Q-1

for solving this problem, we need to follow some steps:

Step 1] Forming Rotation Matrix (3×3) which will be used generating transformation of stiffness matrix (6×6).

Rotation Matrix might be generated in several ways; one way may be Euler bunge convention

Here, Reference basis (x, y, z) can rotate to the new basis (x_1, y_1, z_1) using three subsequent rotations.

$$(x, y, z) \xrightarrow[\phi_1]{z \text{ axis}} (U, V, z) \xrightarrow[\phi]{U \text{ axis}} (U, W, z_1) \xrightarrow[\phi_2]{z_1 \text{ axis}} (x_1, y_1, z_1)$$

First rotation: original z axis
second rotation: current x axis
third rotation: current z axis.

z - x - z rotation.
$R_1(\phi_1) \rightarrow R_2(\phi) \rightarrow R_3(\phi_2)$

rotation range, $\phi_1 \in [0, 2\pi]$

$\phi \in [0, \pi]$

$\phi_2 \in [0, 2\pi]$

Now three rotation matrix are,

$$R_1(\phi_1) = \begin{bmatrix} -\cos\phi_1 & \sin\phi_1 & 0 \\ -\sin\phi_1 & \cos\phi_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}; \quad R_2(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}$$

$$R_3(\phi_2) = \begin{bmatrix} \cos\phi_2 & \sin\phi_2 & 0 \\ -\sin\phi_2 & \cos\phi_2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Now overall Rotation Matrix (R); $R = R_1(\phi_1) \times R_2(\phi) \times R_3(\phi_2)$

$$R = \begin{bmatrix} \cos\phi_1\cos\phi_2 - \sin\phi_1\sin\phi_2\cos\phi & \sin\phi_1\cos\phi_2 + \cos\phi_1\sin\phi_2\cos\phi & \sin\phi_2\sin\phi \\ -\cos\phi_1\sin\phi_2 - \sin\phi_1\cos\phi_2\cos\phi & -\sin\phi_1\sin\phi_2 + \cos\phi_1\cos\phi_2\cos\phi & \cos\phi_2\sin\phi \\ \sin\phi_1\sin\phi & -\cos\phi_1\sin\phi & \cos\phi \end{bmatrix}$$

Euler

~~Solving for k~~

Here, the three subsequent rotation ~~is~~ used burge convention & ϕ_1, ϕ, ϕ_2 are euler angle which used burge convention at three rotation.

Now, for deriving the euler angle, if we know the different components of R(3×3) matrix, we can derive the three angle $\phi = \cos^{-1}(R(3,3))$ as well as other angle, ϕ_1 & ϕ_2

Subsequent Matlab code for getting R matrix is given below.

$R = \Omega \Rightarrow$ same

$$k^{(3)}_{3\times 3} = k^{3}_{ij} = \Omega_{\text{mod}(i+1,3)\,j}\;\Omega_{\text{mod}(i+2,3)\,\text{mod}(j+1,3)}$$

$$= \begin{bmatrix} R_{21}R_{31} & R_{22}R_{32} & R_{23}R_{33} \\ R_{11}R_{31} & R_{12}R_{32} & R_{13}R_{33} \\ R_{11}R_{21} & R_{12}R_{22} & R_{13}R_{3} \end{bmatrix}$$

Sometime Ω writing is boring for me so, i write R insted of Ω

$$k^{(4)}_{3\times 3} = k^{(4)}_{ij} = \Omega_{\text{mod}(i+1,3)\,\text{mod}(j+1,3)}\;\Omega_{\text{mod}(i+2,3)\,\text{mod}(j+2,3)} + \Omega_{\text{mod}(i+1,3)\,\text{mod}(j+2,3)}\;\Omega_{\text{mod}(i+2,3)\,\text{mod}(j+1,3)}$$

$$= \begin{bmatrix} R_{22}R_{33}+R_{23}R_{32} & R_{21}R_{33}+R_{23}R_{31} & R_{21}R_{32}+R_{22}R_{31} \\ R_{12}R_{33}+R_{13}R_{32} & R_{11}R_{33}+R_{13}R_{31} & R_{11}R_{32}+R_{12}R_{31} \\ R_{12}R_{23}+R_{13}R_{22} & R_{11}R_{23}+R_{13}R_{21} & R_{11}R_{22}+R_{12}R_{21} \end{bmatrix}$$

solve this by index notation - ?
Ask dr. Xiang ?

(3,2) element = ?

$$\text{mod}(i,3) = \begin{cases} i ; i \leq 3 \\ i-3 ; \quad i > 3 \end{cases}$$

Now, (2,2) element, $i=2$, $j=2$

$\Rightarrow 2\,\Omega_{i \bmod (j+1,3)}\,\Omega_{i \bmod (j+2,3)}$

↳ $2+1 \leq 3 \rightarrow i=2$; ↳ $2+2=4$, $2+3=5$

$= 2\,\Omega_2\,2\,\Omega_2$

Let's consider, (3,2) element

So, elasticity tensors transformation : new ↘ old/ref

$$C^{m}_{ijkl} = \Omega_{ip}\,\Omega_{jq}\,C^{e}_{pqrs}\,\Omega_{kr}\,\Omega_{ls}$$

i, j, k, l - free
p, q, r, s → dummy

Let Assume, $\boxed{K = \Omega_{ip}\,\Omega_{jp}}$ Here $\boxed{K = 6\times 6 \text{ Matrix}}$

m → new/rotated co-ordinate
e → old/ref co-ordinate

Now, $C^{m} = K C^{e} K^{T}$

Now, $K = \begin{bmatrix} k^{(1)}_{3\times3} & 2k^{(2)}_{3\times3} \\ k^{(3)}_{3\times3} & k^{4}_{3\times3} \end{bmatrix}_{6\times6}$

th position.

(1) $k_{3\times3} = \Omega_{ij}\,\Omega_{ij}$ → square

$= \Omega_{ij}^{2}$

2 → square / ^2

$$= \begin{bmatrix} \Omega_{11}^{2} & \Omega_{12}^{2} & \Omega_{13}^{2} \\ \Omega_{21}^{2} & \Omega_{22}^{2} & \Omega_{23}^{2} \\ \Omega_{31}^{2} & \Omega_{32}^{2} & \Omega_{33}^{2} \end{bmatrix}$$

$$2k^{(2)}_{3\times3} = 2k^{(2)}_{ij} = \Omega_{i\,\mathrm{mod}(j+1,3)}\,\Omega_{j\,\mathrm{mod}(j+2,3)}$$

$$= \begin{bmatrix} 2\Omega_{12}\Omega_{13} & 2\Omega_{11}\Omega_{13} & 2\Omega_{11}\Omega_{12} \\ 2\Omega_{21}\Omega_{23} & \boxed{2\Omega_{21}\Omega_{23}} & 2\Omega_{21}\Omega_{22} \\ 2\Omega_{32}\Omega_{33} & 2\Omega_{31}\Omega_{33} & 2\Omega_{31}\Omega_{32} \end{bmatrix}$$

Step 2: Transformation of stiffness Matrix's (T or Ω or G)

This T is a 6×6 Matrix which will be used to convert the stiffness matrix from original co-ordinate to rotated co-ordinate (global)

$$C^{new} = T^T \times C^{original} \times T$$

$C^{original}$ → original stiffness Matrix

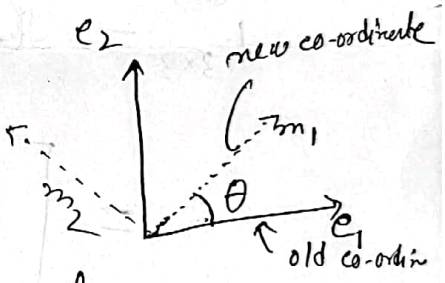
C^{new} → rotated global stiffness Matrix

Now, obtain this transformation Matrix is tricky since it is 6×6 Matrix

T → Transformation Matrix

This transformation Matrix is obtained from ~~two way~~ Rotation Matrix. Rotation Matrix might be calculated in two ways.

way-1 if original or reference stiffness tensor basis vector is (e_1, e_2, e_3) & Rotated stiffness tensor basis vector is (m_1, m_2, m_3) & and they are rotated at an angle θ



Here, Ω might be denoted as R

So, transformation tensor Ω_{ij} or R_{ij} might be

$$\Omega_{ij} = R_{ij} = \begin{bmatrix} m_1 \cdot e_1 & m_1 \cdot e_2 & m_1 \cdot e_3 \\ m_2 \cdot e_1 & m_2 \cdot e_2 & m_2 \cdot e_3 \\ m_3 \cdot e_1 & m_3 \cdot e_2 & m_3 \cdot e_3 \end{bmatrix}$$

Here $\Omega \Omega^T = \Omega^T \Omega = I$ orthogonal

Now, we can transform everything from current or reference/ local co-ordinate to rotated (global co-ordinate)

m → m_1, m_2, m_3 (new) co-ordinate

e → e_1, e_2, e_3 coordinate (reference)

Stress: $\sigma_{ij}^{m} = \Omega_{ik}\, \sigma_{kl}^{e}\, \Omega_{jl}$

(σ_{ij}: local/ref; σ_{kl}: rotated/current/global)

Strain: $\varepsilon_{ij}^{m} = \Omega_{ik}\, \varepsilon_{kl}\, \Omega_{jl}$

Thermal expansion: $\alpha_{ij}^{m} = \Omega_{ik}\, \alpha_{kl}^{m}\, \Omega_{jl}$

elasticity tensor: $C_{ijkl}^{m} = \Omega_{ip}\Omega_{jq}\, C_{pqrs}^{e}\, \Omega_{kr}\Omega_{ls}$

Final - k matrix

$$k = \begin{bmatrix} R_{11}^2 & R_{12}^2 & R_{13}^2 & 2R_{12}R_{13} & 2R_{11}R_{13} & 2R_{11}R_{12} \\ R_{21}^2 & R_{22}^2 & R_{23}^2 & 2R_{22}R_{23} & 2R_{21}R_{23} & 2R_{21}R_{22} \\ R_{31}^2 & R_{32}^2 & R_{33}^2 & 2R_{32}R_{33} & 2R_{31}R_{33} & 2R_{31}R_{32} \\ R_{21}R_{31} & R_{22}R_{32} & R_{23}R_{33} & R_{22}R_{33}+R_{23}R_{32} & R_{21}R_{33}+R_{23}R_{31} & R_{21}R_{32}+R_{22}R_{31} \\ R_{11}R_{31} & R_{12}R_{32} & R_{13}R_{33} & R_{12}R_{33}+R_{13}R_{32} & R_{11}R_{33}+R_{13}R_{31} & R_{11}R_{32}+R_{12}R_{31} \\ R_{11}R_{21} & R_{12}R_{22} & R_{13}R_{23} & R_{12}R_{23}+R_{13}R_{22} & R_{11}R_{23}+R_{13}R_{21} & R_{11}R_{22}+R_{12}R_{21} \end{bmatrix}$$

Here, we don't know the unit vector basic vector e_1, e_2, e_3 or, m_1, m_2, m_3 & its angle between them.

for R or Ω matrix we will use previous euler-bunge Rotation matrix

$$R = \Omega = R(\phi_1) \times R(\phi) \times R(\phi_2)$$

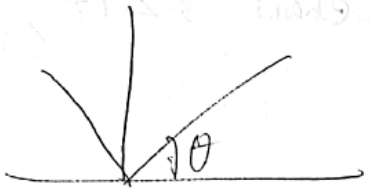
R → Rotation matrix

STEP

Ω = transformation tensor = rotation tensor = R

ϕ_1, ϕ, ϕ_2

1) $R^{new}_{3\times3} =$ [rotation matrix Euler bunge]



2) $k_{6\times6} =$ [will be derived from R]

3) $S^{new}_{6\times6} =$ [C_{11} ...]

(S → stiffness matrix − 6×6)

$S^{new} = k^T S^{rot} k$

Step-3 Stiffness Matrix for **Ag** : | BCC - body centre cube, FCC → face centre cube

3.2.16 → Bowles

c_{11}, c_{12}, c_{13}, c_{33}, c_{44}

Ag is a fcc Material & and only three constant are needed for constructing it's stiffness Matrix

Stiffness Matrix(c) for Linear Elastic cubic ~~Mag~~ Materials (3.2.16)

$$c = \begin{bmatrix} c_{11} & c_{12} & c_{12} \\ c_{12} & c_{11} & c_{12} \\ 0 & 0 & \\ 0 & 0 & \\ 0 & 0 & \\ 0 & 0 & \end{bmatrix}$$

we just need c_{11}, c_{12}, c_{44} to determine c.

for any fcc ~~Mv~~

$$c = \begin{bmatrix} c_{11} & c_{12} & c_{12} & 0 & 0 & 0 \\ c_{12} & c_{11} & c_{12} & 0 & 0 & 0 \\ c_{12} & c_{12} & c_{11} & 0 & 0 & 0 \\ 0 & 0 & 0 & c_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & c_{44} & 0 \\ 0 & 0 & 0 & 0 & 0 & c_{44} \end{bmatrix}$$

For Ag → chart 3.2.17, $c_{11} = 124.00$ GPa

$c_{44} = 46.10$ GPa

$c_{12} = 93.40$ GPa

90, 0, 0

cos

cos

sin

~~sin 0~~

$\sin\varphi_1 \cos\phi_2 + \cos\varphi_1 \sin\varphi_2 \cos\phi$

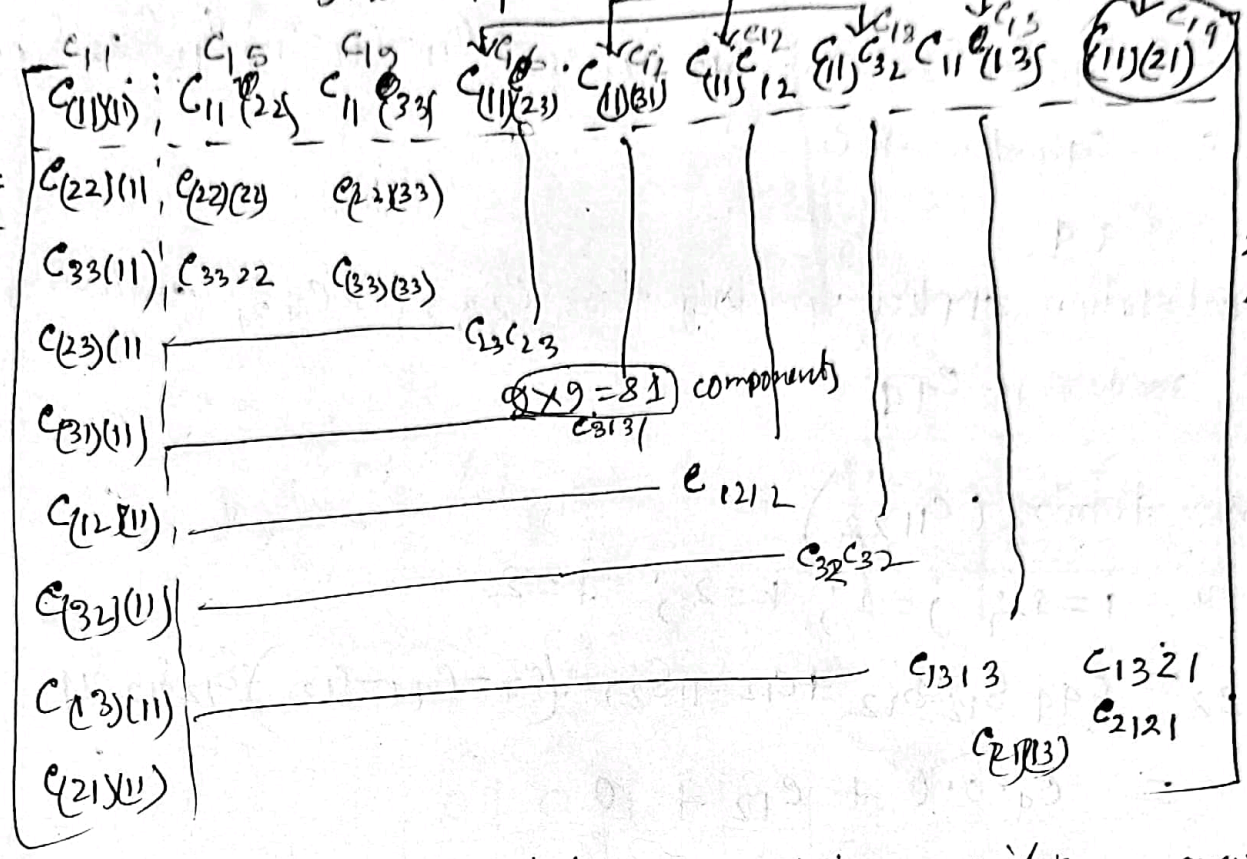
~~cos 90 cos~~

$\sin 90 \cos 0 + \sin 0 \cos 0$

$1 \times 1 + 0$

For cubic symmetry

$$c_{ijkl} = c_{44}\delta_{ik}\delta_{jl} + c_{12}\delta_{ij}\delta_{kl} + (c_{11} - c_{44} - c_{12})\delta_{ik}\delta_{jl}\delta_{ij}$$



we can test this by examining specific components.

