
       else if (ngrains.gt.1) then
        if(mod(nstatv,nstatvpoly).gt.0) then
          print*, '=== WARNING IN SDVINIA === '
          print*,'nstatv is: ', nstatv, ' and is not a multiple of ',
     &      nstatvpoly
        else if (ngrains*nstatvpoly.ne.nstatv) then
          print*, '=== WARNING IN SDVINIA === '
          print*, 'simulation has ', ngrains, ' grains and nstatv is',
     &      nstatv
          print*, 'nstatv should be set to: ', nstatvpoly*ngrains
        end if
       else
        print*, '=== WARNING IN SDVINIA === '
        print*, 'ngrains should be a positive integer, but it is',
     &    ngrains
        print*, 'ngrains is controlled by initializing statev3'
       end if

C     Load angles for orientation from fortran in hardcode.f
      if (orient.le.3) then !load ang from fortran
        iseed = int(bigint*seed)
        if(orient.eq.1) then !az31 rolled data
```

```
50            call azrolledtex(angs)
51          else if (orient.eq.2) then !az31 ecae data
52            call azecaetex(angs)
53          else if (orient.eq.3) then !amx602 data
54            call amxtex(angs)
55          else
56            print*, '=== INVALID TEXTURE FILE CHOICE, EXITING === '
57            print*, '-- Change value of ORIENTATION in input file'
58            stop
59          end if
60        end if
61
62  c     single xtal
63        if (ngrains.eq.1) then
64          if (orient.ge.4) then
65            call single_xtal_orients(orient, phi1, theta, phi2)
66          else
67            itemp = i8_uniform(iseed, nmax)
68            phi1 = angs(itemp, 1)
69            theta = angs(itemp, 2)
70            phi2 = angs(itemp, 3)
71          end if
72          call create_rmat_bunge(phi1, theta, phi2, rmat)
73          statev(1) = rmat(1,1)
74          statev(2) = rmat(1,2)
75          statev(3) = rmat(1,3)
76          statev(4) = rmat(2,1)
77          statev(5) = rmat(2,2)
78          statev(6) = rmat(2,3)
79          statev(7) = rmat(3,1)
80          statev(8) = rmat(3,2)
81          statev(9) = rmat(3,3)
82  c     polyxtal
83        else
84          do i=1,ngrains
85            itemp = i8_uniform(iseed, nmax)
86            phi1 = angs(itemp, 1)
87            theta = angs(itemp, 2)
88            phi2 = angs(itemp, 3)
89            call create_rmat_bunge(phi1, theta, phi2, rmat)
90            statev((i-1)*nstatvpoly+1) =  rmat(1,1)
91            statev((i-1)*nstatvpoly+2) =  rmat(1,2)
92            statev((i-1)*nstatvpoly+3) =  rmat(1,3)
93            statev((i-1)*nstatvpoly+4) =  rmat(2,1)
94            statev((i-1)*nstatvpoly+5) =  rmat(2,2)
95            statev((i-1)*nstatvpoly+6) =  rmat(2,3)
96            statev((i-1)*nstatvpoly+7) =  rmat(3,1)
97            statev((i-1)*nstatvpoly+8) =  rmat(3,2)
98            statev((i-1)*nstatvpoly+9) =  rmat(3,3)
99          end do
100       end if
101
102       return
103       end
104
105
106 c-----------------------------------------------------------------------
107 c  ABAQUS STATE VARIABLE INITIALIZATION SUBROUTINE
108 c-----------------------------------------------------------------------
109
110       SUBROUTINE SDVINI(STATEV,COORDS,NSTATV,NCRDS,NOEL,NPT,LAYER,KSPT)
111 C
112       IMPLICIT NONE
113       INTEGER NSTATV,NCRDS,NOEL,NPT,LAYER,KSPT
114       DOUBLE PRECISION STATEV, COORDS
```

```
      DIMENSION STATEV(NSTATV),COORDS(NCRDS)
C

      double precision phi1, theta, phi2 !Bunge Euler Angle (rad)
      double precision rmat(3,3)

      !TODO - CHANGE SO READS IN FROM ELSEWHERE

      if (NCRDS.le.3) then !load an angle file
        print*, '=== INVALID ORIENTATION FOR SINGLE XTAL, EXITING === '
        print*, '-- Note 1-3 are reserved for polyxtal deformation'
        print*, '-- Change value of ORIENTATION in input file'
        stop
      else
        call single_xtal_orients(NCRDS, phi1, theta, phi2)
      end if

      call create_rmat_bunge(phi1, theta, phi2, rmat)

