```
!=====================================================================!
!                                                                     !
! Software Name : FrontISTR Ver. 3.4                                  !
!                                                                     !
!      Module Name : lib                                              !
!                                                                     !
!                    Written by Xi YUAN (AdavanceSoft)                !
!                               K. Satoh (Advancesoft)                !
!                                                                     !
!      Contact address :  IIS,The University of Tokyo, CISS          !
!                                                                     !
!      "Structural Analysis for Large Scale Assembly"                 !
!                                                                     !
!=====================================================================!
!=====================================================================!
!> ¥brief   This module manages calculation relates with materials
!!
!> ¥author     Xi YUAN (AdavanceSoft), K. Satoh (Advancesoft)
!> ¥date       2010/01/12
!> ¥version    0.00
!=====================================================================!

module m_MatMatrix

  use mMaterial
  use mMechGauss
  use m_ElasticLinear
  use mHyperElastic
  use m_ElastoPlastic
  use mViscoElastic
  use mCreep
  use mUElastic
  use mUmat

  implicit none
  INTEGER, PARAMETER, PRIVATE :: kreal = kind(0.0d0)

  contains

!> Fetch the nlgeom flag of the material
    integer function getNlgeomFlag( gauss )
      type( tGaussStatus ), intent(in) :: gauss      !> status of qudrature point
      getNlgeomFlag = gauss%pMaterial%nlgeom_flag
    end function

```

```
!> Calculate constituive matrix
    subroutine MatlMatrix( gauss, sectType, matrix, dt, cdsys, temperature )
      type( tGaussStatus ), intent(in) :: gauss          !> status of qudrature point
      INTEGER, INTENT(IN)              :: sectType       !> plane strain/stress or 3D
      REAL(KIND=kreal), INTENT(OUT)    :: matrix(:,:)    !> constitutive matrix
      REAL(KIND=kreal), INTENT(IN)     :: dt             !> time increment
      REAL(kind=kreal), INTENT(IN)     :: cdsys(3,3)     !> material coordinate system
      REAL(KIND=kreal), INTENT(IN), optional  :: temperature   !> temperature

      integer :: i
      real(kind=kreal)            :: cijkl(3,3,3,3)
      TYPE( tMaterial ), pointer  :: matl
      matl=>gauss%pMaterial

      if( matl%mtype==USERELASTIC ) then
        call uElasticMatrix( matl%variables(101:), gauss%strain, matrix )
      elseif( isViscoelastic(matl%mtype) ) then
        if( present(temperature) ) then
          call calViscoelasticMatrix( matl, sectTYPE, dt, matrix, temperature )
        else
          call calViscoelasticMatrix( matl, sectTYPE, dt, matrix )
        endif
      elseif( isElastic(matl%mtype) ) then
        i = getElasticType(gauss%pMaterial%mtype)
        if( i==0 ) then
          if( present(temperature) ) then
            call calElasticMatrix( matl, sectTYPE, matrix, temperature  )
          else
            call calElasticMatrix( matl, sectTYPE, matrix )
          endif
        elseif(  i==1 ) then
          if( present(temperature) ) then
            call calElasticMatrix_ortho( gauss%pMaterial, sectTYPE, cdsys, matrix, temperature )