Here, $11 = 1$, $22 = 2$, $33 = 3$, 44

① normal components → c_{1111}, c_{2222}

② Shear components → c_{1212}, c_{2323}

③ cross terms → c_{1122} (coupling between different normal stress)

For normal components (1), c_{1111} → $i=j=k=l=1$ | c_{ijkl} $i=l$, $k=j$

$$c_{ijkl} = c_{44}\delta_{ik}\delta_{jl} + c_{12}\delta_{ij}\delta_{kl} + (c_{11} - c_{44} - c_{12})\delta_{ik}\delta_{jl}\delta_{ij}$$

$$\Rightarrow c_{1111} = c_{44}\cdot 1 + c_{12}\cdot 1 + (c_{11} - c_{44} - c_{12})\, 1\cdot 1\cdot 1$$

$$\Rightarrow c_{1111} = c_{44} + c_{12} + c_{11} - c_{44} - c_{12}$$

$$c_{(11)(11)} = c_{11}$$

$(1,6)^{th}$ term

c_{11} c_{21} $i=1$

$0 = c_{44}\,0\,c_{12} + c_{11}$ $j=1$, $k=2$, $l=1$

$-c_{44} - c_{12}$

$c_{44}\times0 + c_{12}\times0 + (\ )\times0$

Same calculation for c_{2222}, c_{3333} proving that normal stress in each principle direction is indeed c_{11}

2. shear components: $(C_{1212}; C_{2323})$ $\frac{C_{ijkl}}{j=k;\ j=l}$

Here, $i=1;\ j=2,\ k=1,\ l=2$

$C_{1212} = \delta\, C_{44}\delta_{11}\delta_{22} + C_{12}\delta_{12}\delta_{12}\delta_{12} + (C_{11}-C_{44}-C_{12})\delta_{11}\delta_{22}\delta_{12}$

$= C_{44} + 0 + 0$

$= C_{44}$

This calculation applies similarly for C_{2323} & C_{3131} confirming the share modulous C_{44}

(3) cross terms (C_{1122}) $C_{ijkl} \mid i=j,\ k=l,\ i \neq k$

Here, $i=1;\ j=1;\ k=2;\ l=2$

$\Rightarrow C_{1122} = C_{44}\delta_{12}\delta_{12} + C_{12}\delta_{11}\delta_{22} + (C_{11}-C_{44}-C_{12})\delta_{12}\delta_{12}\delta_{11}$

$= C_{4}\cdot 0\cdot 0 + C_{12} + 0$

$= C_{12}$

This demonstrate that cross terms ; represents the interaction between different normal stresses are controlled by C_{12}

This breakdown validates that the given expression for C_{ijkl} correctly represents the stiffness tensor for a cubic symmetric material using three independent elastic ~~tensor~~ constant C_{11}, C_{12}, C_{44}. This tensor structure ensures that all necessary and sufficient conditions to describe the elastic behaviour of cubic crystal are meet.

i	k	l	C_{ijkl}
1	1	1	C_{1111}
1	1	2	C_{1112}
1	1	3	C_{1113}
1	2	1	C_{1121}
1	2	2	C_{1122}
1	2	3	C_{1123}
1	3	1	C_{1131}
1	3	2	C_{1132}
1	3	3	C_{1133}
2	1	1	C_{2111}
2	1	2	C_{2112}
2	1	3	C_{2113}
2	2	1	C_{2121}
2	2	2	C_{2122}
2	2	3	C_{2123}
2	3	1	C_{2131}
2	3	2	C_{2132}
2	3	3	C_{2133}
3	1	1	C_{3111}
3	1	2	C_{3112}
3	1	3	C_{3113}
3	2	1	C_{3121}
3	2	2	C_{3122}
3	2	3	C_{3123}
3	3	1	C_{3131}
3	3	2	C_{3132}
3	3	3	C_{3133}
1	1	1	C_{1111}
1	1	2	C_{1112}
1	1	3	C_{1113}
1	2	1	C_{1121}
1	2	2	C_{1122}
1	2	3	C_{1123}
1	3	1	C_{1131}
1	3	2	C_{1132}
1	3	3	C_{1133}
2	1	1	C_{2111}
2	1	2	C_{2112}
2	1	3	C_{2113}
2	2	1	C_{2121}
2	2	2	C_{2122}
2	2	3	C_{2123}
2	3	1	C_{2131}
2	3	2	C_{2132}
2	3	3	C_{2133}
2	1	1	C_{2211}
2	2	1	C_{2212}
2	1	3	C_{2213}
2	2	4	C_{2224}
2	2	2	C_{2222}
2	2	3	C_{2223}
2	3	1	C_{2231}
2	3	2	C_{2232}
2	3	3	C_{2233}
2	3	1	C_{2311}
2	3	2	C_{2312}
2	3	3	C_{2313}
2	3	4	C_{2321}
2	3	2	C_{2322}
2	3	3	C_{2323}
2	2	3	C_{2231}
2	3	3	C_{2332}
2	3	3	C_{2333}
3	1	1	C_{3111}
3	1	2	C_{3112}
3	1	3	C_{3113}
3	2	1	C_{3121}
3	2	2	C_{3122}
3	2	3	C_{3123}
3	3	1	C_{3131}
3	3	2	C_{3132}
3	3	3	C_{3133}
3	2	1	C_{3211}
3	2	2	C_{3212}
3	2	3	C_{3213}
3	2	1	C_{3221}
3	2	2	C_{3222}
3	2	3	C_{3223}
3	3	2	C_{3232}
3	3	2	C_{3232}
3	2	3	C_{3233}
3	1	1	C_{3311}
3	3	2	C_{3312}
3	3	3	C_{3313}
3	3	2	C_{3321}
3	3	2	C_{3322}
3	3	3	C_{3323}
3	3	1	C_{3331}
3	3	2	C_{3332}
3	3	3	C_{3333}

This is general 3D matrix 81 components

Vigot notation is different

$C_{11} \rightarrow 200$
$C_{12} \rightarrow 125$
$C_{aa} \rightarrow 75$

C_{ijks} 6.

Vigot

$C_{9\times9} =$

	1	2	3	4	5	6	7	8	9
1	168	0	0	0	21	0	0	0	121
2	0	168	0	75	0	0	0	0	0
3	0	0	168	0	0	0	75	0	0
4	0	75	168	168	0	0	0	0	0
5	121	0	0	0	168	0	0	0	121
6	0	0	0	0	168	168	0	75	0
7	0	0	75	0	0	0	168	0	0
8	0	0	0	0	0	75	0	168	0
9	121	0	0	0	121	0	0	0	168

$C_{11} \rightarrow xx, xx$ yy yy zz zz

$C_{12} \rightarrow xx-yy$ normal $xx-zz$ interaction

$C_{99} \rightarrow \frac{xy-yx}{yz-zy}$

$C_{6\times6} =$

168	121	121			
121	168	121			
121	121	168			
			75		
				75	
					75

Mapping (normal strs)

$C_6(1,1) = C_9(1,1)$
$C_6(2,4) = C_9(1,1)$
$C_6(3,3) = C_9(3,3)$

shear

$C_6(4,4) = C_9(8,8)$
$C_6(5,5) = C_9(7,7)$

if $c_{11} = 200$; $c_{12} = 125$, $c_{44} = 75$

$C_{9\times9} =$

200	0	0	0	125	0	0	0	125
0	75	0	0	0	0	0	0	0
0	0	75	75	0	0	0	0	0
0	0	0	75	0	0	0	0	125
125	0	0	0	200	0	0	0	0
0	0	0	0	0	75	0	0	0
0	0	0	0	0	0	75	0	0
0	0	0	0	0	0	0	75	0
125	0	0	0	125	0	0	0	200

$C_{6\times6} =$ (Voigt)

$$\begin{bmatrix} 200 & 125 & 125 & 0 & 0 & 0 \\ 125 & 200 & 0 & 0 & 0 & 0 \\ 125 & 0 & 200 & 0 & 0 & 0 \\ 0 & 0 & 0 & c_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & c_{44} & 0 \\ 0 & 0 & 0 & 0 & 0 & c_{44} \end{bmatrix}$$

→ $c_{11} \rightarrow c_{xxxx}, c_{yyyy}, c_{zzzz}$ → direct normal components

→ $c_{12} \rightarrow c_{xxyy}, c_{xxzz}$ $c_{zzxy} \rightarrow$ normal interaction

→ $c_{44} \rightarrow c_{xyyx}, c_{yzzy}, c_{zx,xz}$ → share position.

~~C_{ijkl}~~ Sodo code for generating of C_{ijkl}

(I) initiate the 4D stress tensor

$C = zeros(3, 3, 3, 3)$

(II) for i = 1:3

for j = 1:3

for k = 1:3

for l = 1:3

$C_{3333}, C_{2222}, C_{1111}$ (norma) ← if i == k && j == i && i == j

$C(i,j,k,l) = C11$

C_{2323}, C_{1212} ← elseif i == k && j = l

shear ← $C(i,j,k,l) = C44$

elseif i == j && k == l && i != k

C_{2233} C_{1122} ← $C(i,j,k,l) = C12$

end

end

end

end

Main equation

$$C_{ijkl} = C_{44}\delta_{ik}\delta_{jl} + C_{12}\delta_{ij}\delta_{kl} + (C_{11} - C_{44} - C_{12})\delta_{ik}\delta_{jl}\delta_{ij}$$

Another issue: $C_{11} = C_{1111} = C_{2222} = C_{3333}$

$C_{12} = C_{1122} = C_{1133} = C_{2233}$

$C_{44} = C_{2323} = C_{3131} = C_{1212}$

$C_{11} > C_{12} > C_{44}$

previous transformation matrix was 2D ~~9x9 = 81 elemen~~ 6x6 in size, But Here we are using 4D matrix (3x3x3x3). So, Although Euler bunge Rotation matrix is same, the transformation matrix will be different. and we will create this transformation matrix from rotation matrix The transformation tensor T for a stifness tensor C based on a rotation of matrix R should be constructed as.

$$T_{ijkl} = R_{ip} R_{jq} R_{kr} R_{ls}$$

T → transformation matrix
R → Rotation Matrix
C → Material stifness matrix

So,

$$C^{new}_{ijkl} = T_{ipjqkrls} C^{ref}_{pqrs}$$

(i → a free index
p, q, r, s → a dummy index)

Here, $T_{ipjqkrls}$ is essentially the outer product of R with itself four times, reindexed to cover all combinations correctly.

Code Implementation:

Firstly, I will try to modify the HW four code since previously Material Stiffness matrix was 6*6 in vigot notation now I need to transfer it to 3*3*3*3 4D matrix to make it compatible throughout the code So it is fourth order tensor. Then, I will run this code outside and validate it with the given angle from point four.

The main code for that part is given below:

```
clc
 clearvars
%euler_bunge convention
%taking three consecutive rotaion angle from user
phi_1 =input('Enter the phi_1 value: ');
phi = input('Enter the phi value: ');
phi_2 =input('Enter the phi_2 value: ');

%Getting the cubic symmetric stiffness matrix matrix or Ag
disp('Three Co-efficient of cubic symmetric stiffness matrix matrix of Ag:');
C11 =input('Enter the C11 value: ');
C12 = input('Enter the C12 value: ');
C44 =input('Enter the C44 value: ');

%converting  degree to radian
%matlab don't understand degree mode and neither transform it radian
%automaticly
t1=phi_1*pi/180;
t2=phi*pi/180;
t3=phi_2*pi/180;

%converting trichonomitric angle to suitable formet
%assuming c1,c2,c3, s1,s2,s3 respectively for smooth coding

c1=cos(t1);
c2=cos(t2);
c3=cos(t3);

s1=sin(t1);
s2=sin(t2);
s3=sin(t3);
```

```
%Rotation Matrix(3*3)
R =[ c1*c3-s1*s3*c2    s1*c3+c1*s3*c2    s3*s2;
   -c1*s3-s1*c3*c2   -s1*s3+c1*c3*c2    c3*s2;
         s1*s2            -c1*s2           c2  ];
%%creating the material stifness matrix (3*3*3*3) in reference configuration
% Initialize the 4D tensor
C_ref = zeros(3, 3, 3, 3);% later it will be suppressed
%C_ref = zeros(ndof,ncoord,ndof,ncoord); %stifness in reference co-ordiante


    % Populate the tensor according to cubic symmetry rules
    for i = 1:3
        for j = 1:3
            for k = 1:3
                for l = 1:3
                    if i == k && j == l && i == j %c1111
                        C_ref(i, j, k, l) =C_ref(i,j,k,l)+ C11;  % Normal
terms along axes
                    elseif i == k && j == l
                        C_ref(i, j, k, l) =C_ref(i,j,k,l)+ C44;  % Shear
terms on plane normals
                    elseif i == j && k == l && i ~= k %this will eliminate
c1111 from this
                        C_ref(i, j, k, l) =C_ref(i,j,k,l) + C12;  % Coupling
between different axes c1122
                    end
                end
            end
        end
    end



%%creating transformation matrix T(3*3*3*3*3*3*3*3) total 8D
%initialize the new Transformation matrix which will be generated from
roation matrix
T = zeros(3, 3, 3, 3, 3, 3, 3, 3);
    for i = 1:3
        for j = 1:3
            for k = 1:3
                for l = 1:3
                    for m = 1:3
                        for n = 1:3
                            for o = 1:3
                                for p = 1:3
                                    T(i, j, k, l, m, n, o, p) =  R(i, m) *
R(j, n) * R(k, o) * R(l, p);
                                end
                            end
                        end
                    end
                end
            end
        end
    end
```

```
C = zeros(3, 3, 3, 3);%later it will be suppressed
%C = zeros(ndof,ncoord,ndof,ncoord); %stifness in new co-ordiante
%C_ref = zeros(ndof,ncoord,ndof,ncoord); %stifness in reference coordiante

 % Apply the transformation tensor T to rotate the stiffness tensor C_ref
for i = 1:3
    for j = 1:3
        for k = 1:3
            for l = 1:3
                sum = 0; % Initialize sum for the contraction
                for m = 1:3
                    for n = 1:3
                        for o = 1:3
                            for p = 1:3
                                sum = sum + T(i, j, k, l, m, n, o, p) *
C_ref(m, n, o, p);
                            end
                        end
                    end
                end
                C(i, j, k, l) = sum; % Assign the computed value to the
rotated tensor
            end
        end
    end
end
R
C_ref
C
```

For the validation process, will put some known euler angle which the results is know.

1. (0,0,0)

This angle will not change the material stifness matrix and the rotation matrix will be an identity matrix. The running of the code will look like this

Editor - F:\Spring 2024\continum\cmprojects2\main_function.m

main_function.m

```
clc
clearvars
%euler_bunge convention
%taking three consecutive rotaion angle from user
phi_1 =input('Enter the phi_1 value: ');
phi = input('Enter the phi value: ');
phi_2 =input('Enter the phi_2 value: ');
```

Command Window

```
Enter the phi_1 value: 0
Enter the phi value: 0
Enter the phi_2 value: 0
Three Co-efficient of cubic symmetric stiffness matrix matrix of Cu:
Enter the C11 value: 225
Enter the C12 value: 153
Enter the C44 value: 115

R =

     1     0     0
     0     1     0
     0     0     1
```

So Ratation matrix is correct.