      STATEV(1) = rmat(1,1)
      STATEV(2) = rmat(1,2)
      STATEV(3) = rmat(1,3)
      STATEV(4) = rmat(2,1)
      STATEV(5) = rmat(2,2)
      STATEV(6) = rmat(2,3)
      STATEV(7) = rmat(3,1)
      STATEV(8) = rmat(3,2)
      STATEV(9) = rmat(3,3)
      STATEV(10) = 0.0d+0 !gamma basal
      STATEV(11) = 0.0d+0 !gamma twin 1
      STATEV(12) = 0.0d+0 !gamma twin 2
      STATEV(13) = 0.0d+0 !gamma twin 3
      STATEV(14) = 0.0d+0 !gamma twin 4
      STATEV(15) = 0.0d+0 !gamma twin 5
      STATEV(16) = 0.0d+0 !gamma twin 6
      STATEV(17) = 0.0d+0 !epsilon non-basal slip
      STATEV(18) = 0.0d+0 !epsilondot non-basal slip
      STATEV(19) = 0.0d+0 !energy per unit reference volume
      STATEV(20) = 0.0d+0 !temperature stored in isv
      STATEV(21) = 0.0d+0 !epeff due to gambas
      STATEV(22) = 0.0d+0 !epeff due to gamtw
      STATEV(23) = 0.0d+0 !vf of all twinning
      STATEV(24) = 0.0d+0 !total effective plastic strain
      STATEV(25) = 0.0d+0 !total effective plastic strain rate

C
      RETURN
      END

c-----------------------------------------------------------------------
c  ABAQUS STATE VARIABLE INITIALIZATION SUBROUTINE - POLYCRYSTAL
c-----------------------------------------------------------------------

      SUBROUTINE SDVINIP(STATEV,COORDS,NSTATV,NCRDS,NOEL,NPT,LAYER,KSPT)
      IMPLICIT NONE

C     Variables required by abaqus
      INTEGER NSTATV,NCRDS,NOEL,NPT,LAYER,KSPT
      DOUBLE PRECISION STATEV, COORDS
      DIMENSION STATEV(NSTATV),COORDS(NCRDS)

C     Variables used for iteration
      double precision phi1, theta, phi2 !Bunge Euler Angle (rad)
      double precision rmat(3,3)
      integer I, NSTATVPG, NGRAINS
```

```
180
181 C     Variables associated with sampling random grains
182       integer u, norients, nmax, iseed, iseedinit, itemp, i8_uniform
183       parameter (u=20,nmax=10000, iseedinit=1111)
184       double precision angs(nmax, 3)
185
186 C     Load angles for orientation files if necessary
187       if (NCRDS.le.3) then !load an angle file
188         iseed = iseedinit
189         if(NCRDS.eq.1) then !az31 rolled data
190           open (u, FILE='orients/az31roll.txt', STATUS='OLD')
191         else if (NCRDS.eq.2) then !az31 ecae data
192           open (u, FILE='orients/az31ecae4bc.txt', STATUS='OLD')
193         else if (NCRDS.eq.3) then !amx602 data
194           open (u, FILE='orients/amx602extr.txt', STATUS='OLD')
195         else
196           print*, '=== INVALID TEXTURE FILE CHOICE, EXITING === '
197           print*, '-- Change value of ORIENTATION in input file'
198           stop
199         end if
200         read(u,*) norients
201         call inpdat(u,norients,nmax,angs)
202         close(u)
203       else
204         call single_xtal_orients(NCRDS, phi1, theta, phi2)
205       end if
206
207       NSTATVPG = 31
208       NGRAINS = NSTATV / NSTATVPG
209
210       DO I=1,NGRAINS
211
212         if (NCRDS.ne.0) then
213           itemp = i8_uniform(iseed, norients)
214           phi1 = angs(itemp, 1)
215           theta = angs(itemp, 2)
216           phi2 = angs(itemp, 3)
217           call create_rmat_bunge(phi1, theta, phi2, rmat)
218         end if
219
220         call create_rmat_bunge(phi1, theta, phi2, rmat)
221
222         STATEV((I-1)*NSTATVPG+1) = rmat(1,1)
223         STATEV((I-1)*NSTATVPG+2) = rmat(1,2)
224         STATEV((I-1)*NSTATVPG+3) = rmat(1,3)
225         STATEV((I-1)*NSTATVPG+4) = rmat(2,1)
226         STATEV((I-1)*NSTATVPG+5) = rmat(2,2)
227         STATEV((I-1)*NSTATVPG+6) = rmat(2,3)
228         STATEV((I-1)*NSTATVPG+7) = rmat(3,1)
229         STATEV((I-1)*NSTATVPG+8) = rmat(3,2)
230         STATEV((I-1)*NSTATVPG+9) = rmat(3,3)
231         STATEV((I-1)*NSTATVPG+10) = 0.0d+0 !gamma basal
232         STATEV((I-1)*NSTATVPG+11) = 0.0d+0 !gamma twin 1
233         STATEV((I-1)*NSTATVPG+12) = 0.0d+0 !gamma twin 2
234         STATEV((I-1)*NSTATVPG+13) = 0.0d+0 !gamma twin 3
235         STATEV((I-1)*NSTATVPG+14) = 0.0d+0 !gamma twin 4
236         STATEV((I-1)*NSTATVPG+15) = 0.0d+0 !gamma twin 5
237         STATEV((I-1)*NSTATVPG+16) = 0.0d+0 !gamma twin 6
238         STATEV((I-1)*NSTATVPG+17) = 0.0d+0
239         STATEV((I-1)*NSTATVPG+18) = 0.0d+0
240         STATEV((I-1)*NSTATVPG+19) = 0.0d+0
241         STATEV((I-1)*NSTATVPG+20) = 0.0d+0
242         STATEV((I-1)*NSTATVPG+21) = 0.0d+0
243         STATEV((I-1)*NSTATVPG+22) = 0.0d+0
244         STATEV((I-1)*NSTATVPG+23) = 0.0d+0
245         STATEV((I-1)*NSTATVPG+24) = 0.0d+0
```

