Brown University

ENGN2340

Final Project

Implement "Augmented Lagrangian" Hybrid Element

Yuhao Wang

December 16, 2015

1 Introduction

Volumetric Locking is exhibited by incompressible materials such as rubber having poisson's ratio near or equal to 0.5 resulting in an overly stiff response.

At each integration point of a fully integrated element the volume remains almost the constant and hence overconstraints the kinematically admissible displacement field. Consider an 8 noded 3D Hexahedral element. A Fully integrated element would have 8 integration points per element resulting in 8 constrains/element. But in this case only 3 d.o.f are available to satisfy these constraints. This results in overconstraining of mesh, also known as "Locking".

How to Avoid Volumetric Locking: 1) Use Reduced Integration: It has fewer volumetric constrains

2) Use of Selective Reduced Integration: It treats the volumetric and deviatoric parts of stiffness matrix separately.

3) Use of B-Bar method: Similar to selective reduced integration. But instead of separating volume integral into two parts, the definition of strain is modified.

4) Use of Hybrid Elements: They work by including the hydrostatic stress distribution as an additional unknown variable, which must be computed at the same time as the displacement field. This allows the stiff terms to be removed from the system of finite element equations.

5) Reduced Integration with Hourglass Control: Artificial stiffness is added to the element which constrains the hourglass mode.

2 "Augmented Lagrangian" Hybrid Element

In order to address simple hybrid element's issues of zero diagonals and only works for fully incompressibility, we need to use "Augmented Lagrangian" Hybrid Element. We allow some compressibility but still solve separately for pressure. For solving linear elastic problems, the weak form of the field equations are:

$$\int_v c^{dev}_{ijkl} \frac{\partial \eta_i}{\partial x_j} \frac{\partial u_k}{\partial x_l} \, \mathrm{d}V + \int_v p \frac{\partial \eta_i}{\partial x_i} \, \mathrm{d}V - \int_v b_i \eta_i \, \mathrm{d}V - \int_{s_2} t^*_i \eta_i \, \mathrm{d}A = 0$$
$$\int_v \left(k \frac{\partial u_k}{\partial x_k} - p\right) \frac{q}{k} \, \mathrm{d}V = 0$$

The FEA equations are:

$$K_{aibk}u_k^b + Q_{aib}p^b = F_i^a$$
$$Q_{akb}u_k^b - \Pi_{ab}p^b = 0$$
$$u_i^a = u_i^*(x_i^a)$$

While

$$K_{aibk} = \int_{V_e} c_{ijkl}\frac{\partial N^a(x)}{\partial x_j}\frac{\partial N^b(x)}{\partial x_l} - K\frac{\partial N^a(x)}{\partial x_i}\frac{\partial N^b(x)}{\partial x_l}\,\mathrm{d}V$$
$$Q_{aib} = \int_V \frac{\partial N^a(x)}{\partial x_i}M^b(x)\,\mathrm{d}V$$
$$\Pi_{ab} = \int_{V_e} \frac{1}{K}M^a(x)M^b(x)\,\mathrm{d}V$$

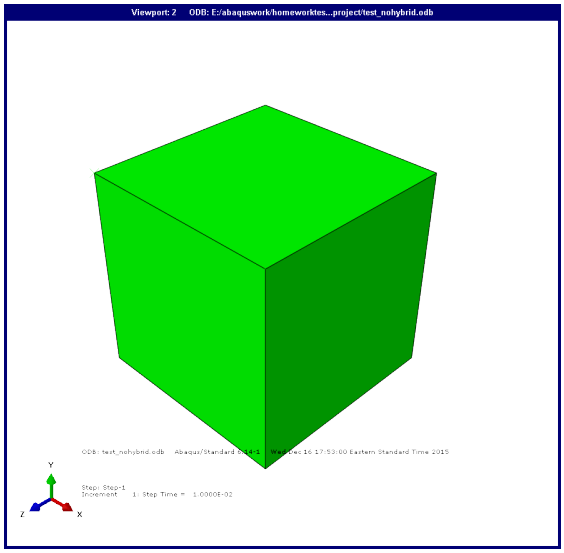
which $K = \frac{E}{3(1-2\mu)}$ is the bulk modulus. N is the shape function of node and M is the shape function of pressure points.

This type of element would work for both near incompressible materials and fully incompressible materials as long as we deal with the infinite bulk modulus caused by full incompressibility. Furthermore, it's suitable for all solvers since there's no zero on diagonal.