Now the stiffness matrix. Here C_ref means in reference co-ordinate material stiffness matrix. C means new orientation material stiffness matrix.

```
C_ref(:,:,2,1) =

     0     0     0
   115     0     0
     0     0     0


C_ref(:,:,3,1) =

     0     0     0
     0     0     0
   115     0     0


C_ref(:,:,1,2) =

     0   115     0
     0     0     0
     0     0     0
```

```
C(:,:,2,1) =

     0     0     0
   115     0     0
     0     0     0


C(:,:,3,1) =

     0     0     0
     0     0     0
   115     0     0


C(:,:,1,2) =

     0   115     0
     0     0     0
     0     0     0
```

This is correct

Let’s check another set

```
C_ref(:,:,3,2) =

     0     0     0
     0     0     0
     0   115     0


C_ref(:,:,1,3) =

     0     0   115
     0     0     0
     0     0     0


C_ref(:,:,2,3) =

     0     0     0
     0     0   115
     0     0     0


C_ref(:,:,3,3) =

   153     0     0
     0   153     0
     0     0   225
```

```
C_ref(:,:,3,2) =

     0     0     0
     0     0     0
     0   115     0


C_ref(:,:,1,3) =

     0     0   115
     0     0     0
     0     0     0


C_ref(:,:,2,3) =

     0     0     0
     0     0   115
     0     0     0


C_ref(:,:,3,3) =

   153     0     0
     0   153     0
     0     0   225
```

This is also correct

We can check all nine sets just like that. The code is giving correct results.

2.90.0.-90

The physical significance of this set is also similar to 0,0,0. No net rotation.

```
Editor - F:\Spring 2024\continum\cmprojects2\main_function.m
main_function.m
1 -     clc
2 -     clearvars
3       %euler bunge convention
```

```
Command Window
Enter the phi_1 value: 90
Enter the phi value: 0
Enter the phi_2 value: -90
Three Co-efficient of cubic symmetric stiffness matrix matrix of Cu:
Enter the C11 value: 225
Enter the C12 value: 153
Enter the C44 value: 115

R =

     1     0     0
     0     1     0
     0     0     1
```

The rotation matrix is an identity. So it is correct.

Now the output of the stiffness matrix:

```
C_ref(:,:,1,1) =

   225     0     0
     0   153     0
     0     0   153


C_ref(:,:,2,1) =

     0     0     0
   115     0     0
     0     0     0


C_ref(:,:,3,1) =

     0     0     0
     0     0     0
   115     0     0
```

```
C(:,:,1,1) =

   225     0     0
     0   153     0
     0     0   153


C(:,:,2,1) =

     0     0     0
   115     0     0
     0     0     0


C(:,:,3,1) =

     0     0     0
     0     0     0
   115     0     0
```

Results are correct

Let’s check another set of results.

```
C(:,:,3,2) =

     0     0     0
     0     0     0
     0   115     0


C(:,:,1,3) =

     0     0   115
     0     0     0
     0     0     0


C(:,:,2,3) =

     0     0     0
     0     0   115
     0     0     0


C(:,:,3,3) =

   153     0     0
     0   153     0
     0     0   225
```

```
C_ref(:,:,3,2) =

     0     0     0
     0     0     0
     0   115     0


C_ref(:,:,1,3) =

     0     0   115
     0     0     0
     0     0     0


C_ref(:,:,2,3) =

     0     0     0
     0     0   115
     0     0     0


C_ref(:,:,3,3) =

   153     0     0
     0   153     0
     0     0   225
```

The results are correct.

2.0,0,90

Y direction of the external frame will be aligned to X direction. Z axis remains the same. When there is a 90 degree of roation along any axis, the relative orientation between different axis remains the same since this is cubic symmetric. So no change in the stifness matrix

```
Enter the phi_1 value: 0
Enter the phi value: 0
Enter the phi_2 value: 90
Three Co-efficient of cubic symmetric stiffness matrix matrix of Cu:
Enter the C11 value: 225
Enter the C12 value: 153
Enter the C44 value: 115

R =

    0.0000    1.0000         0
   -1.0000    0.0000         0
         0         0    1.0000
```

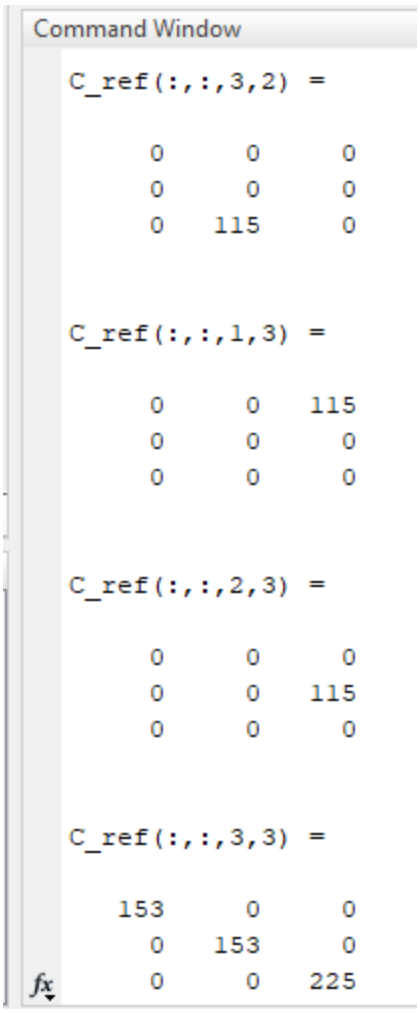
Lets check the results is same or not\?

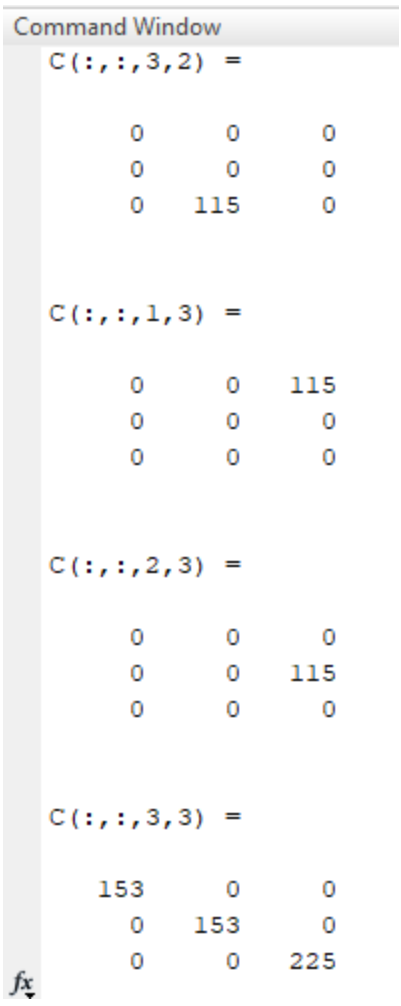
```
C(:,:,1,2) =

   -0.0000  115.0000         0
   -0.0000    0.0000         0
         0         0         0


C(:,:,2,2) =

  153.0000    0.0000         0
    0.0000  225.0000         0
         0         0  153.0000


C(:,:,3,2) =

     0     0     0
     0     0     0
     0   115     0


C(:,:,1,3) =

     0     0   115
     0     0     0
     0     0     0
```

Command Window

```
C_ref(:,:,2,1) =

     0     0     0
   115     0     0
     0     0     0


C_ref(:,:,3,1) =

     0     0     0
     0     0     0
   115     0     0


C_ref(:,:,1,2) =

     0   115     0
     0     0     0
     0     0     0


C_ref(:,:,2,2) =

   153     0     0
     0   225     0
     0     0   153
```

So the results are the same. Let's check another set





So the results of the code is correct.

4. 45,45,45

I am not sure about the any specific significance of that angle. The output of this angle is given below.

```
Command Window
  Enter the phi_1 value: 45
  Enter the phi value: 45
  Enter the phi_2 value: 45
  Three Co-efficient of cubic symmetric stiffness matrix matrix of Cu:
  Enter the C11 value: 225
  Enter the C12 value: 153
  Enter the C44 value: 115

  R =

      0.1464    0.8536    0.5000
     -0.8536   -0.1464    0.5000
      0.5000   -0.5000    0.7071
```

Output; results will be different

```
C_ref(:,:,1,1) =

   225     0     0
     0   153     0
     0     0   153


C_ref(:,:,2,1) =

     0     0     0
   115     0     0
     0     0     0


C_ref(:,:,3,1) =

     0     0     0
     0     0     0
   115     0     0


C_ref(:,:,1,2) =

     0   115     0
     0     0     0
     0     0     0
```

```
C(:,:,1,1) =

  242.4687    1.3438    9.5017
    1.3438  148.9687   -5.7010
    9.5017   -5.7010  139.5625


C(:,:,2,1) =

    1.3437   -4.0313   -5.7010
  110.9688    1.3438   -5.7010
   -5.7010   -5.7010   -2.6875


C(:,:,3,1) =

    9.5017   -5.7010  -13.4375
   -5.7010   -5.7010   -2.6875
  101.5625   -2.6875   -3.8007


C(:,:,1,2) =

    1.3437  110.9687   -5.7010
   -4.0313    1.3438   -5.7010
   -5.7010   -5.7010   -2.6875
```

Now lets discuss the implementation of this code in the large code of Solidmechnics.org website. First modification will be in the input syntax and set up a working directory, new materialstifness function looks like this.

FEM_2Dor3D_linelast_cubic_symmetry.m

```
133  function C = materialstiffness(ndof,ncoord,strain,materialprops)
134
135 -    mu = materialprops(1);
136 -    nu = materialprops(2);
137      %Three Co-efficient of cubic symmetric stiffness matrix matrix of Cu
138      %if you put this in gigapascle value change the scale factor at postprocessing function to make the displacement
139 -    C11 = 225e9;
140 -    C12 = 153e9;
141 -    C44 = 115e9;
142
143 -    C = zeros(ndof,ncoord,ndof,ncoord); %stifness in new co-ordiante initialization
144 -    C_ref = zeros(ndof,ncoord,ndof,ncoord); %stifness in reference co-ordiante initialization
145
146      %which set you want to ran just active it or vice versa
147 -    phi_1 =0; phi = 0; phi_2 =0;
148      %phi_1 =0; phi = 0; phi_2 =0;
149      %phi_1 =0; phi = 0; phi_2 =0;
150      %phi_1 =0; phi = 0; phi_2 =0;
151
152      %converting  degree to radian
153      %matlab don't understand degree mode and neither transform it radian
154      %automaticly
155 -    t1=phi_1*pi/180;
156 -    t2=phi*pi/180;
```

Editor - F:\Spring 2024\continum\cmprojects2\FEM_2Dor3D_linelast_cubic_symmetry.m

FEM_2Dor3D_linelast_cubic_symmetry.m

```
157 -        t3=phi_2*pi/180;
158
159          %converting trichonomitric angle to suitable formet
160          %assuming c1,c2,c3, s1,s2,s3 respectively for smooth coding
161
162 -        c1=cos(t1);
163 -        c2=cos(t2);
164 -        c3=cos(t3);
165
166 -        s1=sin(t1);
167 -        s2=sin(t2);
168 -        s3=sin(t3);
169
170          %Rotation Matrix(3*3)
171 -        R =[ c1*c3-s1*s3*c2     s1*c3+c1*s3*c2     s3*s2;
172             -c1*s3-s1*c3*c2    -s1*s3+c1*c3*c2     c3*s2;
173                   s1*s2              -c1*s2            c2  ]
174      %%creating the material stifness matrix (3*3*3*3) in reference configuration
175      % Initialize the 4D tensor
176
177 -       if (ncoord == 2)
178
179      %  planestrain = 0 => plane stress, planestrain = 1 => plane strain
180 -       planestrain = materialprops(3);
```

Editor - F:\Spring 2024\continum\cmprojects2\FEM_2Dor3D_linelast_cubic_symmetry.m

FEM_2Dor3D_linelast_cubic_symmetry.m

```
%%creating the material stifness matrix (3*3*3*3) in reference configuration
% Initialize the 4D tensor

   if (ncoord == 2)

%  planestrain = 0 => plane stress, planestrain = 1 => plane strain
   planestrain = materialprops(3);
     for i = 1:2
       for j = 1:2
         for k = 1:2
           for l = 1:2
             if (planestrain==1)
               if (i==j && k==l)
                   C(i,j,k,l) = C(i,j,k,l)+2*mu*nu/(1-2*nu);
               end
             else
               if (i==j && k==l)
                   C(i,j,k,l) = C(i,j,k,l)+2*mu*nu/(1-nu);
               end
             end
             if (i==l && k==j)
                 C(i,j,k,l) = C(i,j,k,l)+mu;
             end
             if (i==k && j==l)
                 C(i,j,k,l) = C(i,j,k,l)+mu;
             end
           end
         end
       end
     end

  elseif (ncoord == 3)
%change in code starts here
   % Populate the tensor according to cubic symmetry rules
   for i = 1:3
       for j = 1:3
           for k = 1:3
               for l = 1:3
                   if i == k && j == l && i == j %c1111
                       C_ref(i, j, k, l) =C_ref(i,j,k,l)+ C11;  % Normal terms along axes
                   elseif i == k && j == l
                       C_ref(i, j, k, l) =C_ref(i,j,k,l)+ C44;  % Shear terms on plane normals
                   elseif i == j && k == l && i ~= k %this will eliminate c1111 from this
                       C_ref(i, j, k, l) =C_ref(i,j,k,l) + C12;  % Coupling between different axes c1122
                   end
               end
```

```
                        C_ref(i, j, k, l) =C_ref(i,j,k,l) + C12;  % Coupling between different axes c1122
                    end
                end
            end
        end
    end

    C_ref

%%creating transformation matrix T(3*3*3*3*3*3*3*3) total 8D
%initialize the new Transformation matrix which will be generated from roation matrix
T = zeros(3, 3, 3, 3, 3, 3, 3, 3);
    for i = 1:3
        for j = 1:3
            for k = 1:3
                for l = 1:3
                    for m = 1:3
                        for n = 1:3
                            for o = 1:3
                                for p = 1:3
                                    T(i, j, k, l, m, n, o, p) =  R(i, m) * R(j, n) * R(k, o) * R(l, p);
                                end
                            end
```

FEM_2Dor3D_linelast_cubic_symmetry.m

```
                                    T(i, j, k, l, m, n, o, p) =  R(i, m) * R(j, n) * R(k, o) * R(l, p);
                                end
                            end
                        end
                    end
                end
            end
        end
    end
% Apply the transformation tensor T to rotate the stiffness tensor C_ref
            for i = 1:3
                for j = 1:3
                    for k = 1:3
                        for l = 1:3
                            sum = 0; % Initialize sum for the contraction
                            for m = 1:3
                                for n = 1:3
                                    for o = 1:3
                                        for p = 1:3
                                            sum = sum + T(i, j, k, l, m, n, o, p) * C_ref(m, n, o, p);
                                        end
                                    end
                                end
                            end
```

```
                                end
                            end
                            C(i, j, k, l) = sum; % Assign the computed value to the rotated tensor
                        end
                    end
                end
            end

            C
    end
end
%modification ends here
```

N.B. Follow the line number.