```
246           STATEV((I-1)*NSTATVPG+25) = 0.0d+0
247           STATEV((I-1)*NSTATVPG+26) = 0.0d+0  !stress comp 1
248           STATEV((I-1)*NSTATVPG+27) = 0.0d+0  !stress comp 2
249           STATEV((I-1)*NSTATVPG+28) = 0.0d+0  !stress comp 3
250           STATEV((I-1)*NSTATVPG+29) = 0.0d+0  !stress comp 4
251           STATEV((I-1)*NSTATVPG+30) = 0.0d+0  !stress comp 5
252           STATEV((I-1)*NSTATVPG+31) = 0.0d+0  !stress comp 6
253
254         END DO
255
256         RETURN
257         END
258
259   c=====================================================================
260   c=====================================================================
261   c     Determines the bunge angles based on an integer input NCRDS
262   c---------------------------------------------------------------------
263
264         subroutine single_xtal_orients(NCRDS, phi1, theta, phi2)
265         implicit none
266
267   c     input
268         integer NCRDS
269
270   c     output
271         double precision phi1, theta, phi2
272
273   c     util
274         double precision pi
275
276         pi = 3.141592653589793d+0
277
278         if (NCRDS.eq.4) then
279   c     c axis
280   c     HOSFORD ---  a (P13), e(32), f(P31)
281           phi1 = 0.0d+0
282           theta = 0.0d+0
283           phi2 = 0.0d+0
284         else if (NCRDS.eq.5) then
285   c     HOSFORD ---  b (P13)
286           phi1 = pi/6.0d+0 !30 deg
287           theta = 0.0d+0
288           phi2 = 0.0d+0
289         else if (NCRDS.eq.6) then
290   c     HOSFORD ---  c (P13)
291           phi1 = 0.0d+0
292           theta = pi/2.0d+0 !90 deg
293           phi2 = 0.0d+0
294         else if (NCRDS.eq.7) then
295   c     HOSFORD ---  d (P13)
296           phi1 = 0.0d+0
297           theta = pi/2.0d+0 !90 deg
298           phi2 = pi/6.0d+0 !30 deg
299         else if (NCRDS.eq.8) then
300   c     HOSFORD ---  g (P13)
301           phi1 = pi/2.0d+0 !90 deg
302           theta = pi/4.0d+0  !45 deg
303           phi2 = -pi/2.0d+0  !-90 deg
304         else if (NCRDS.eq.9) then
305   c     slightly off c axis
306           phi1 = 0.0d+0
307           theta = 0.1d+0
308           phi2 = 0.0d+0
309         else if (NCRDS.eq.10) then
310   c     random single xtal
```