          else
            call calElasticMatrix_ortho( gauss%pMaterial, sectTYPE, cdsys, matrix )
          endif
        else
          print *, "Elasticity type", matl%mtype, "not supported"
          stop
        endif
      elseif( matl%mtype==NEOHOOKE .or. matl%mtype==MOONEYRIVLIN ) then
        call calElasticMooneyRivlin( matl, sectType, cijkl, gauss%strain  )
        call mat_c2d( cijkl, matrix, sectType )
      elseif( matl%mtype==ARRUDABOYCE )  then
        call calElasticArrudaBoyce( matl, sectType, cijkl, gauss%strain )
```

```
91          call mat_c2d( cijkl, matrix, sectType )
92        elseif( matl%mtype==USERHYPERELASTIC )  then
93          call uElasticMatrix( matl%variables(101:), gauss%strain, matrix )
94        elseif( isElastoplastic(matl%mtype) )  then
95          if( present( temperature ) ) then
96            call calElastoPlasticMatrix( matl, sectType, gauss%stress,  &
97                      gauss%istatus(1), gauss%fstatus, matrix, temperature  )
98          else
99            call calElastoPlasticMatrix( matl, sectType, gauss%stress,  &
100                    gauss%istatus(1), gauss%fstatus, matrix  )
101         endif
102       elseif( matl%mtype==USERMATERIAL ) then
103         call uMatlMatrix( matl%name, matl%variables(101:), gauss%strain,  &
104             gauss%stress, gauss%fstatus, matrix, dt, gauss%ttime )
105       elseif( matl%mtype==NORTON ) then
106         if( present( temperature ) ) then
107           call iso_creep( matl, sectTYPE, gauss%stress, gauss%strain, gauss%fstatus,  &
108                       gauss%plstrain, dt, gauss%ttime, matrix, temperature  )
109         else
110            call iso_creep( matl, sectTYPE, gauss%stress, gauss%strain, gauss%fstatus,  &
111                       gauss%plstrain, dt, gauss%ttime, matrix  )
112         endif
113       else
114         stop "Material type not supported!"
115       endif
116
117     end subroutine
118
119 !
120 !> Update strain and stress for elastic and hyperelastic materials
121     subroutine StressUpdate( gauss, sectType, strain, stress, dt )
122       type( tGaussStatus ), intent(inout) :: gauss      !> status of qudrature point
123       integer, intent(in)                 :: sectType   !> plane strain/stress or 3D
124       real(kind=kreal), intent(in)        :: strain(6)  !> strain
125       real(kind=kreal), intent(out)       :: stress(6)  !> stress
126       real(kind=kreal), intent(in), optional  :: dt         !> time increment
127
128       select case ( gauss%pMaterial%mtype )
129         case ( NEOHOOKE, MOONEYRIVLIN )  ! Mooney-Livlin Hyperelastic material
130           call calUpdateElasticMooneyRivlin( gauss%pMaterial, sectType, strain, stress )
131         case ( ARRUDABOYCE )  ! Arruda-Boyce Hyperelastic material
132           call calUpdateElasticArrudaBoyce( gauss%pMaterial, sectType, strain, stress )
133         case ( USERHYPERELASTIC, USERELASTIC ) ! user-defined
134           call uElasticUpdate( gauss%pMaterial%variables(101:), strain, stress )
135         case ( VISCOELASTIC )
```

```
136             if( .not. present(dt) ) stop "error in viscoelastic update!"
137             call UpdateViscoelastic( gauss%pMaterial, sectType, strain, stress, gauss%fstatus,
138   dt )
139           case ( NORTON )
140             if( .not. present(dt) ) stop "error in viscoelastic update!"