There are some minor changed in the post-processing file. Since the Stiffness constant are in giga pascle. So deformation might not be compatible for this size. So change this scale factor value by trial and error to get a visibale deformable plot over an undeformable plot.

```
101     %
102     %================================== POST-PROCESSING ==================================
103     %
104     % Create a plot of the deformed mesh
105     %
106
107 -     defcoords = zeros(ndof,nnode);
108 -     scalefactor = 1.0; % if the displacement is small change it
109 -     for i = 1:nnode
110 -       for j = 1:ndof
111 -          defcoords(j,i) = coords(j,i) + scalefactor*dofs(ndof*(i-1)+j);
112 -       end
113 -     end
114
```

I did not do any change in the input file (future work)

Calling the input file in the main code looked like this;

```
51 - infile=fopen('linear_elastic_brick8.txt','r');
52 - outfile=fopen('linear_elastic_brick8_results_zero_rotation.txt','w');
53
```

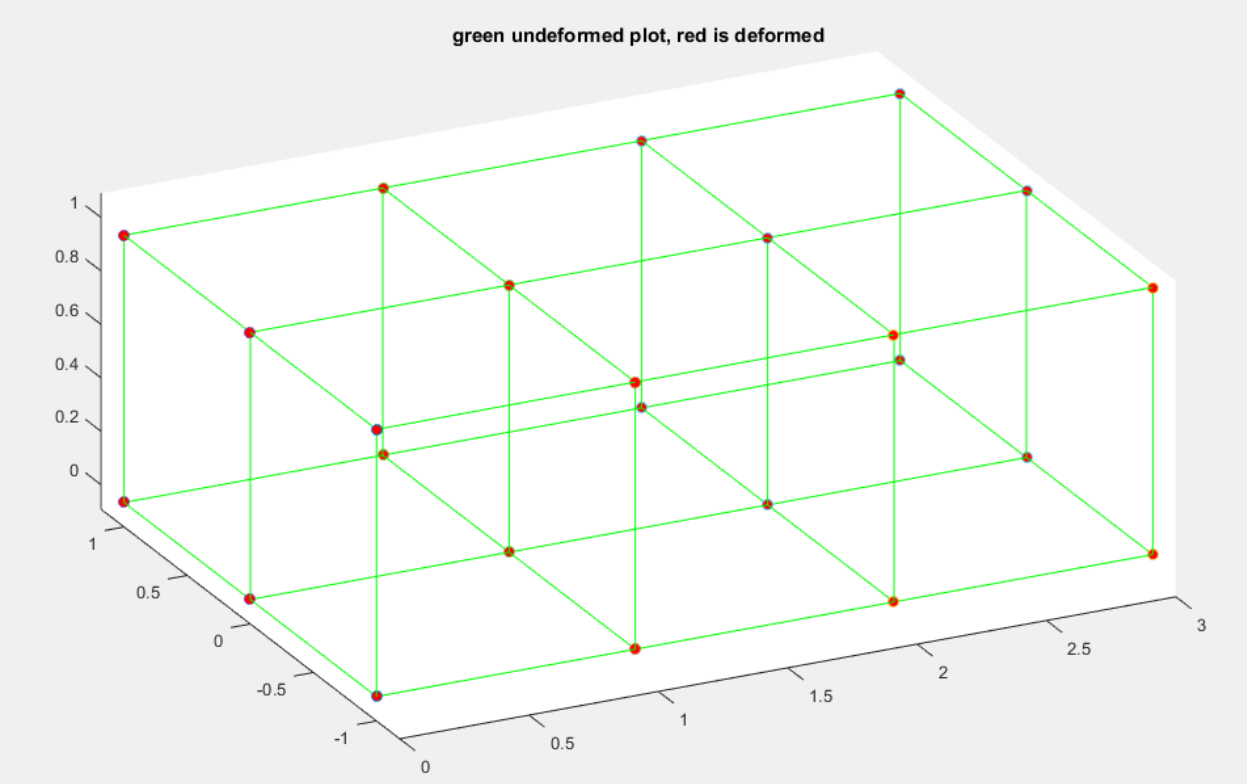
Now I ran the code for different euler angle just like the validation process. If I analyze the results it validate the code.

1. Output for (0,0,0) rotation.

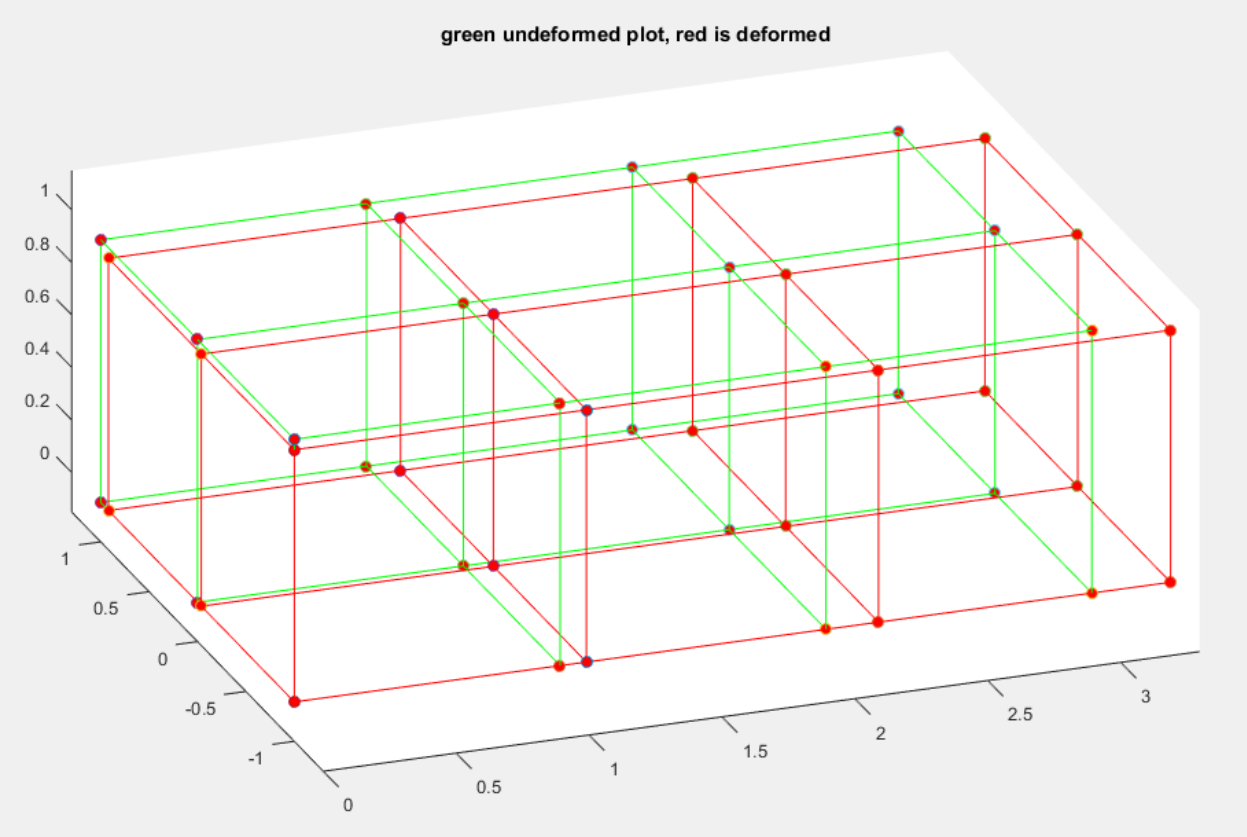
By trial and error, the scale factor was:

```
102     %================================== POST-PROCESSING ==================================
103     %
104     % Create a plot of the deformed mesh
105     %
106
107 -     defcoords = zeros(ndof,nnode);
108 -     scalefactor = 10000000000.0; % if the displacement is small change it
109 -     for i = 1:nnode
110 -       for j = 1:ndof
```

Now the undeformed plot:



Deformed over undeformed Plot:



Check for stiffness matrix. For (0,0,0)

```
R =

     1     0     0
     0     1     0
     0     0     1


C_ref(:,:,1,1) =

   1.0e+11 *

    2.2500         0         0
         0    1.5300         0
         0         0    1.5300


C_ref(:,:,2,1) =

   1.0e+11 *

         0         0         0
    1.1500         0         0
         0         0         0
```

```
C(:,:,1,1) =

   1.0e+11 *

    2.2500         0         0
         0    1.5300         0
         0         0    1.5300


C(:,:,2,1) =

   1.0e+11 *

         0         0         0
    1.1500         0         0
         0         0         0
```

Lets check other sets,

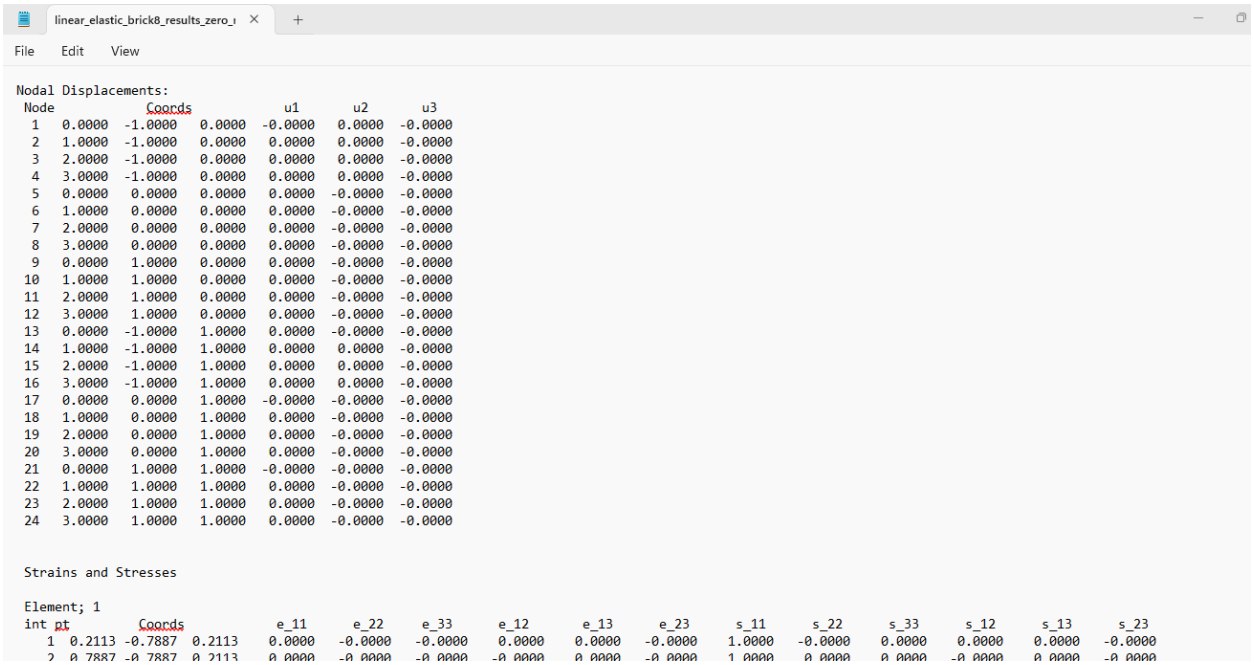
```
C(:,:,1,3) =

   1.0e+11 *

         0         0    1.1500
         0         0         0
         0         0         0


C(:,:,2,3) =

   1.0e+11 *

         0         0         0
         0         0    1.1500
         0         0         0


C(:,:,3,3) =

   1.0e+11 *

    1.5300         0         0
         0    1.5300         0
         0         0    2.2500
```

Command Window

```
C_ref(:,:,1,3) =

   1.0e+11 *

         0         0    1.1500
         0         0         0
         0         0         0


C_ref(:,:,2,3) =

   1.0e+11 *

         0         0         0
         0         0    1.1500
         0         0         0


C_ref(:,:,3,3) =

   1.0e+11 *

    1.5300         0         0
         0    1.5300         0
         0         0    2.2500
```

reference and new material stiffness matrix is correct for the zero Euler angle. So change in the stifness matrix.

Results for the stiffness matrix are given below. (details will be given at the appendix)



linear_elastic_brick8_results_zero_

File Edit View

Nodal Displacements:

Node	Coords			u1	u2	u3
1	0.0000	-1.0000	0.0000	-0.0000	0.0000	-0.0000
2	1.0000	-1.0000	0.0000	0.0000	0.0000	-0.0000
3	2.0000	-1.0000	0.0000	0.0000	0.0000	-0.0000
4	3.0000	-1.0000	0.0000	0.0000	0.0000	-0.0000
5	0.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000
6	1.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000
7	2.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000
8	3.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000
9	0.0000	1.0000	0.0000	0.0000	-0.0000	-0.0000
10	1.0000	1.0000	0.0000	0.0000	-0.0000	-0.0000
11	2.0000	1.0000	0.0000	0.0000	-0.0000	-0.0000
12	3.0000	1.0000	0.0000	0.0000	-0.0000	-0.0000
13	0.0000	-1.0000	1.0000	0.0000	-0.0000	-0.0000
14	1.0000	-1.0000	1.0000	0.0000	0.0000	-0.0000
15	2.0000	-1.0000	1.0000	0.0000	0.0000	-0.0000
16	3.0000	-1.0000	1.0000	0.0000	0.0000	-0.0000
17	0.0000	0.0000	1.0000	-0.0000	-0.0000	-0.0000
18	1.0000	0.0000	1.0000	0.0000	-0.0000	-0.0000
19	2.0000	0.0000	1.0000	0.0000	-0.0000	-0.0000
20	3.0000	0.0000	1.0000	0.0000	-0.0000	-0.0000
21	0.0000	1.0000	1.0000	-0.0000	-0.0000	-0.0000
22	1.0000	1.0000	1.0000	0.0000	-0.0000	-0.0000
23	2.0000	1.0000	1.0000	0.0000	-0.0000	-0.0000
24	3.0000	1.0000	1.0000	0.0000	-0.0000	-0.0000

Strains and Stresses

Element; 1

int pt	Coords			e_11	e_22	e_33	e_12	e_13	e_23	s_11	s_22	s_33	s_12	s_13	s_23
1	0.2113	-0.7887	0.2113	0.0000	-0.0000	-0.0000	0.0000	0.0000	-0.0000	1.0000	-0.0000	0.0000	0.0000	0.0000	-0.0000
2	0.7887	-0.7887	0.2113	0.0000	-0.0000	-0.0000	0.0000	0.0000	-0.0000	1.0000	-0.0000	0.0000	-0.0000	0.0000	-0.0000

Discussion: here the 0,0,0 euler angle doesn't have any net rotation. So we find the same results in the stiffness matrix reference and new orientation. Displacement stress-strain follows the linear elastic property of bcc or fcc material.

2. 90,0,0 output

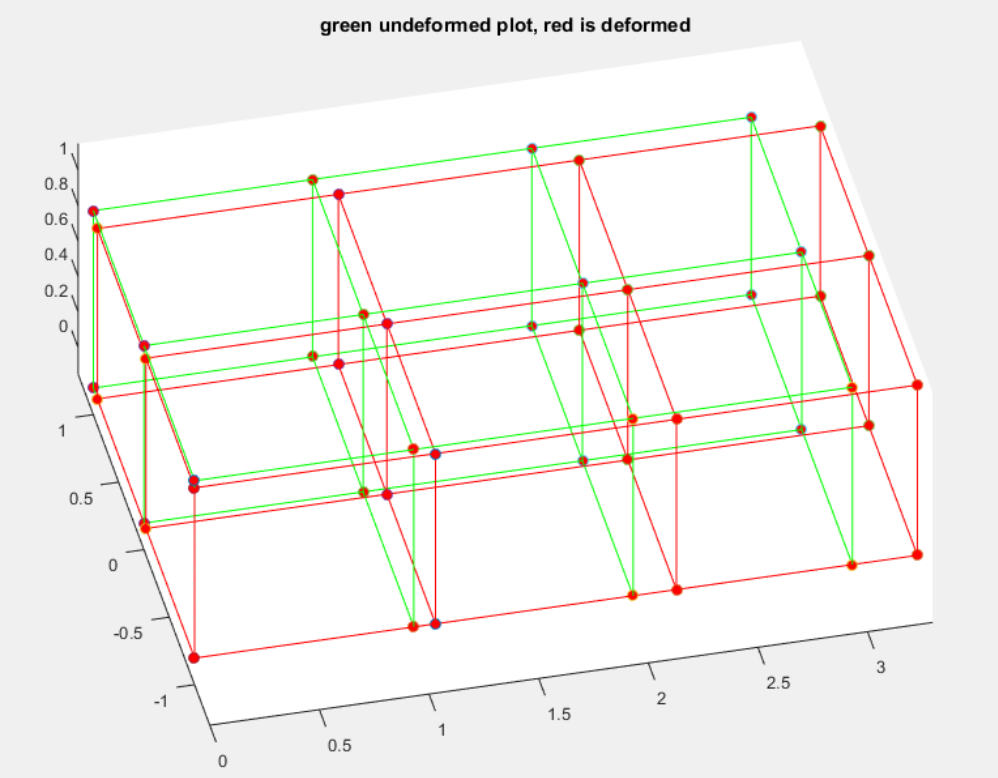
Validation:

```
R =

     1     0     0
     0     1     0
     0     0     1


C_ref(:,:,1,1) =

   1.0e+11 *

    2.2500         0         0
         0    1.5300         0
         0         0    1.5300


C_ref(:,:,2,1) =

   1.0e+11 *

         0         0         0
    1.1500         0         0
         0         0         0
```

```
C(:,:,1,1) =

   1.0e+11 *

    2.2500         0         0
         0    1.5300         0
         0         0    1.5300


C(:,:,2,1) =

   1.0e+11 *

         0         0         0
    1.1500         0         0
         0         0         0
```

Here the validation Process is correct

Deformed vs undeformed results (scale factor 100000000)