```
311         phi1 = 1.4d+0
312         theta = 0.7d+0
313         phi2 = 0.2d+0
314       else if (NCRDS.eq.11) then
315         phi1 = 0.0d+0
316         theta = pi/4.0d+0 !45 deg
317         phi2 = pi/6.0d+0 !30 deg
318       else if (NCRDS.eq.12) then
319 c       ecae idealized texture
320 c       143, 285, 0 or (37, 75, 0)
321         phi1 = 2.4958d+0
322         theta = 4.9742d+0
323         phi2 = 0.0d+0
324       else
325         print*, '=== INVALID TEXTURE FILE CHOICE, EXITING === '
326         print*, '-- Change value of ORIENTATION in input file'
327         stop
328       end if
329
330       return
331       end
332
333
334
335       subroutine pumat( stress,  statev,  ddsdde,  sse,     spd,
336      &                  scd,     rpl,     ddsddt,  drplde,  drpldt,
337      &                  strain,  dstrain, time,    dtime,   temp,
338      &                  dtemp,   predef,  dpred,   cmname,  ndi,
339      &                  nshr,    ntens,   nstatv,  props,   nprops,
340      &                  coords,  drot,    pnewdt,  celent,  dfgrd0,
341      &                  dfgrd1,  noel,    npt,     layer,   kspt,
342      &                  kstep,   kinc )
343
344       implicit none
345
346 c     -- UMAT DELCARATIONS --
347       ! Dimension variables passed into the UMAT sub (not all are used)
348       integer ndi      ! Number of direct stress components
349       integer nshr     ! Number of shear stress components
350       integer ntens    ! Size of stess or stran array (ndi + nshr)
351       integer nstatv   ! Number of SDVs
352       integer nprops   ! Number of material constants
353       integer noel     ! Element number
354       integer layer    ! Layer number (for composites)
355       integer kspt     ! Section point number within layer
356       integer kstep    ! Step number
357       integer kinc     ! Increment number
358       integer npt      ! Integration point number
359       character*7 cmname ! Material name
360       double precision sse ! Specific elastic stain energy
361
362       double precision
363      & celent,         ! Characteristic element length
364      & dtime,          ! Time increment
365      & temp,           ! Temperature at start of increment
366      & dtemp,          ! Temperature increment
367      & pnewdt,         ! Ratio of new time increment to time
368                        ! increment being used
369      & spd,            ! Specific plastic dissipation
370      & scd,            ! Specific creep dissipation
371      & rpl,            ! Volumetic heat generation per unit time
372      & drpldt,         ! Varation of rpl with temperature
373      & coords(3),      ! Coordinates of Gauss pt. being evaluated
374      & ddsdde(ntens,ntens), ! Tangent Stiffness Matrix
375      & ddsddt(ntens),  ! Change in stress per change in temperature
```

```
376       & dfgrd1(3,3),    ! Deformation gradient at end of step
377       & dfgrd0(3,3),    ! Deformation gradient at beginning of step
378       & dpred(1),       ! Change in predefined state variables
379       & drplde(ntens),  ! Change in heat generation per change in strain
380       & drot(3,3),      ! Rotation matrix
381       & dstrain(ntens), ! Strain increment tensor stored in vector form
382       & predef(1),      ! Predefined state vars dependent on field
383                         ! variables
384       & props(nprops),  ! Material properties passed in
385       & statev(nstatv), ! State Variables
386       & strain(ntens),  ! Strain tensor stored in vector form
387       & stress(ntens),  ! Cauchy stress tensor stored in vector form
388       & time(2)         ! Step Time and Total Time
389
390   c     -- UTIL --
391         integer i,j, ii, jj, nstatvpg, nstatvreg, ngrains
392         parameter (nstatvreg=25, nstatvpg=31)
393         double precision stressg(6), statevg(nstatvreg), ddsddeg(6,6)
394         double precision temp0, tempg
395
396         ngrains = nstatv / nstatvpg
397
398   c     initialize vars that will be averaged over
399         temp0 = temp
400         temp = 0.0d+0
401         do i=1,6
402           stress(i) = 0.0d+0
403           do j=1,6
404             ddsdde(i,j) = 0.0d+0
405           end do
406         end do
407
408   c     loop over grains
409         do i=1,ngrains
410   c       initialize state variables for each grain
411           do j=1,nstatvreg
412             statevg(j) = statev((i-1)*nstatvpg+j)
413           end do
414
415   c       access stored stress components
416           stressg(1) = statev((i-1)*nstatvpg+(nstatvreg+1))
417           stressg(2) = statev((i-1)*nstatvpg+(nstatvreg+2))
418           stressg(3) = statev((i-1)*nstatvpg+(nstatvreg+3))
419           stressg(4) = statev((i-1)*nstatvpg+(nstatvreg+4))
420           stressg(5) = statev((i-1)*nstatvpg+(nstatvreg+5))
421           stressg(6) = statev((i-1)*nstatvpg+(nstatvreg+6))
422
423   c       initialize temperature to avg init temperature
424           tempg = temp0
425
426   c       call umat
427           call cumat(stressg,  statevg,  ddsddeg,  sse,     spd,
428       &    scd,     rpl,     ddsddt,  drplde,  drpldt,
429       &    strain,  dstrain, time,    dtime,   tempg,
430       &    dtemp,   predef,  dpred,   cmname,  ndi,
431       &    nshr,    ntens,   nstatv,  props,   nprops,
432       &    coords,  drot,    pnewdt,  celent,  dfgrd0,
433       &    dfgrd1,  noel,    npt,     layer,   kspt,
434       &    kstep,   kinc)
435
436   c       record state variables from each grain
437           do j=1,nstatvreg
438             statev((i-1)*nstatvpg+j) = statevg(j)
439           end do
440   c       store calculated stress components
```