141             call    update_iso_creep(    gauss%pMaterial,    sectType,    strain,    stress,
142   gauss%fstatus,gauss%plstrain, dt, gauss%ttime )
143           case ( USERMATERIAL)  ! user-defined
144             call uUpdate(  gauss%pMaterial%name, gauss%pMaterial%variables(101:),   &
145                strain, stress, gauss%fstatus, dt, gauss%ttime )
146         end select
147
148       end subroutine StressUpdate
149
150   !> Transfer rank 4 constituive matrix to rank 2 form
151   subroutine mat_c2d( cijkl, dij, itype )
152     real(kind=kreal),    intent(in)  :: cijkl(3,3,3,3)
153     real(kind=kreal),    intent(out) :: dij(6,6)
154     integer,             intent(in)  :: itype
155
156     dij(:,:) = 0.d0
157     SELECT CASE( itype )
158     CASE( D3 )
159       dij(1,1) = cijkl(1,1,1,1)   ! ---
160       dij(1,2) = cijkl(1,1,2,2)
161       dij(1,3) = cijkl(1,1,3,3)
162       dij(1,4) = cijkl(1,1,1,2)
163       dij(1,5) = cijkl(1,1,2,3)
164       dij(1,6) = cijkl(1,1,3,1)
165       dij(2,1) = cijkl(2,2,1,1)   ! ---
166       dij(2,2) = cijkl(2,2,2,2)
167       dij(2,3) = cijkl(2,2,3,3)
168       dij(2,4) = cijkl(2,2,1,2)
169       dij(2,5) = cijkl(2,2,2,3)
170       dij(2,6) = cijkl(2,2,3,1)
171       dij(3,1) = cijkl(3,3,1,1)   ! ---
172       dij(3,2) = cijkl(3,3,2,2)
173       dij(3,3) = cijkl(3,3,3,3)
174       dij(3,4) = cijkl(3,3,1,2)
175       dij(3,5) = cijkl(3,3,2,3)
176       dij(3,6) = cijkl(3,3,3,1)
177       dij(4,1) = cijkl(1,2,1,1)   ! ---
178       dij(4,2) = cijkl(1,2,2,2)
179       dij(4,3) = cijkl(1,2,3,3)
180       dij(4,4) = cijkl(1,2,1,2)
```

```
181       dij(4,5) = cijkl(1,2,2,3)
182       dij(4,6) = cijkl(1,2,3,1)
183       dij(5,1) = cijkl(2,3,1,1)   ! ---
184       dij(5,2) = cijkl(2,3,2,2)
185       dij(5,3) = cijkl(2,3,3,3)
186       dij(5,4) = cijkl(2,3,1,2)
187       dij(5,5) = cijkl(2,3,2,3)
188       dij(5,6) = cijkl(2,3,3,1)
189       dij(6,1) = cijkl(3,1,1,1)   ! ---
190       dij(6,2) = cijkl(3,1,2,2)
191       dij(6,3) = cijkl(3,1,3,3)
192       dij(6,4) = cijkl(3,1,1,2)
193       dij(6,5) = cijkl(3,1,2,3)
194       dij(6,6) = cijkl(3,1,3,1)
195     !
196       CASE( PlaneStress, PlaneStrain )
197         dij(1,1) = cijkl(1,1,1,1)   ! ---
198         dij(1,2) = cijkl(1,1,2,2)
199         dij(1,3) = cijkl(1,1,1,2)
200         dij(2,1) = cijkl(2,2,1,1)   ! ---
201         dij(2,2) = cijkl(2,2,2,2)
202         dij(2,3) = cijkl(2,2,1,2)
203         dij(3,1) = cijkl(1,2,1,1)   ! ---
204         dij(3,2) = cijkl(1,2,2,2)
205         dij(3,3) = cijkl(1,2,1,2)
206       CASE( AxisSymetric )
207         dij(1,1) = cijkl(1,1,1,1)
208         dij(1,2) = cijkl(1,1,2,2)
209         dij(1,3) = cijkl(1,1,1,2)
210         dij(1,4) = cijkl(1,1,3,3)
211         dij(2,1) = cijkl(2,2,1,1)
212         dij(2,2) = cijkl(2,2,2,2)
213         dij(2,3) = cijkl(2,2,1,2)
214         dij(2,4) = cijkl(2,2,3,3)
215         dij(3,1) = cijkl(1,2,1,1)
216         dij(3,2) = cijkl(1,2,2,2)
217         dij(3,3) = cijkl(1,2,1,2)
218         dij(3,4) = cijkl(1,2,3,3)
219         dij(4,1) = cijkl(3,3,1,1)
220         dij(4,2) = cijkl(3,3,2,2)
221         dij(4,3) = cijkl(3,3,1,2)
222         dij(4,4) = cijkl(3,3,3,3)
223       CASE( Shell )
224       END SELECT
225
```

```
226   end subroutine mat_c2d
227
228
229         ! (Gaku Hashimoto, The University of Tokyo, 2012/11/15) <
230   !####################################################################
231         SUBROUTINE MatlMatrix_Shell                          &
232                    (gauss, sectType, D,                      &
233                     e1_hat, e2_hat, e3_hat, cg1, cg2, cg3, &
234                     alpha)
235   !####################################################################
236
237         TYPE(tGaussStatus), INTENT(IN)  :: gauss