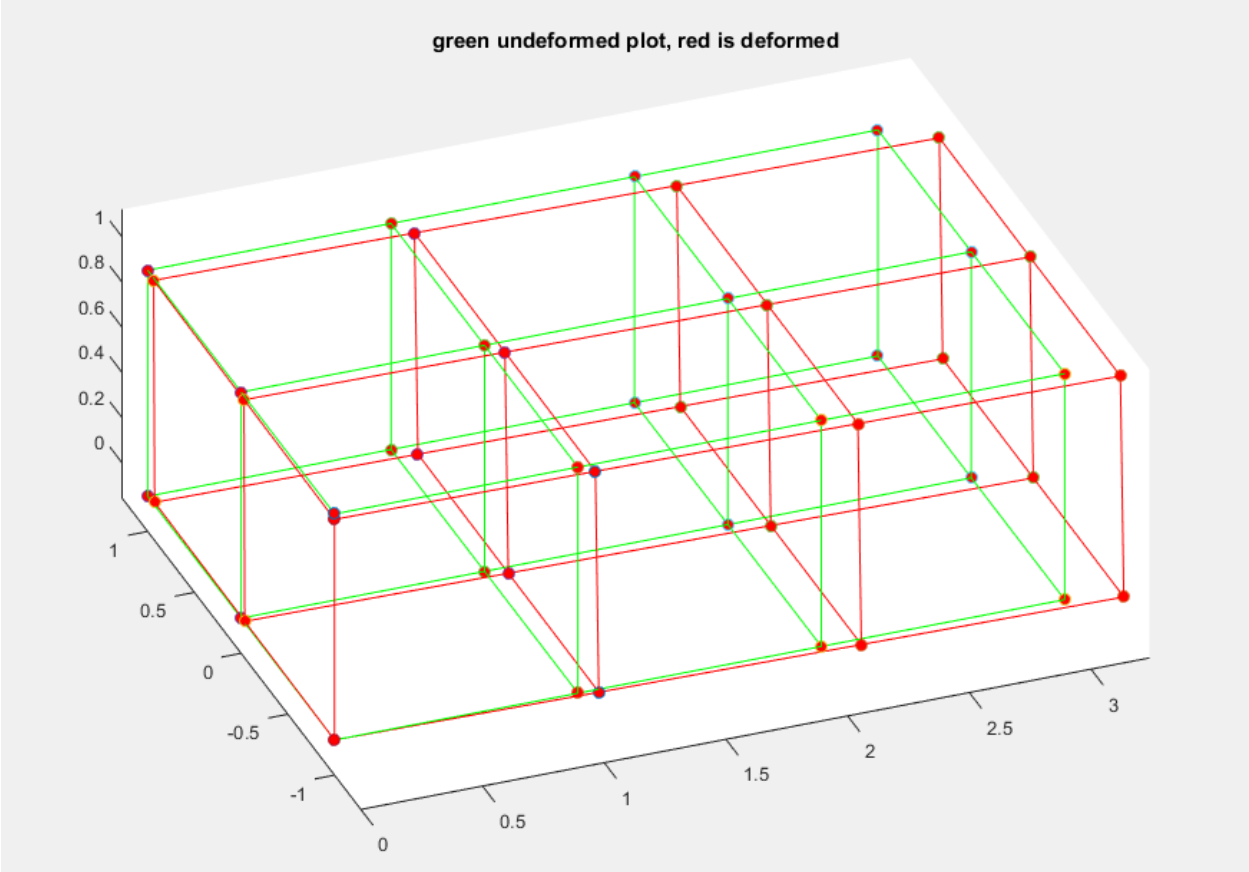


I will show the text format results at appendix

In the same way we can prove the 0,0,90 results validate the code.

For 45,45,45

The deformed vs undeformed plot is given below.



Details results are given in the appendix.

Results for the Matlab code using E=200GPA and nu=0.3

We know E=2*G(1+nu) where G is the share modolus.

So G= $6*10^{10}$Pa

Now the modified input file will be

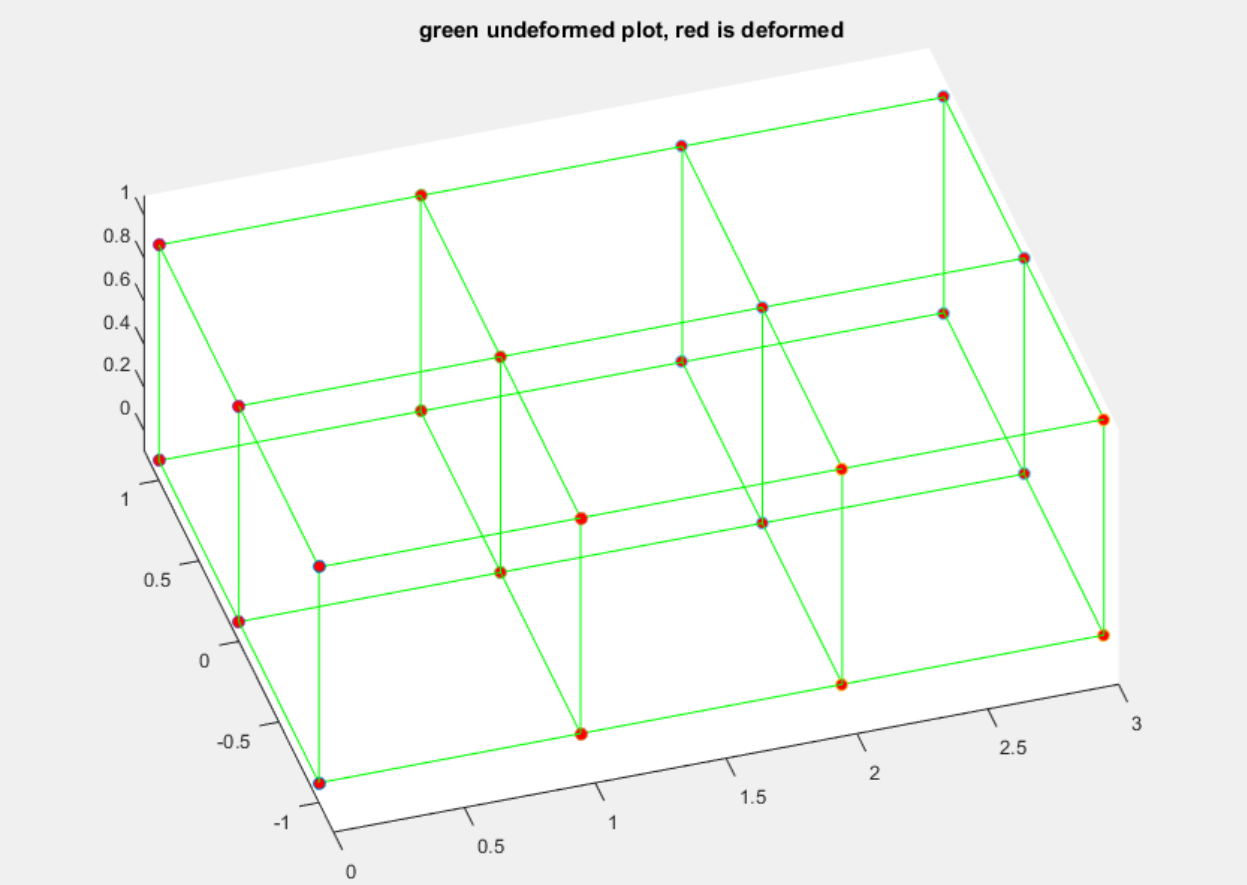
File Edit View

```
No._material_props:    2
    Shear_modulus:   60000000000.
    Poissons_ratio:  0.3
No._coords_per_node:   3
No._DOF_per_node:      3
No._nodes:             24
Nodal_coords:
    0.0   -1.0    0.0
    1.0   -1.0    0.0
    2.0   -1.0    0.0
    3.0   -1.0    0.0
    0.0    0.0    0.0
    1.0    0.0    0.0
    2.0    0.0    0.0
    3.0    0.0    0.0
    0.0    1.0    0.0
    1.0    1.0    0.0
    2.0    1.0    0.0
    3.0    1.0    0.0
    0.0   -1.0    1.0
    1.0   -1.0    1.0
    2.0   -1.0    1.0
    3.0   -1.0    1.0
    0.0    0.0    1.0
    1.0    0.0    1.0
    2.0    0.0    1.0
    3.0    0.0    1.0
    0.0    1.0    1.0
    1.0    1.0    1.0
    2.0    1.0    1.0
    3.0    1.0    1.0
No._elements:                       6
Max_no._nodes_on_any_one_element:   8
element_identifier; no. nodes on element; connectivity:
```

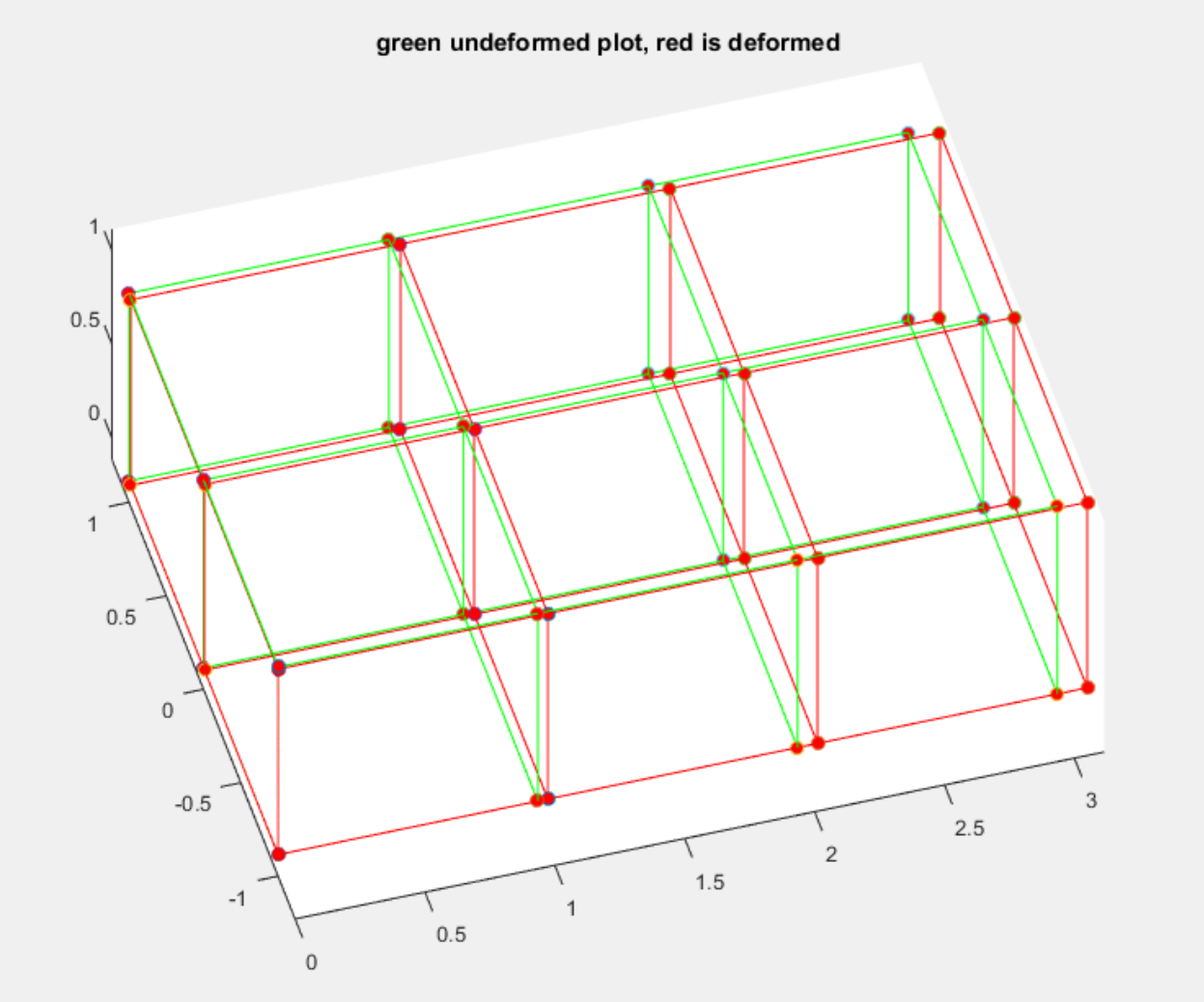
Ln 12, Col 1 | 1 of 1,309 characters

Now I will ran this input file from the main file of the solidmechanics.org website mentioned code.

Undeformed Plot



Deformed and undeformed plot in the same plot



Discussion:

In this project:

1. A big picture of the code was first give me a basic understating of the different parts of the code.
2. After understating the project requirements I figure out which function I need to modify.
3. I tried to learn the cubic symmetry of bcc or fcc material so that I can generate my own element stiffness matrix
4. HW 4 four code gave me a startup to build that code. But there are some dimension problems.
5. I learned how to handle 4D matrix in MATLAB since it is a generalized code it can handle different things
6. I validate the tensorial equation for cubic symmetric material

7. I learnt how to rotate the material stiffness tensor usin rotation angle using Euler Bunge angle.
8. I validate my own code using some known euler angle significance. The validation process give satisfactory results
9. I learned how the FEA knowledge is applied throughout the code
10. Finally I ran the main code and visualize the plot and get the text results.

This code give me an big picture how to modify a part of a big code and use it in a specified situation.