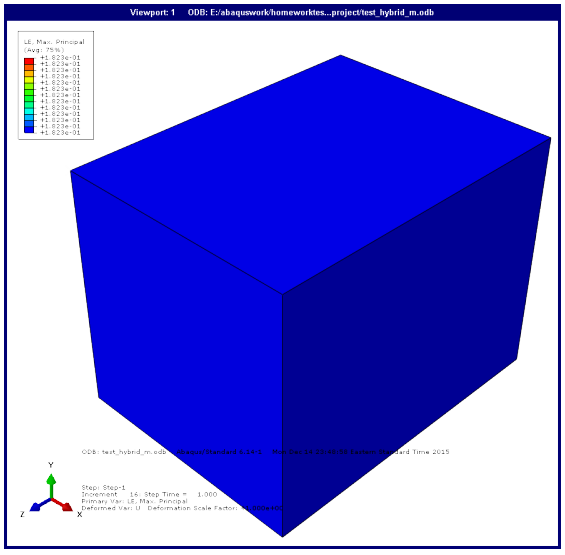
3 Implementing Elements in Abaqus UEL

Followed the procedure discussed in class, I implemented the element in Abaqus UEL. To test the element , I created both one-element input file and complex geometry input file.

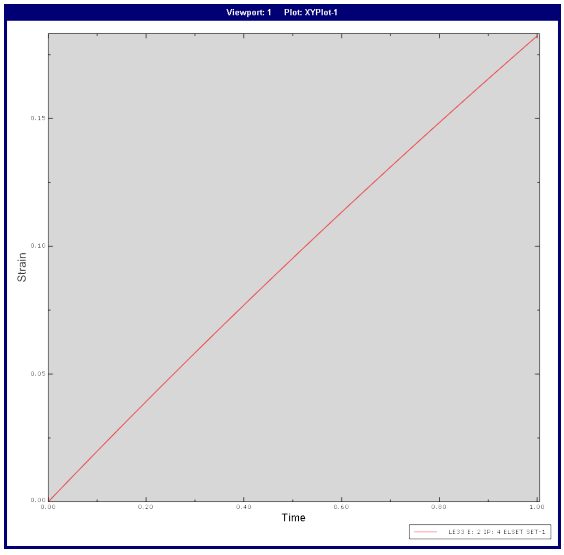
For the one element case: simple tension, set $\nu = 0.5 - 1^{-18}$, dimension 3*3*3, uniaxial tension -1 along z-axis. Figure(a) shows conventional element suffers from volumetric locking, Figure(b) shows result of hybrid element.



(a) convenrional element

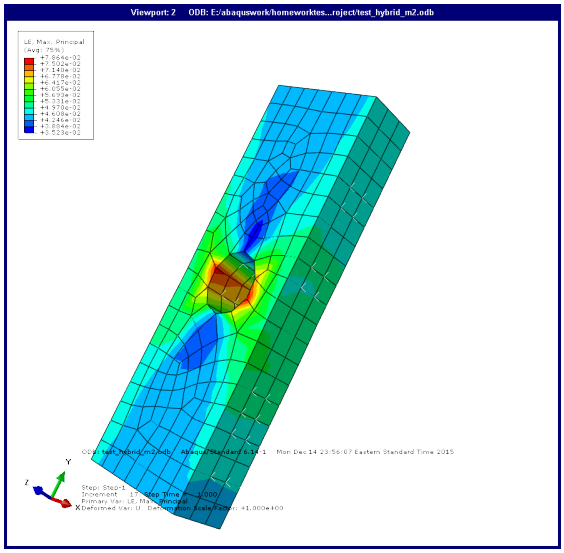


(b) hybrid element

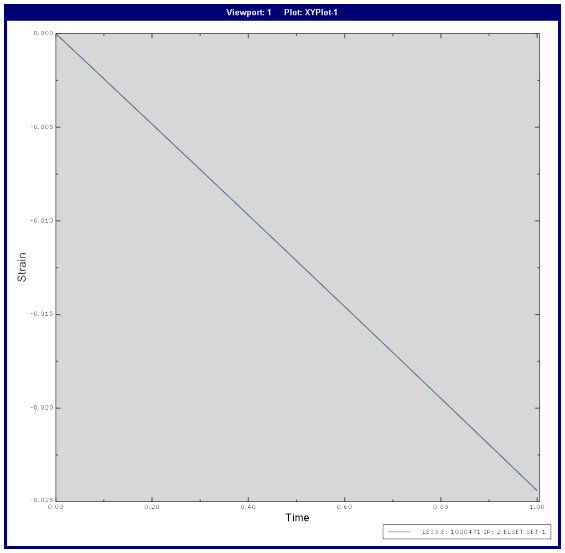


(c) hybrid element strain curve

For complex geometry case: simple tension, set$\nu = 0.4999$, dimension 60*20*10, uniaxial tension 3 on y-axis.