```
441           statev((i-1)*nstatvpg+nstatvreg+1) = stressg(1)
442           statev((i-1)*nstatvpg+nstatvreg+2) = stressg(2)
443           statev((i-1)*nstatvpg+nstatvreg+3) = stressg(3)
444           statev((i-1)*nstatvpg+nstatvreg+4) = stressg(4)
445           statev((i-1)*nstatvpg+nstatvreg+5) = stressg(5)
446           statev((i-1)*nstatvpg+nstatvreg+6) = stressg(6)
447
448   c       add up variables that are averaged over
449           do ii=1,6
450             stress(ii) = stress(ii) + stressg(ii)
451             do jj=1,6
452               ddsdde(ii,jj) = ddsdde(ii,jj) + ddsddeg(ii,jj)
453             end do
454           end do
455           temp = temp + tempg
456         end do
457
458   c     divide averaged quantities by number of grains
459         do ii=1,6
460           stress(ii) = stress(ii) / dble(ngrains)
461           do jj=1,6
462             ddsdde(ii,jj) = ddsdde(ii,jj)  / dble(ngrains)
463           end do
464         end do
465         temp = temp / dble(ngrains)
466
467         return
468         end
469
470   c=====================================================================
471   c=====================================================================
472   c   Reads entries open file u with n vars. Nmax is the max # of entries
473   c     possible to read, and x is the matrix data is stored in.
474   c
475   c   -- This is for reading in data files with 3 columns
476   c---------------------------------------------------------------------
477
478         subroutine inpdat(u,n,nmax,x)
479         implicit none
480
481         integer i,n,u,nmax
482         double precision x(nmax,3)
483
484         if (n.GT.nmax) then
485            write(*,*) 'Error: n = ', n, 'is larger than nmax =', nmax
486            goto 9999
487         endif
488
489         do i= 1, n
490            read(u,100) x(i,1), x(i,2), x(i,3)
491         end do
492     100 format (3(F10.4))
493
494         return
495    9999 stop
496         end
497
498   c=====================================================================
499   c=====================================================================
500   c   Given a seed, return a random integer n from 1 to max
501   c---------------------------------------------------------------------
502         function i8_uniform( seed, n)
503
504         implicit none
505
          integer i4_huge
```

```
506
507         parameter ( i4_huge = 2147483647 )
508         integer k
509         integer i8_uniform
510         integer seed, n
511
512         if (seed .eq. 0 ) then
513           print*, ' ----------------------'
514           print*, 'i8_uniform - Fatal error!'
515           print*, '  Input value of SEED = 0.'
516           stop
517         end if
518
519         k = seed / 127773
520
521         seed = 16807 * ( seed - k * 127773 ) - k * 2836
522
523         if ( seed .lt. 0 ) then
524           seed = seed + i4_huge
525         end if
526   c
527   c  Although SEED can be represented exactly as a 32 bit integer,
528   c  it generally cannot be represented exactly as a 32 bit real number!
529   c
530         i8_uniform = nint((dble(seed)*4.656612875D-10)*(n-1.0d+0))+1
531
532         return
533         end
```