238         INTEGER, INTENT(IN)             :: sectType
239         REAL(KIND = kreal), INTENT(OUT) :: D(:, :)
240         REAL(KIND = kreal), INTENT(IN)  :: e1_hat(3), e2_hat(3), e3_hat(3)
241         REAL(KIND = kreal), INTENT(IN)  :: cg1(3), cg2(3), cg3(3)
242         REAL(KIND = kreal), INTENT(OUT) :: alpha
243
244   !--------------------------------------------------------------------
245
246         REAL(KIND = kreal)       :: c(3, 3, 3, 3)
247         TYPE(tMaterial), POINTER :: matl
248
249   !--------------------------------------------------------------------
250
251         matl => gauss%pMaterial
252
253   !--------------------------------------------------------------------
254
255         IF( isElastic(matl%mtype) ) THEN
256
257          CALL LinearElastic_Shell                     &
258               (matl, sectType, c,                     &
259                e1_hat, e2_hat, e3_hat, cg1, cg2, cg3, &
260                alpha)
261
262          CALL mat_c2d_Shell(c, D, sectType)
263
264         ELSE
265
266          STOP "Material type not supported!"
267
268         END IF
269
270   !--------------------------------------------------------------------
```

```
271
272         RETURN
273
274     !####################################################################
275         END SUBROUTINE MatlMatrix_Shell
276     !####################################################################
277         ! > (Gaku Hashimoto, The University of Tokyo, 2012/11/15)
278
279
280         ! (Gaku Hashimoto, The University of Tokyo, 2012/11/15) <
281     !####################################################################
282         SUBROUTINE mat_c2d_Shell(c, D, itype)
283     !####################################################################
284
285         REAL(KIND = kreal), INTENT(IN)  :: c(:, :, :, :)
286         REAL(KIND = kreal), INTENT(OUT) :: D(:, :)
287         INTEGER, INTENT(IN)             :: itype
288
289     !--------------------------------------------------------------------
290
291         INTEGER :: index_i(5), index_j(5), &
292                    index_k(5), index_l(5)
293         INTEGER :: i, j, k, l
294         INTEGER :: is, js
295
296     !--------------------------------------------------------------------
297
298         index_i(1) = 1
299         index_i(2) = 2
300         index_i(3) = 1
301         index_i(4) = 2
302         index_i(5) = 3
303
304         index_j(1) = 1
305         index_j(2) = 2
306         index_j(3) = 2
307         index_j(4) = 3
308         index_j(5) = 1
309
310         index_k(1) = 1
311         index_k(2) = 2
312         index_k(3) = 1
313         index_k(4) = 2
314         index_k(5) = 3
315
```

```
316         index_l(1) = 1
317         index_l(2) = 2
318         index_l(3) = 2
319         index_l(4) = 3
320         index_l(5) = 1
321
322   !-----------------------------------------------------------------
323
324         D(:, :) = 0.0D0
325
326   !-----------------------------------------------------------------
327
328         SELECT CASE( itype )
329         CASE( Shell )
330
331          DO js = 1, 5
332
333           DO is = 1, 5
334
335            i = index_i(is)
336            j = index_j(is)
337            k = index_k(js)
338            l = index_l(js)
339
340            D(is, js) = c(i, j, k, l)
341
342           END DO
343
344          END DO
345
346         END SELECT
347
348   !-----------------------------------------------------------------
349
350         RETURN
351
352   !#################################################################
353         END SUBROUTINE mat_c2d_Shell
354   !#################################################################
355         ! > (Gaku Hashimoto, The University of Tokyo, 2012/11/15)
356
357   end module m_MatMatrix
```