Code for cubid symmetry:

```
function FEM_2Dor3D_linelast_cubic_symmetry
%
%          Example 2D and 3D Linear elastic FEM code
%            Currently coded to run either plane strain or plane stress
(2DOF) or general 3D but
%            could easily be modified for axisymmetry too.
%
%        Variables read from input file;
%        nprops               No. material parameters
%        materialprops(i)     List of material parameters
%        ncoord               No. spatial coords (2 for 2D, 3 for 3D)
%        ndof                 No. degrees of freedom per node (2 for 2D, 3 for
3D)
%                             (here ndof=ncoord, but the program allows them
to be different
%                             to allow extension to plate & beam elements with
C^1 continuity)
%        nnode                No. nodes
%        coords(i,j)          ith coord of jth node, for i=1..ncoord;
j=1..nnode
%        nelem                No. elements
%        maxnodes             Max no. nodes on any one element (used for array
dimensioning)
%        nelnodes(i)          No. nodes on the ith element
%        elident(i)           An integer identifier for the ith element.  Not
used
%                             in this code but could be used to switch on
reduced integration,
%                             etc.
%        connect(i,j)         List of nodes on the jth element
%        nfix                 Total no. prescribed displacements
%        fixnodes(i,j)        List of prescribed displacements at nodes
%                             fixnodes(1,j) Node number
%                             fixnodes(2,j) Displacement component number (1,
2 or 3)
%                             fixnodes(3,j) Value of the displacement
%        ndload               Total no. element faces subjected to tractions
%        dloads(i,j)          List of element tractions
%                             dloads(1,j) Element number
%                             dloads(2,j) face number
%                             dloads(3,j), dloads(4,j), dloads(5,j) Components
of traction
%                             (assumed uniform)
%
%  To run the program you first need to set up an input file, as described in
%  the lecture notes.  Then change the fopen command below to point to the
file.
%  Also change the fopen command in the post-processing step (near the bottom
of the
%  program) to point to a suitable output file.  Then execute the file in
%  the usual fashion (e.g. hit the green arrow at the top of the MATLAB
%  editor window)
%
%
```

```
% ==================== Read data from the input file
==========================
%
%
% YOU NEED TO CHANGE THE PATH & FILE NAME TO POINT TO YOUR INPUT FILE
%
%infile=fopen('linear_elastic_Brick20.txt','r');
%outfile=fopen('FEM_results.txt','w');

infile=fopen('linear_elastic_brick8.txt','r');
outfile=fopen('linear_elastic_brick8_results_all45.txt','w');

[nprops,materialprops,ncoord,ndof,nnode,coords,nelem,maxnodes,connect,nelnode
s,elident,nfix,fixnodes,ndload,dloads] = read_input_file(infile);

fclose(infile);

% Plot the initial mesh as a check
close all
figure
plotmesh(coords,ncoord,nnode,connect,nelem,elident,nelnodes,'g');

%
%============================ MAIN FEM ANALYSIS PROCEDURE
=======================
%
%   dofs        Nodal displacements.  Let u_i^a be ith displacement component
%               at jth node.  Then dofs contain (u_1^1, u_2^1, u_1^2,
u_2^2....) for 2D
%               and (u_1^1, u_2^1, u_3^1, u_1^2, u_2^2, u_3^2....) for 3D
%
%   K           Global stiffness matrix.  Stored as (K_1111 K_1112  K_1121
K_1122...
%                                                    K_1211 K_1212  K_1221
K_1222...
%                                                    K_2111 K_2112  K_2121
K_2122...)
%               for 2D problem and similarly for 3D problem
%   r           Force vector.  Currently only includes contribution from
tractions
%               acting on element faces (i.e. body forces are neglected)
%
  dofs = zeros(ndof*nnode,1);

  K =
globalstiffness(ncoord,ndof,nnode,coords,nelem,maxnodes,elident,nelnodes,conn
ect,materialprops,dofs);

  r =
globaltraction(ncoord,ndof,nnode,ndload,coords,nelnodes,elident,connect,dload
s,dofs);

%
%  Fix constrained nodes.  We should really do this in a way that preserves
the symmetry of K
```

```
%  but it's not worth it for the small demo problems here.
%
  for n = 1:nfix
     rw = ndof*(fixnodes(1,n)-1) + fixnodes(2,n);
     for cl = 1:ndof*nnode
        K(rw,cl) = 0;
     end
     K(rw,rw) = 1.;
     r(rw) = fixnodes(3,n);
  end
%
% Solve for the displacements
%

  dofs = K\r;

%
%================================= POST-PROCESSING
=================================
%
% Create a plot of the deformed mesh
%

  defcoords = zeros(ndof,nnode);
  scalefactor = 10000000000.0; % if the displacement is small change it
  for i = 1:nnode
    for j = 1:ndof
       defcoords(j,i) = coords(j,i) + scalefactor*dofs(ndof*(i-1)+j);
    end
  end

 figure
 plotmesh(coords,ncoord,nnode,connect,nelem,elident,nelnodes,'g');
 hold on
 plotmesh(defcoords,ncoord,nnode,connect,nelem,elident,nelnodes,'r');

 print_results(outfile, ...
    nprops,materialprops,ncoord,ndof,nnode,coords, ...
    nelem,maxnodes,connect,nelnodes,elident, ...
    nfix,fixnodes,ndload,dloads,dofs)

fclose(outfile);
end

%================ Material Stiffness ================================
%
%    Computes elasticity tensor C_{ijkl} = shear modulus and Poissons ratio
%    Currently coded either for plane strain, plane stress or general 3D.
%
function C = materialstiffness(ndof,ncoord,strain,materialprops)

   mu = materialprops(1);
   nu = materialprops(2);
  %Three Co-efficient of cubic symmetric stiffness matrix matrix of Cu
```

```
  %if you put this in gigapascle value change the scale factor at
postprocessing function to make the displacement visual
    C11 = 225e9;
    C12 = 153e9;
    C44 = 115e9;

    C = zeros(ndof,ncoord,ndof,ncoord); %stifness in new co-ordiante
initialization
    C_ref = zeros(ndof,ncoord,ndof,ncoord); %stifness in reference co-
ordiante initialization

    %which set you want to ran just active it or vice versa
    %phi_1 =0; phi = 0; phi_2 =0;
    %phi_1 =90; phi = 0; phi_2 =-90;
    %phi_1 =0; phi = 0; phi_2 =90;
    phi_1 =45; phi = 45; phi_2 =45;

    %converting  degree to radian
    %matlab don't understand degree mode and neither transform it radian
    %automaticly
    t1=phi_1*pi/180;
    t2=phi*pi/180;
    t3=phi_2*pi/180;

    %converting trichonomitric angle to suitable formet
    %assuming c1,c2,c3, s1,s2,s3 respectively for smooth coding

    c1=cos(t1);
    c2=cos(t2);
    c3=cos(t3);

    s1=sin(t1);
    s2=sin(t2);
    s3=sin(t3);

    %Rotation Matrix(3*3)
    R =[ c1*c3-s1*s3*c2    s1*c3+c1*s3*c2    s3*s2;
       -c1*s3-s1*c3*c2   -s1*s3+c1*c3*c2    c3*s2;
             s1*s2            -c1*s2           c2  ]
%%creating the material stifness matrix (3*3*3*3) in reference configuration
% Initialize the 4D tensor

   if (ncoord == 2)

%  planestrain = 0 => plane stress, planestrain = 1 => plane strain
   planestrain = materialprops(3);
     for i = 1:2
       for j = 1:2
         for k = 1:2
           for l = 1:2
             if (planestrain==1)
               if (i==j && k==l)
                   C(i,j,k,l) = C(i,j,k,l)+2*mu*nu/(1-2*nu);
               end
             else
```

```
                if (i==j && k==l)
                    C(i,j,k,l) = C(i,j,k,l)+2*mu*nu/(1-nu);
                end
            end
            if (i==l && k==j)
                C(i,j,k,l) = C(i,j,k,l)+mu;
            end
            if (i==k && j==l)
                C(i,j,k,l) = C(i,j,k,l)+mu;
            end
          end
        end
      end
    end

   elseif (ncoord == 3)
 %change in code starts here
    % Populate the tensor according to cubic symmetry rules
    for i = 1:3
        for j = 1:3
            for k = 1:3
                for l = 1:3
                    if i == k && j == l && i == j %c1111
                        C_ref(i, j, k, l) =C_ref(i,j,k,l)+ C11;  % Normal
terms along axes
                    elseif i == k && j == l
                        C_ref(i, j, k, l) =C_ref(i,j,k,l)+ C44;  % Shear
terms on plane normals
                    elseif i == j && k == l && i ~= k %this will eliminate
c1111 from this
                        C_ref(i, j, k, l) =C_ref(i,j,k,l) + C12;  % Coupling
between different axes c1122
                    end
                end
            end
        end
    end

 C_ref

%%creating transformation matrix T(3*3*3*3*3*3*3*3) total 8D
%initialize the new Transformation matrix which will be generated from
roation matrix
T = zeros(3, 3, 3, 3, 3, 3, 3, 3);
    for i = 1:3
        for j = 1:3
            for k = 1:3
                for l = 1:3
                    for m = 1:3
                        for n = 1:3
                            for o = 1:3
                                for p = 1:3
                                    T(i, j, k, l, m, n, o, p) =  R(i, m) *
R(j, n) * R(k, o) * R(l, p);
                                end
                            end
```

```
                        end
                    end
                end
            end
        end
    end
% Apply the transformation tensor T to rotate the stiffness tensor C_ref
            for i = 1:3
                for j = 1:3
                    for k = 1:3
                        for l = 1:3
                            sum = 0; % Initialize sum for the contraction
                            for m = 1:3
                                for n = 1:3
                                    for o = 1:3
                                        for p = 1:3
                                            sum = sum + T(i, j, k, l, m, n,
o, p) * C_ref(m, n, o, p);
                                        end
                                    end
                                end
                            end
                            C(i, j, k, l) = sum; % Assign the computed value
to the rotated tensor
                        end
                    end
                end
            end

        C
   end
end
%modification ends here
%================= Material Stress ==================================
%
%   Computes stress sigma_{ij} given strain epsilon_{ij}
%
function stress = materialstress(ndof,ncoord,strain,materialprops)

   C = materialstiffness(ndof,ncoord,strain,materialprops);
   stress = zeros(ndof,ncoord);
   for i = 1 : ndof
     for j = 1 : ncoord
        for k = 1 : ndof
          for l = 1: ncoord
            stress(i,j) = stress(i,j) + C(i,j,k,l)*strain(k,l);
          end
        end
     end
   end
  end
%
%====================== No. integration points ============================
%
%   Defines the number of integration points:be used for
%   each element type
```

```
%
function n = numberofintegrationpoints(ncoord,nelnodes,elident)

   if (ncoord == 1)
     n = nelnodes;
   elseif (ncoord == 2)
     if (nelnodes == 3)
         n = 1;
     end
     if (nelnodes == 6)
         n = 3;
     end
     if (nelnodes == 4)
         n = 4;
     end
     if (nelnodes == 8)
         n = 9;
     end
   elseif (ncoord == 3)
     if (nelnodes == 4)
         n = 1 ;
     end
     if (nelnodes == 10)
         n = 4;
     end
     if (nelnodes == 8)
         n = 8;
     end
     if (nelnodes == 20)
         n = 27;
     end
   end
end
%
%====================== INTEGRATION POINTS ==================================
%
%   Defines positions of integration points
%
function xi = integrationpoints(ncoord,nelnodes,npoints,elident)

   xi = zeros(ncoord,npoints);
%
%  1D elements
%
   if (ncoord == 1)
     if (npoints==1)
       xi(1,1) = 0.;
     elseif (npoints == 2)
       xi(1,1) = -0.5773502692;
       xi(1,2) = -xi(1,1);
     elseif (npoints == 3)
       xi(1,1) = -0.7745966692;
       xi(1,2) = 0.0;
       xi(1,3) = -xi(1,1);
     end
%
```

```
%  2D elements
%
   elseif (ncoord == 2)
%
%    Triangular element
%
     if ( nelnodes == 3 || nelnodes == 6 )
       if (npoints == 1)
         xi(1,1) = 1./3.;
         xi(2,1) = 1./3.;
       elseif (npoints == 3)
         xi(1,1) = 0.6;
         xi(2,1) = 0.2;
         xi(1,2) = 0.2;
         xi(2,2) = 0.6;
         xi(1,3) = 0.2;
         xi(2,3) = 0.2;
       elseif (npoints == 4)
         xi(1,1) = 1./3.;
         xi(2,1) = 1./3.;
         xi(1,2) = 0.6;
         xi(2,2) = 0.2;
         xi(1,3) = 0.2;
         xi(2,3) = 0.6;
         xi(1,4) = 0.2;
         xi(2,4) = 0.2;
       end
%
%    Rectangular element
%
     elseif ( nelnodes==4 || nelnodes==8 )

       if (npoints == 1)
         xi(1,1) = 0.;
         xi(2,1) = 0.;
       elseif (npoints == 4)
         xi(1,1) = -0.5773502692;
         xi(2,1) = xi(1,1);
         xi(1,2) = -xi(1,1);
         xi(2,2) = xi(1,1);
         xi(1,3) = xi(1,1);
         xi(2,3) = -xi(1,1);
         xi(1,4) = -xi(1,1);