(d) contour plot of hybrid element



(e) strain curve of hybrid element

The implementation would solve the volumetric locking problem successfully.

4 Listing subroutine code piece

```
SUBROUTINE UEL(RHS,STIF,SVARS,ENERGY,NDOFEL,NRHS,NSVARS,
&          PROPS,NPROPS,COORDS,MCRD,NNODE,U,DU,V,A,JTYPE,TIME,DTIME,
&          KSTEP,KINC,JELEM,PARAMS,NDLOAD,JDLTYP,ADLMAG,PREDEF,NPREDF,
&          LFLAGS,MLVARX,DDLMAG,MDLOAD,PNEWDT,JPROPS,NJPROP,PERIOD)
C

INCLUDE 'ABA_PARAM.INC'
C       IMPLICIT DOUBLE PRECISION (A-H,O-Z)


double precision RHS(MLVARX,*),STIF(NDOFEL,NDOFEL),PROPS(*)
double precision SVARS(*),ENERGY(8),COORDS(MCRD,NNODE),U(NDOFEL)
double precision DU(MLVARX,*),V(NDOFEL),A(NDOFEL),TIME(2),
+                 PARAMS(*)
double precision ADLMAG(MDLOAD,*),DDLMAG(MDLOAD,*)
double precision PREDEF(2,NPREDF,NNODE)

double precision DTIME
double precision PNEWDT
double precision PERIOD

integer NDOFEL
integer NRHS
integer NSVARS
integer NPROPS
integer MCRD
integer NNODE
integer JTYPE
integer KSTEP
integer KINC
integer JELEM
integer NDLOAD
```

```
integer NPREDF
integer MLVARX
integer MDLOAD
integer NJPROP

integer LFLAGS(*),JPROPS(*)
integer JDLTYP(MDLOAD,*)
integer        n_points              ! number of integration points
integer  kint, i,j,k,l
C
C     Only relevant variables are commented. Refer to ABAQUS manual for
C
C     Variables with intent(out)
C     RHS          Residual Vector
C     STIF         Stiffness matrix
C
C     Variables with intent(in)
C     PROPS        Element property array for the Xu-Needleman model
C                  PROPS(1)    Mass density
C                  PROPS(2)    Rate of change of Angular speed
C                  PROPS(3:5)    n1:n3
C                  PROPS(6:8)    x01:x03
C     COORDS       Nodal coordinate array: COORD(J,N) is jth coord of nt
C     U            Total accumulated DOF array.  Contains accumulated di
C                  ordered as (u_i^1, u_i^2)
C     DU           Incremental displacements, ordered as
C                  DU(2*(N-1)+I,1) = u_i^n
C
C     NNODE        No. nodes on element.
C     NDOFEL       No. degrees of freedom for element (total)
C     NPROPS       No. real valued element properties
C     MLVARX       Dimensioning variable.
C     NRHS         No. RHS vectors.
C     MCRD         Largest of max value of COORDINATES parameter or acti
```

```
C
C

C     Local variables
double precision nodal_displacements(MCRD,NDOFEL/MCRD)
double precision COORDC(MCRD,NDOFEL/MCRD)
! Reshaped total displacement matrix
double precision xi(MCRD,27)          ! Integration points
double precision w(27)                ! Integration weights
double precision N(20)                ! Shape functions
double precision M(20)
double precision dNdxi(20,MCRD)       ! Shape function derivatives
double precision dNdx(20,MCRD)
double precision dxdxi(MCRD,MCRD)     ! Jacobian
double precision dxidx(MCRD,MCRD)     ! Jacobian inverse
double precision determinant          ! Jacobian determinant

double precision rho                  ! Mass density (per unit ref vol)
double precision Omega                ! Angular velocity
double precision axis(3)              ! Rotation axis vector