         xi(2,4) = -xi(1,1);
       elseif (npoints == 9)
         xi(1,1) = -0.7745966692;
         xi(2,1) = xi(1,1);
         xi(1,2) = 0.0;
         xi(2,2) = xi(1,1);
         xi(1,3) = -xi(1,1);
         xi(2,3) = xi(1,1);
         xi(1,4) = xi(1,1);
         xi(2,4) = 0.0;
         xi(1,5) = 0.0;
         xi(2,5) = 0.0;
         xi(1,6) = -xi(1,1);
         xi(2,6) = 0.0;
```

```
        xi(1,7) = xi(1,1);
        xi(2,7) = -xi(1,1);
        xi(1,8) = 0.;
        xi(2,8) = -xi(1,1);
        xi(1,9) = -xi(1,1);
        xi(2,9) = -xi(1,1);
      end
    end
%
%   3D elements
%
  elseif (ncoord == 3)
%
%  3D elements
%
    if (nelnodes == 4 || nelnodes==10 )
      if (npoints == 1)
        xi(1,1) = 0.25;
        xi(2,1) = 0.25;
        xi(3,1) = 0.25;
      elseif (npoints == 4)
        xi(1,1) = 0.58541020;
        xi(2,1) = 0.13819660;
        xi(3,1) = xi(2,1);
        xi(1,2) = xi(2,1);
        xi(2,2) = xi(1,1);
        xi(3,2) = xi(2,1);
        xi(1,3) = xi(2,1);
        xi(2,3) = xi(2,1);
        xi(3,3) = xi(1,1);
        xi(1,4) = xi(2,1);
        xi(2,4) = xi(2,1);
        xi(3,4) = xi(2,1);
      end
    elseif ( nelnodes==8 || nelnodes==20 )
      if (npoints == 1)
        xi(1,1) = 0.;
        xi(2,1) = 0.;
        xi(3,1) = 0.;
      elseif (npoints == 8)
        x1D = [-0.5773502692,0.5773502692];
        for k = 1:2
          for j = 1:2
            for i = 1:2
              n = 4*(k-1) + 2*(j-1) + i;
              xi(1,n) = x1D(i);
              xi(2,n) = x1D(j);
              xi(3,n) = x1D(k);
            end
          end
        end
      elseif (npoints == 27)
        x1D = [-0.7745966692,0.,0.7745966692];
        for k = 1:3
          for j = 1:3
            for i = 1:3
              n = 9*(k-1) + 3*(j-1) + i;
```

```
                xi(1,n) = x1D(i);
                xi(2,n) = x1D(j);
                xi(3,n) = x1D(k);
              end
            end
          end
        end
      end
    end
end

%
%================ INTEGRATION WEIGHTS ================================
%
%   Defines integration weights w_i
%
function w = integrationweights(ncoord,nelnodes,npoints,elident)

   w = zeros(npoints,1);

%
%  1D elements
%
   if (ncoord == 1)
     if (npoints == 1)
       w(1) = 2.;
     elseif (npoints == 2)
       w = [1.,1.];
     elseif (npoints == 3)
       w = [0.555555555,0.888888888,0.555555555];
     end
%
%  2D elements
%
   elseif (ncoord == 2)
%
%    Triangular element
%
     if ( nelnodes == 3 || nelnodes == 6 )
       if (npoints == 1)
         w(1) = 0.5;
       elseif (npoints == 3)
         w(1) = 1./6.;
         w(2) = 1./6.;
         w(3) = 1./6.;
       elseif (npoints == 4)
         w = [-27./96.,25./96.,25/96.,25/96.];
       end
%
%    Rectangular element
%
     elseif ( nelnodes==4 || nelnodes==8 )

       if (npoints == 1)
         w(1) = 4.;
       elseif (npoints == 4)
```

```
        w = [1.,1.,1.,1.];
      elseif (npoints == 9 )
        w1D = [0.555555555,0.888888888,0.55555555555];
        for j = 1:3
          for i = 1:3
            n = 3*(j-1)+i;
            w(n) = w1D(i)*w1D(j);
          end
        end
      end
    end

  elseif (ncoord == 3)
%
%  3D elements
%
    if (nelnodes == 4 || nelnodes==10 )
      if (npoints == 1)
        w(1) = 1./6.;
      elseif (npoints == 4)
        w = [1./24.,1./24.,1./24.,1./24.];
      end
    elseif ( nelnodes==8 || nelnodes==20 )
      if (npoints == 1)
        w(1) = 8.;
      elseif (npoints == 8)
        w = [1.,1.,1.,1.,1.,1.,1.,1.];
      elseif (npoints == 27)
        w1D = [0.555555555,0.888888888,0.55555555555];
        for k = 1:3
          for j = 1:3
            for i = 1:3
              n = 9*(k-1)+3*(j-1)+i;
              w(n) = w1D(i)*w1D(j)*w1D(k);
            end
          end
        end
      end
    end
  end
end

%
%================ SHAPE FUNCTIONS ================================
%
%        Calculates shape functions for various element types
%
function N = shapefunctions(nelnodes,ncoord,elident,xi)


  N = zeros(nelnodes,1);
%
%  1D elements
%
 if (ncoord == 1)
   if (nelnodes==2)
```

```
      N(1) = 0.5*(1.+xi(1));
      N(2) = 0.5*(1.-xi(1));
    elseif (nelnodes == 3)
      N(1) = -0.5*xi(1)*(1.-xi(1));
      N(2) =  0.5*xi(1)*(1.+xi(1));
      N(3) = (1.-xi(1))*(1.+xi(1));
    end
%
%  2D elements
%
  elseif (ncoord == 2)
%
%    Triangular element
%
    if ( nelnodes == 3 )
      N(1) = xi(1);
      N(2) = xi(2);
      N(3) = 1.-xi(1)-xi(2);
    elseif ( nelnodes == 6 )
      xi3 = 1.-xi(1)-xi(2);
      N(1) = (2.*xi(1)-1.)*xi(1);
      N(2) = (2.*xi(2)-1.)*xi(2);
      N(3) = (2.*xi3-1.)*xi3;
      N(4) = 4.*xi(1)*xi(2);
      N(5) = 4.*xi(2)*xi3;
      N(6) = 4.*xi3*xi(1);
%
%    Rectangular element
%
    elseif ( nelnodes == 4 )
      N(1) = 0.25*(1.-xi(1))*(1.-xi(2));
      N(2) = 0.25*(1.+xi(1))*(1.-xi(2));
      N(3) = 0.25*(1.+xi(1))*(1.+xi(2));
      N(4) = 0.25*(1.-xi(1))*(1.+xi(2));
    elseif (nelnodes == 8)
      N(1) = -0.25*(1.-xi(1))*(1.-xi(2))*(1.+xi(1)+xi(2));
      N(2) = 0.25*(1.+xi(1))*(1.-xi(2))*(xi(1)-xi(2)-1.);
      N(3) = 0.25*(1.+xi(1))*(1.+xi(2))*(xi(1)+xi(2)-1.);
      N(4) = 0.25*(1.-xi(1))*(1.+xi(2))*(xi(2)-xi(1)-1.);
      N(5) = 0.5*(1.-xi(1)*xi(1))*(1.-xi(2));
      N(6) = 0.5*(1.+xi(1))*(1.-xi(2)*xi(2));
      N(7) = 0.5*(1.-xi(1)*xi(1))*(1.+xi(2));
      N(8) = 0.5*(1.-xi(1))*(1.-xi(2)*xi(2));
    end
%
  elseif (ncoord==3)

    if (nelnodes == 4)
      N(1) = xi(1);
      N(2) = xi(2);
      N(3) = xi(3);
      N(4) = 1.-xi(1)-xi(2)-xi(3);
    elseif (nelnodes == 10)
      xi4 = 1.-xi(1)-xi(2)-xi(3);
      N(1) = (2.*xi(1)-1.)*xi(1);
      N(2) = (2.*xi(2)-1.)*xi(2);
      N(3) = (2.*xi(3)-1.)*xi(3);
```

```
      N(4) = (2.*xi4-1.)*xi4;
      N(5) = 4.*xi(1)*xi(2);
      N(6) = 4.*xi(2)*xi(3);
      N(7) = 4.*xi(3)*xi(1);
      N(8) = 4.*xi(1)*xi4;
      N(9) = 4.*xi(2)*xi4;
      N(10) = 4.*xi(3)*xi4;
    elseif (nelnodes == 8)
      N(1) = (1.-xi(1))*(1.-xi(2))*(1.-xi(3))/8.;
      N(2) = (1.+xi(1))*(1.-xi(2))*(1.-xi(3))/8.;
      N(3) = (1.+xi(1))*(1.+xi(2))*(1.-xi(3))/8.;
      N(4) = (1.-xi(1))*(1.+xi(2))*(1.-xi(3))/8.;
      N(5) = (1.-xi(1))*(1.-xi(2))*(1.+xi(3))/8.;
      N(6) = (1.+xi(1))*(1.-xi(2))*(1.+xi(3))/8.;
      N(7) = (1.+xi(1))*(1.+xi(2))*(1.+xi(3))/8.;
      N(8) = (1.-xi(1))*(1.+xi(2))*(1.+xi(3))/8.;
    elseif (nelnodes == 20)
      N(1) = (1.-xi(1))*(1.-xi(2))*(1.-xi(3))*(-xi(1)-xi(2)-xi(3)-2.)/8.;
      N(2) = (1.+xi(1))*(1.-xi(2))*(1.-xi(3))*(xi(1)-xi(2)-xi(3)-2.)/8.;
      N(3) = (1.+xi(1))*(1.+xi(2))*(1.-xi(3))*(xi(1)+xi(2)-xi(3)-2.)/8.;
      N(4) = (1.-xi(1))*(1.+xi(2))*(1.-xi(3))*(-xi(1)+xi(2)-xi(3)-2.)/8.;
      N(5) = (1.-xi(1))*(1.-xi(2))*(1.+xi(3))*(-xi(1)-xi(2)+xi(3)-2.)/8.;
      N(6) = (1.+xi(1))*(1.-xi(2))*(1.+xi(3))*(xi(1)-xi(2)+xi(3)-2.)/8.;
      N(7) = (1.+xi(1))*(1.+xi(2))*(1.+xi(3))*(xi(1)+xi(2)+xi(3)-2.)/8.;
      N(8) = (1.-xi(1))*(1.+xi(2))*(1.+xi(3))*(-xi(1)+xi(2)+xi(3)-2.)/8.;
      N(9)  = (1.-xi(1)^2)*(1.-xi(2))*(1.-xi(3))/4.;
      N(10) = (1.+xi(1))*(1.-xi(2)^2)*(1.-xi(3))/4.;
      N(11) = (1.-xi(1)^2)*(1.+xi(2))*(1.-xi(3))/4.;
      N(12) = (1.-xi(1))*(1.-xi(2)^2)*(1.-xi(3))/4.;
      N(13) = (1.-xi(1)^2)*(1.-xi(2))*(1.+xi(3))/4.;
      N(14) = (1.+xi(1))*(1.-xi(2)^2)*(1.+xi(3))/4.;
      N(15) = (1.-xi(1)^2)*(1.+xi(2))*(1.+xi(3))/4.;
      N(16) = (1.-xi(1))*(1.-xi(2)^2)*(1.+xi(3))/4.;
      N(17) = (1.-xi(1))*(1.-xi(2))*(1.-xi(3)^2)/4.;
      N(18) = (1.+xi(1))*(1.-xi(2))*(1.-xi(3)^2)/4.;
      N(19) = (1.+xi(1))*(1.+xi(2))*(1.-xi(3)^2)/4.;
      N(20) = (1.-xi(1))*(1.+xi(2))*(1.-xi(3)^2)/4.;
    end
  end

end

%
%================= SHAPE FUNCTION DERIVATIVES ======================
%
function dNdxi = shapefunctionderivs(nelnodes,ncoord,elident,xi)

  dNdxi = zeros(nelnodes,ncoord);
%
% 1D elements
%
  if (ncoord == 1)
    if (nelnodes==2)
      dNdxi(1,1) = 0.5;
      dNdxi(2,1) = -0.5;
    elseif (nelnodes == 3)
```

```
      dNdxi(1,1) = -0.5+xi(1);
      dNdxi(2,1) =  0.5+xi(1);
      dNdxi(3,1) = -2.*xi(1);
    end
%
%  2D elements
%
  elseif (ncoord == 2)
%
%    Triangular element
%
    if ( nelnodes == 3 )
      dNdxi(1,1) = 1.;
      dNdxi(2,2) = 1.;
      dNdxi(3,1) = -1.;
      dNdxi(3,2) = -1.;
    elseif ( nelnodes == 6 )
      xi3 = 1.-xi(1)-xi(2);
      dNdxi(1,1) = 4.*xi(1)-1.;
      dNdxi(2,2) = 4.*xi(2)-1.;
      dNdxi(3,1) = -(4.*xi3-1.);
      dNdxi(3,2) = -(4.*xi3-1.);
      dNdxi(4,1) = 4.*xi(2);
      dNdxi(4,2) = 4.*xi(1);
      dNdxi(5,1) = -4.*xi(2);
      dNdxi(5,2) = -4.*xi(1);
      dNdxi(6,1) = 4.*xi3 - 4.*xi(1);
      dNdxi(6,2) = 4.*xi3 - 4.*xi(2);
%
%    Rectangular element
%
    elseif ( nelnodes == 4 )
      dNdxi(1,1) = -0.25*(1.-xi(2));
      dNdxi(1,2) = -0.25*(1.-xi(1));
      dNdxi(2,1) = 0.25*(1.-xi(2));
      dNdxi(2,2) = -0.25*(1.+xi(1));
      dNdxi(3,1) = 0.25*(1.+xi(2));
      dNdxi(3,2) = 0.25*(1.+xi(1));
      dNdxi(4,1) = -0.25*(1.+xi(2));
      dNdxi(4,2) = 0.25*(1.-xi(1));
    elseif (nelnodes == 8)
      dNdxi(1,1) = 0.25*(1.-xi(2))*(2.*xi(1)+xi(2));
      dNdxi(1,2) = 0.25*(1.-xi(1))*(xi(1)+2.*xi(2));
      dNdxi(2,1) = 0.25*(1.-xi(2))*(2.*xi(1)-xi(2));
      dNdxi(2,2) = 0.25*(1.+xi(1))*(2.*xi(2)-xi(1));
      dNdxi(3,1) = 0.25*(1.+xi(2))*(2.*xi(1)+xi(2));
      dNdxi(3,2) = 0.25*(1.+xi(1))*(2.*xi(2)+xi(1));
      dNdxi(4,1) = 0.25*(1.+xi(2))*(2.*xi(1)-xi(2));
      dNdxi(4,2) = 0.25*(1.-xi(1))*(2.*xi(2)-xi(1));
      dNdxi(5,1) = -xi(1)*(1.-xi(2));
      dNdxi(5,2) = -0.5*(1.-xi(1)*xi(1));
      dNdxi(6,1) = 0.5*(1.-xi(2)*xi(2));
      dNdxi(6,2) = -(1.+xi(1))*xi(2);
      dNdxi(7,1) = -xi(1)*(1.+xi(2));
      dNdxi(7,2) = 0.5*(1.-xi(1)*xi(1));
      dNdxi(8,1) = -0.5*(1.-xi(2)*xi(2));
      dNdxi(8,2) = -(1.-xi(1))*xi(2);
```

```
    end
%
%   3D elements
%
  elseif (ncoord==3)

    if (nelnodes == 4)
      dNdxi(1,1) = 1.;
      dNdxi(2,2) = 1.;
      dNdxi(3,3) = 1.;
      dNdxi(4,1) = -1.;
      dNdxi(4,2) = -1.;
      dNdxi(4,3) = -1.;
    elseif (nelnodes == 10)
      xi4 = 1.-xi(1)-xi(2)-xi(3);
      dNdxi(1,1) = (4.*xi(1)-1.);
      dNdxi(2,2) = (4.*xi(2)-1.);
      dNdxi(3,3) = (4.*xi(3)-1.);
      dNdxi(4,1) = -(4.*xi4-1.);
      dNdxi(4,2) = -(4.*xi4-1.);
      dNdxi(4,3) = -(4.*xi4-1.);
      dNdxi(5,1) = 4.*xi(2);
      dNdxi(5,2) = 4.*xi(1);
      dNdxi(6,2) = 4.*xi(3);
      dNdxi(6,3) = 4.*xi(2);
      dNdxi(7,1) = 4.*xi(3);
      dNdxi(7,3) = 4.*xi(1);
      dNdxi(8,1) = 4.*(xi4-xi(1));
      dNdxi(8,2) = -4.*xi(1);
      dNdxi(8,3) = -4.*xi(1);
      dNdxi(9,1) = -4.*xi(2);
      dNdxi(9,2) = 4.*(xi4-xi(2));
      dNdxi(9,3) = -4.*xi(2);
      dNdxi(10,1) = -4.*xi(3)*xi4;
      dNdxi(10,2) = -4.*xi(3);
      dNdxi(10,3) = 4.*(xi4-xi(3));
    elseif (nelnodes == 8)
      dNdxi(1,1) = -(1.-xi(2))*(1.-xi(3))/8.;
      dNdxi(1,2) = -(1.-xi(1))*(1.-xi(3))/8.;
      dNdxi(1,3) = -(1.-xi(1))*(1.-xi(2))/8.;
      dNdxi(2,1) = (1.-xi(2))*(1.-xi(3))/8.;
      dNdxi(2,2) = -(1.+xi(1))*(1.-xi(3))/8.;
      dNdxi(2,3) = -(1.+xi(1))*(1.-xi(2))/8.;
      dNdxi(3,1) = (1.+xi(2))*(1.-xi(3))/8.;
      dNdxi(3,2) = (1.+xi(1))*(1.-xi(3))/8.;
      dNdxi(3,3) = -(1.+xi(1))*(1.+xi(2))/8.;
      dNdxi(4,1) = -(1.+xi(2))*(1.-xi(3))/8.;
      dNdxi(4,2) = (1.-xi(1))*(1.-xi(3))/8.;
      dNdxi(4,3) = -(1.-xi(1))*(1.+xi(2))/8.;
      dNdxi(5,1) = -(1.-xi(2))*(1.+xi(3))/8.;
      dNdxi(5,2) = -(1.-xi(1))*(1.+xi(3))/8.;
      dNdxi(5,3) = (1.-xi(1))*(1.-xi(2))/8.;
      dNdxi(6,1) = (1.-xi(2))*(1.+xi(3))/8.;
      dNdxi(6,2) = -(1.+xi(1))*(1.+xi(3))/8.;
      dNdxi(6,3) = (1.+xi(1))*(1.-xi(2))/8.;
      dNdxi(7,1) = (1.+xi(2))*(1.+xi(3))/8.;
      dNdxi(7,2) = (1.+xi(1))*(1.+xi(3))/8.;
```

```
        dNdxi(7,3) = (1.+xi(1))*(1.+xi(2))/8.;
        dNdxi(8,1) = -(1.+xi(2))*(1.+xi(3))/8.;
        dNdxi(8,2) = (1.-xi(1))*(1.+xi(3))/8.;
        dNdxi(8,3) = (1.-xi(1))*(1.+xi(2))/8.;
      elseif (nelnodes == 20)
        dNdxi(1,1) = (-(1.-xi(2))*(1.-xi(3))*(-xi(1)-xi(2)-xi(3)-2.)-(1.-
xi(1))*(1.-xi(2))*(1.-xi(3)))/8.;
        dNdxi(1,2) = (-(1.-xi(1))*(1.-xi(3))*(-xi(1)-xi(2)-xi(3)-2.)-(1.-
xi(1))*(1.-xi(2))*(1.-xi(3)))/8.;
        dNdxi(1,3) = (-(1.-xi(1))*(1.-xi(2))*(-xi(1)-xi(2)-xi(3)-2.)-(1.-
xi(1))*(1.-xi(2))*(1.-xi(3)))/8.;

        dNdxi(2,1) = ((1.-xi(2))*(1.-xi(3))*(xi(1)-xi(2)-xi(3)-
2.)+(1.+xi(1))*(1.-xi(2))*(1.-xi(3)))/8.;
        dNdxi(2,2) = (-(1.+xi(1))*(1.-xi(3))*(xi(1)-xi(2)-xi(3)-2.)-
(1.+xi(1))*(1.-xi(2))*(1.-xi(3)))/8.;
        dNdxi(2,3) = (-(1.+xi(1))*(1.-xi(2))*(xi(1)-xi(2)-xi(3)-2.)-
(1.+xi(1))*(1.-xi(2))*(1.-xi(3)))/8.;

        dNdxi(3,1) = ((1.+xi(2))*(1.-xi(3))*(xi(1)+xi(2)-xi(3)-
2.)+(1.+xi(1))*(1.+xi(2))*(1.-xi(3)))/8.;
        dNdxi(3,2) = ((1.+xi(1))*(1.-xi(3))*(xi(1)+xi(2)-xi(3)-
2.)+(1.+xi(1))*(1.+xi(2))*(1.-xi(3)))/8.;
        dNdxi(3,3) = (-(1.+xi(1))*(1.+xi(2))*(xi(1)+xi(2)-xi(3)-2.)-
(1.+xi(1))*(1.+xi(2))*(1.-xi(3)))/8.;

        dNdxi(4,1) = (-(1.+xi(2))*(1.-xi(3))*(-xi(1)+xi(2)-xi(3)-2.)-(1.-