double precision x0(3)                ! Reference point vector for rotat
double precision xx(MCRD)              ! coord integration points
double precision int_dis(MCRD)         ! displacement integration points
double precision B(6,NDOFEL)
double precision bb
double precision rr(NDOFEL)
double precision km(NDOFEL,NDOFEL)
double precision temp_1
double precision temp_2
double precision residu(NDOFEL)
double precision stiff(NDOFEL,NDOFEL)
double precision DDSDDE(6,6)
double precision strain(6)
double precision stress(6)
```

```
double precision d44,d11,d12
integer planestrain,np_points
double precision E,xnu
double precision ddev(6,6)
double precision matrixpp(8,8),matrixup(NDOFEL,8),KK
double precision invpp(8,8),matrixm(6,20)

! Initialize the nodal displacements and nodal coordinates
nodal_displacements = reshape(U,(/MCRD,NDOFEL/MCRD/))

! Obtain current nodal coordinates
!


if (NPROPS<3) then
write(6,*) ' Not enough properties were provided for UEL '
write(6,*) ' NPROPS ',NPROPS
write(6,*) ' PROPS ',PROPS(1:nprops)
stop
endif


E = 2.d0*props(1)*(1.d0+props(2));
xnu = PROPS(2)
planestrain=PROPS(3)

if(xnu==0.5d0)then
xnu=(0.5d0)-1.d-18
end if
RHS(1:NDOFEL,1) = 0.d0
STIF(1:NDOFEL,1:NDOFEL) = 0.d0
residu = 0.d0
stiff = 0.d0
if (MCRD==2) then
```

```
if (nnode == 3) n_points = 4
if (nnode == 4) n_points = 4
if (nnode == 6) n_points = 4
if (nnode == 8) n_points = 4
if (nnode == 9) n_points = 9
else if (MCRD==3) then
if (nnode == 4) n_points = 1
if (nnode == 10) n_points = 4
if (nnode == 8) n_points = 8
if (nnode == 20) n_points = 27
else
write(6,*) ' Received strange number of coordinates in UEL '
stop
endif

if (MCRD == 1) then
np_points = nnode-1;
else if (MCRD == 2) then
if (nnode == 3)then
np_points = 1;
else if (nnode == 6)then
np_points = 3;

else if (nnode == 4) then
np_points = 1;

else if (nnode == 8) then
np_points = 4;
end if
elseif (MCRD == 3) then
if (nnode == 4) then
np_points = 1 ;

else if (nnode == 10)then
```

```
np_points = 4;

else if (nnode == 8) then
np_points = 1;

else if (nnode == 20) then
np_points = 8;
end if
end if



C
C       Set up integration points and weights
call initialize_integration_points(n_points,NNODE,MCRD,
+                                  xi(1:MCRD,1:n_points), w(1:n_points))

C
ddsdde=0.d0
d44 = 0.5D0*E/(1+xnu)
d11 = (1.D0-xnu)*E/( (1+xnu)*(1-2.D0*xnu) )
d12 = xnu*E/( (1+xnu)*(1-2.D0*xnu) )
DDSDDE(1:3,1:3) = d12
DDSDDE(1,1) = d11
DDSDDE(2,2) = d11
DDSDDE(3,3) = d11
DDSDDE(4,4) = d44
DDSDDE(5,5) = d44
DDSDDE(6,6) = d44

KK=E/(3.d0*(1.d0-2.d0*xnu))
DDEV=DDSDDE
DDEV(1:3,1:3)=DDSDDE(1:3,1:3)-KK
!         ddev(1:3,1:3)=-E/(3.d0*(1.d0+xnu))
!         DDEV(1,1)=2.d0*E/(1.d0+xnu)-E/(3.d0*(1.d0+xnu))
```

```
!         DDEV(2,2)=2.d0*E/(1.d0+xnu)-E/(3.d0*(1.d0+xnu))
!         DDEV(3,3)=2.d0*E/(1.d0+xnu)-E/(3.d0*(1.d0+xnu))
!         DDEV(4,4)=E/(1.d0+xnu)
!         DDEV(5,5)=E/(1.d0+xnu)
!         DDEV(6,6)=E/(1.d0+xnu)


matrixpp=0.d0
matrixup=0.d0
KK=0.d0

write(6,*)'d',U
write(6,*)'NDOFEL',NDOFEL
stiff=0.d0
DO kint=1,n_points
call calculate_shapefunctions(xi(1:MCRD,kint),nnode,MCRD,N,dNdxi)
dxdxi = matmul(COORDS(1:MCRD,1:nnode),dNdxi(1:nnode,1:MCRD))
call invert_small(MCRD,dxdxi,dxidx,determinant)
dNdx(1:nnode,1:MCRD) = matmul(dNdxi(1:nnode,1:MCRD),dxidx)
B=0.d0
B(1,1:3*nnode-2:3) = dNdx(1:nnode,1)
B(2,2:3*nnode-1:3) = dNdx(1:nnode,2)
B(3,3:3*nnode:3)   = dNdx(1:nnode,3)
B(4,1:3*nnode-2:3) = dNdx(1:nnode,2)
B(4,2:3*nnode-1:3) = dNdx(1:nnode,1)
B(5,1:3*nnode-2:3) = dNdx(1:nnode,3)
B(5,3:3*nnode:3)   = dNdx(1:nnode,1)
B(6,2:3*nnode-1:3) = dNdx(1:nnode,3)
B(6,3:3*nnode:3)   = dNdx(1:nnode,2)
strain = matmul(B,U(1:NDOFEL))
stress = matmul(ddsdde,strain)
residu(1:NDOFEL) =residu(1:NDOFEL)-
+ matmul(transpose(B(1:6,1:NDOFEL)),stress(1:6))*w(kint)*determinant
```

```
stiff(1:NDOFEL,1:NDOFEL)=stiff(1:NDOFEL,1:NDOFEL)+(matmul(transpose(B(1
+1:NDOFEL)),matmul(DDSDDE(1:6,1:6),B(1:6,1:NDOFEL)))-transpose(B(1:6,
+1:NDOFEL))*KK*B(1:6,1:NDOFEL))*w(kint)*determinant
END DO
write(6,*)'stiff',stiff
marixpp=0.d0
matrixup=0.d0
xi=0.d0
w=0.d0
call initialize_integration_points(np_points,NNODE,MCRD,
+                                  xi(1:MCRD,1:np_points), w(1:np_points))

write(6,*)'xi',xi
write(6,*)'w',w
DO kint=1,np_points
call calculate_shapefunctions(xi(1:MCRD,kint),nnode,MCRD,N,dNdxi)
dxdxi = matmul(COORDS(1:MCRD,1:nnode),dNdxi(1:nnode,1:MCRD))
call invert_small(MCRD,dxdxi,dxidx,determinant)
dNdx(1:nnode,1:MCRD) = matmul(dNdxi(1:nnode,1:MCRD),dxidx)
B=0.d0
B(1,1:3*nnode-2:3) = dNdx(1:nnode,1)
B(2,2:3*nnode-1:3) = dNdx(1:nnode,2)
B(3,3:3*nnode:3)   = dNdx(1:nnode,3)
B(4,1:3*nnode-2:3) = dNdx(1:nnode,2)
B(4,2:3*nnode-1:3) = dNdx(1:nnode,1)
B(5,1:3*nnode-2:3) = dNdx(1:nnode,3)
B(5,3:3*nnode:3)   = dNdx(1:nnode,1)
B(6,2:3*nnode-1:3) = dNdx(1:nnode,3)
B(6,3:3*nnode:3)   = dNdx(1:nnode,2)
M=0.d0
if (np_points>1) then
call calculate_shapefunctions(xi(1:MCRD,kint),np_points,MCRD,M,dMdxi)
else
M(1) = 1.d0;
```

```
end if

KK=E/(3.d0*(1.d0-2.d0*xnu))

matrixm=0.d0
do i=1,3
matrixm(i,1:np_points)=M(1:np_points)
end do
write(6,*)'Bmatrix',B
matrixup(1:NDOFEL,1:np_points)=matmul(transpose(B),matrixm(1:6,
+1:np_points))*w(kint)*determinant+matrixup(1:NDOFEL,1:np_points)

!         do i=1,nnode
!        do j=1,MCRD
!        do k=1,np_points
!          matrixup(MCRD*(i-1)+j,k)=matrixup(MCRD*(i-1)+j,k)+M(k)*dNdx(i,j
!       +w(kint)*determinant
!           end do
!           end do
!           end do

write(6,*)'dndx',dNdx
do i=1,np_points
do j=1,np_points
matrixpp(i,j)=matrixpp(i,j)-M(i)*M(j)*w(kint)*determinant/KK
end do
end do
END DO
write(6,*)'det',determinant
write(6,*)'kk',kk
!    matrixup=matrixup/3.d0
invpp=0.d0
if(np_points==1)then
invpp(1,1)=1.d0/matrixpp(1,1)
```

```
else
call invert_small(np_points,matrixpp(1:np_points,1:np_points),
+invpp(1:np_points,1:np_points),dt)
end if
write(6,*)'np_points',np_points
if(np_points==1)then
do i=1,NDOFEL
do j=1,NDOFEL
STIF(i,j)=stiff(i,j)-matrixup(i,1)*invpp(1,1)*matrixup(j,1)
end do
end do

else
STIF(1:NDOFEL,1:NDOFEL)=stiff(1:NDOFEL,1:NDOFEL)-
+matmul(matrixup(1:NDOFEL,1:np_points),matmul(invpp(1:np_points,
+1:np_points),transpose(matrixup(1:NDOFEL,1:np_points))))
end if
RHS(1:NDOFEL,1) = residu(1:NDOFEL)

!            write(6,*)'Bmatrix',B
write(6,*)'matrixup',matrixup
write(6,*)'matrixpp', matrixpp
!             write(6,*)'residu',residu
write(6,*)'stif',STIF
RETURN
END
```