xi(1))*(1.+xi(2))*(1.-xi(3)))/8.;
        dNdxi(4,2) = ((1.-xi(1))*(1.-xi(3))*(-xi(1)+xi(2)-xi(3)-2.)+(1.-
xi(1))*(1.+xi(2))*(1.-xi(3)))/8.;
        dNdxi(4,3) = (-(1.-xi(1))*(1.+xi(2))*(-xi(1)+xi(2)-xi(3)-2.)-(1.-
xi(1))*(1.+xi(2))*(1.-xi(3)))/8.;
        dNdxi(5,1) = (-(1.-xi(2))*(1.+xi(3))*(-xi(1)-xi(2)+xi(3)-2.)-(1.-
xi(1))*(1.-xi(2))*(1.+xi(3)))/8.;
        dNdxi(5,2) = (-(1.-xi(1))*(1.+xi(3))*(-xi(1)-xi(2)+xi(3)-2.)-(1.-
xi(1))*(1.-xi(2))*(1.+xi(3)))/8.;
        dNdxi(5,3) = ((1.-xi(1))*(1.-xi(2))*(-xi(1)-xi(2)+xi(3)-2.)+(1.-
xi(1))*(1.-xi(2))*(1.+xi(3)))/8.;
        dNdxi(6,1) = ((1.-xi(2))*(1.+xi(3))*(xi(1)-xi(2)+xi(3)-
2.)+(1.+xi(1))*(1.-xi(2))*(1.+xi(3)))/8.;
        dNdxi(6,2) = (-(1.+xi(1))*(1.+xi(3))*(xi(1)-xi(2)+xi(3)-2.)-
(1.+xi(1))*(1.-xi(2))*(1.+xi(3)))/8.;
        dNdxi(6,3) = ((1.+xi(1))*(1.-xi(2))*(xi(1)-xi(2)+xi(3)-
2.)+(1.+xi(1))*(1.-xi(2))*(1.+xi(3)))/8.;
        dNdxi(7,1) = ((1.+xi(2))*(1.+xi(3))*(xi(1)+xi(2)+xi(3)-
2.)+(1.+xi(1))*(1.+xi(2))*(1.+xi(3)))/8.;
        dNdxi(7,2) = ((1.+xi(1))*(1.+xi(3))*(xi(1)+xi(2)+xi(3)-
2.)+(1.+xi(1))*(1.+xi(2))*(1.+xi(3)))/8.;
        dNdxi(7,3) = ((1.+xi(1))*(1.+xi(2))*(xi(1)+xi(2)+xi(3)-
2.)+(1.+xi(1))*(1.+xi(2))*(1.+xi(3)))/8.;
        dNdxi(8,1) = (-(1.+xi(2))*(1.+xi(3))*(-xi(1)+xi(2)+xi(3)-2.)-(1.-
xi(1))*(1.+xi(2))*(1.+xi(3)))/8.;
        dNdxi(8,2) = ((1.-xi(1))*(1.+xi(3))*(-xi(1)+xi(2)+xi(3)-2.)+(1.-
xi(1))*(1.+xi(2))*(1.+xi(3)))/8.;
        dNdxi(8,3) = ((1.-xi(1))*(1.+xi(2))*(-xi(1)+xi(2)+xi(3)-2.)+(1.-
xi(1))*(1.+xi(2))*(1.+xi(3)))/8.;
```

```
        dNdxi(9,1)  = -2.*xi(1)*(1.-xi(2))*(1.-xi(3))/4.;
        dNdxi(9,2)  = -(1.-xi(1)^2)*(1.-xi(3))/4.;
        dNdxi(9,3)  = -(1.-xi(1)^2)*(1.-xi(2))/4.;
        dNdxi(10,1)  = (1.-xi(2)^2)*(1.-xi(3))/4.;
        dNdxi(10,2)  = -2.*xi(2)*(1.+xi(1))*(1.-xi(3))/4.;
        dNdxi(10,3)  = -(1.-xi(2)^2)*(1.+xi(1))/4.;
        dNdxi(11,1)  = -2.*xi(1)*(1.+xi(2))*(1.-xi(3))/4.;
        dNdxi(11,2)  = (1.-xi(1)^2)*(1.-xi(3))/4.;
        dNdxi(11,3)  = -(1.-xi(1)^2)*(1.+xi(2))/4.;
        dNdxi(12,1)  = -(1.-xi(2)^2)*(1.-xi(3))/4.;
        dNdxi(12,2)  = -2.*xi(2)*(1.-xi(1))*(1.-xi(3))/4.;
        dNdxi(12,3)  = -(1.-xi(2)^2)*(1.-xi(1))/4.;
        dNdxi(13,1)  = -2.*xi(1)*(1.-xi(2))*(1.+xi(3))/4.;
        dNdxi(13,2)  = -(1.-xi(1)^2)*(1.+xi(3))/4.;
        dNdxi(13,3)  = (1.-xi(1)^2)*(1.-xi(2))/4.;
        dNdxi(14,1)  = (1.-xi(2)^2)*(1.+xi(3))/4.;
        dNdxi(14,2)  = -2.*xi(2)*(1.+xi(1))*(1.+xi(3))/4.;
        dNdxi(14,3)  = (1.-xi(2)^2)*(1.+xi(1))/4.;
        dNdxi(15,1)  = -2.*xi(1)*(1.+xi(2))*(1.+xi(3))/4.;
        dNdxi(15,2)  =  (1.-xi(1)^2)*(1.+xi(3))/4.;
        dNdxi(15,3)  = (1.-xi(1)^2)*(1.+xi(2))/4.;
        dNdxi(16,1)  = -(1.-xi(2)^2)*(1.+xi(3))/4.;
        dNdxi(16,2)  = -2.*xi(2)*(1.-xi(1))*(1.+xi(3))/4.;
        dNdxi(16,3)  = (1.-xi(2)^2)*(1.-xi(1))/4.;
        dNdxi(17,1) = -(1.-xi(2))*(1.-xi(3)^2)/4.;
        dNdxi(17,2) = -(1.-xi(1))*(1.-xi(3)^2)/4.;
        dNdxi(17,3) = -xi(3)*(1.-xi(1))*(1.-xi(2))/2.;
        dNdxi(18,1) = (1.-xi(2))*(1.-xi(3)^2)/4.;
        dNdxi(18,2) = -(1.+xi(1))*(1.-xi(3)^2)/4.;
        dNdxi(18,3) = -xi(3)*(1.+xi(1))*(1.-xi(2))/2.;
        dNdxi(19,1) = (1.+xi(2))*(1.-xi(3)^2)/4.;
        dNdxi(19,2) = (1.+xi(1))*(1.-xi(3)^2)/4.;
        dNdxi(19,3) = -xi(3)*(1.+xi(1))*(1.+xi(2))/2.;
        dNdxi(20,1) = -(1.+xi(2))*(1.-xi(3)^2)/4.;
        dNdxi(20,2) = (1.-xi(1))*(1.-xi(3)^2)/4.;
        dNdxi(20,3) = -xi(3)*(1.-xi(1))*(1.+xi(2))/2.;
      end
    end

  end
%
%================= ELEMENT STIFFNESS MATRIX ================================
%
function kel =
elstif(ncoord,ndof,nelnodes,elident,coord,materialprops,displacement)
%
%  Assemble the element stiffness
%
%    Arguments;
%
%      ncoord             No. coordinates (2 or 3 for 2D or 3D problem)
%      ndof               No. degrees of freedom per node (often ndof =
ncoord)
%      nelnodes           No. nodes on the element
%      elident            Element identifier (not used here - for future
enhancements!)
%      coords(i,a)        ith coord of ath node
```

```
%      materialprops      Material properties passed on:constitutive
procedures
%      displacement(i,a)  ith displacement component at ath node
%
%   Local variables
%      npoints            No. integration points
%      xi(i,inpt)         ith local coord of integration point no. intpt
%      w(intpt)           weight for integration point no. intpt
%      N(a)               Shape function associated with ath node on element
%      dNdxi(a,i)         Derivative of ath shape function wrt ith local
coord
%      dNdx(a,i)          Derivative of ath shape function wrt ith global
coord
%      dxdxi(i,j)         Derivative of ith global coord wrt jth local coord
%      dxidx(i,j)         Derivative of ith local coord wrt jth global coord
%      det                Determinant of jacobian
%      strain(i,j)        strain_ij components
%      dsde(i,j,k,l)      Derivative of stress_ij with respect:strain_kl
%      kel(row,col)       Rows && cols of element stiffness
%
%
   npoints = numberofintegrationpoints(ncoord,nelnodes,elident);
   dNdx = zeros(nelnodes,ncoord);
   dxdxi = zeros(ncoord,ncoord);
   strain = zeros(ndof,ncoord);
   kel = zeros(ndof*nelnodes,ndof*nelnodes);
%
%  Set up integration points && weights
%
   xilist = integrationpoints(ncoord,nelnodes,npoints,elident);
   w = integrationweights(ncoord,nelnodes,npoints,elident);
%
%  Loop over the integration points
%
   for intpt = 1:npoints

%     Compute shape functions && derivatives wrt local coords
%
      for i = 1:ncoord
        xi(i) = xilist(i,intpt);
      end
      N = shapefunctions(nelnodes,ncoord,elident,xi);
      dNdxi = shapefunctionderivs(nelnodes,ncoord,elident,xi);



%
%     Compute the jacobian matrix && its determinant
%
      for i = 1:ncoord
        for j = 1:ncoord
          dxdxi(i,j) = 0.;
          for a = 1:nelnodes
            dxdxi(i,j) = dxdxi(i,j) + coord(i,a)*dNdxi(a,j);
          end
        end
      end
```

```
      dxidx = inv(dxdxi);
      dt = det(dxdxi);
%
%     Convert shape function derivatives:derivatives wrt global coords
%
      for a = 1:nelnodes
        for i = 1:ncoord
          dNdx(a,i) = 0.;
          for j = 1:ncoord
            dNdx(a,i) = dNdx(a,i) + dNdxi(a,j)*dxidx(j,i);
          end
        end
      end
%
%     Compute the (infinitesimal) strain by differentiating displacements
%     This step is not really necessary for linear elasticity calculations
%     where stiffness is independent of strain.  It is included:allow
%     extension:nonlinear materials later.
%
      for i = 1:ncoord
         for j = 1:ncoord
            strain(i,j) = 0.;
            for a = 1:nelnodes
              strain(i,j) = strain(i,j) +
0.5*(displacement(i,a)*dNdx(a,j)+displacement(j,a)*dNdx(a,i));
            end
          end
      end
%
%     Compute the material tangent stiffness (d stress/d strain)
%     ds/de is just C_ijkl for linear elasticity - this notation is used
%     to allow extension to nonlinear problems
%
      dsde = materialstiffness(ndof,ncoord,strain,materialprops);
%
%     Compute the element stiffness
%
      for a = 1:nelnodes
        for i = 1:ndof
          for b = 1:nelnodes
            for k = 1:ndof
              row = ndof*(a-1)+i;
              col = ndof*(b-1)+k;
              for j = 1:ncoord
                for l = 1:ncoord
                  kel(col,row) = kel(col,row) +
dsde(i,j,k,l)*dNdx(b,l)*dNdx(a,j)*w(intpt)*dt;
                end
              end
            end
          end
        end
      end
   end
```

```
end


 %====================== No. nodes on element faces ===============
%
%   This procedure returns the number of nodes on each element face
%   for various element types.  This info is needed for computing
%   the surface integrals associated with the element traction vector
%
function n = nfacenodes(ncoord,nelnodes,elident,face)
   if (ncoord == 2)
     if (nelnodes == 3 || nelnodes == 4)
         n = 2;
     elseif (nelnodes == 6 || nelnodes == 8)
         n=3;
     end
   elseif (ncoord == 3)
     if (nelnodes == 4)
         n = 3;
     elseif (nelnodes == 10)
         n = 6;
     elseif (nelnodes == 8)
         n = 4;
     elseif (nelnodes == 20)
         n = 8;
     end
   end
end
%====================== Lists of nodes on element faces =============
%
%    This procedure returns the list of nodes on an element face
%    The nodes are ordered so that the element face forms either
%    a 1D line element or a 2D surface element for 2D or 3D problems
%
function list = facenodes(ncoord,nelnodes,elident,face)

   i3 = [2,3,1];
   i4 = [2,3,4,1];

   list = zeros(nfacenodes(ncoord,nelnodes,face),1);

   if (ncoord == 2)
     if (nelnodes == 3)
       list(1) = face;
       list(2) = i3(face);
     elseif (nelnodes == 6)
       list(1) = face;
       list(2) = i3(face);
       list(3) = face+3;
     elseif (nelnodes==4)
       list(1) = face;
       list(2) = i4(face);
     elseif (nelnodes==8)
       list(1) = face;
       list(2) = i4(face);
       list(3) = face+4;
```

```
    end
  elseif (ncoord == 3)
    if (nelnodes==4)
      if   (face == 1)
          list = [1,2,3];
      elseif (face == 2)
          list = [1,4,2];
      elseif (face == 3)
          list = [2,4,3];
      elseif (face == 4)
          list = [3,4,1];
      end
    elseif (nelnodes == 10)
      if   (face == 1)
          list = [1,2,3,5,6,7];
      elseif (face == 2)
          list = [1,4,2,8,9,5];
      elseif (face == 3)
          list = [2,4,3,9,10,6];
      elseif (face == 4)
          list = [3,4,1,10,8,7];
      end
    elseif (nelnodes == 8)
      if   (face == 1)
          list = [1,2,3,4];
      elseif (face == 2)
          list = [5,8,7,6];
      elseif (face == 3)
          list = [1,5,6,2];
      elseif (face == 4)
          list = [2,3,7,6];
      elseif (face == 5)
          list = [3,7,8,4];
      elseif (face == 6)
          list = [4,8,5,1];
      end
    elseif (nelnodes == 20)
      if   (face == 1)
          list = [1,2,3,4,9,10,11,12];
      elseif (face == 2)
          list = [5,8,7,6,16,15,14,13];
      elseif (face == 3)
          list = [1,5,6,2,17,13,18,9];
      elseif (face == 4)
          list = [2,6,7,3,18,14,19,10];
      elseif (face == 5)
          list = [3,7,8,4,19,15,20,11];
      elseif (face == 6)
          list = [4,8,5,1,20,16,17,12];
      end
    end
  end
end
%
%====================== ELEMENT DISTRIBUTED LOAD VECTOR ==============
%
function r = eldload(ncoord,ndof,nfacenodes,elident,coords,traction)
```

```
  npoints = numberofintegrationpoints(ncoord-1,nfacenodes);
  xi = zeros(ncoord-1,1);
  dxdxi = zeros(ncoord,ncoord-1);
  r = zeros(ndof*nfacenodes,1);

  xilist = integrationpoints(ncoord-1,nfacenodes,npoints);
  w = integrationweights(ncoord-1,nfacenodes,npoints);

  for intpt = 1:npoints

    for i = 1:ncoord-1
      xi(i) = xilist(i,intpt);
    end

    N = shapefunctions(nfacenodes,ncoord-1,elident,xi);
    dNdxi = shapefunctionderivs(nfacenodes,ncoord-1,elident,xi);
%
%     Compute the jacobian matrix && its determinant
%
    for i = 1:ncoord
      for j = 1:ncoord-1
        dxdxi(i,j) = 0.;
        for a = 1:nfacenodes
          dxdxi(i,j) = dxdxi(i,j) + coords(i,a)*dNdxi(a,j);
        end
      end
    end
    if (ncoord == 2)
      dt = sqrt(dxdxi(1,1)^2+dxdxi(2,1)^2);
    elseif (ncoord == 3)
      dt = sqrt( ((dxdxi(2,1)*dxdxi(3,2))-(dxdxi(2,2)*dxdxi(3,1)))^2 ...
          + ((dxdxi(1,1)*dxdxi(3,2))-(dxdxi(1,2)*dxdxi(3,1)))^2 ...
          + ((dxdxi(1,1)*dxdxi(2,2))-(dxdxi(1,2)*dxdxi(2,1)))^2 );
    end
   for a = 1:nfacenodes
      for i = 1:ndof
        row = ndof*(a-1)+i;
        r(row) = r(row) + N(a)*traction(i)*w(intpt)*dt;
      end
    end
  end
end
%
%====================== Assemble the global stiffness matrix
================
%
function Stif =
globalstiffness(ncoord,ndof,nnode,coords,nelem,maxnodes,elident,nelnodes,conn
ect,materialprops,dofs)
%
%   Assemble the global stiffness matrix
%

   Stif = zeros(ndof*nnode,ndof*nnode);
   lmncoord = zeros(ncoord,maxnodes);
```

```
   lmndof = zeros(ndof,maxnodes);
%
%    Loop over all the elements
%
   for lmn = 1:nelem
%
%    Extract coords of nodes, DOF for the current element
%
      for a = 1:nelnodes(lmn)
        for i = 1:ncoord
          lmncoord(i,a) = coords(i,connect(a,lmn));
        end
        for i = 1:ndof
          lmndof(i,a) = dofs(ndof*(connect(a,lmn)-1)+i);
        end
      end
    n = nelnodes(lmn);
    ident = elident(lmn);
    kel = elstif(ncoord,ndof,n,ident,lmncoord,materialprops,lmndof);
%
%    Add the current element stiffness:the global stiffness
%
    for a = 1:nelnodes(lmn)
      for i = 1:ndof
        for b = 1:nelnodes(lmn)
          for k = 1:ndof
            rw = ndof*(connect(a,lmn)-1)+i;
            cl = ndof*(connect(b,lmn)-1)+k;
            Stif(rw,cl) = Stif(rw,cl) + kel(ndof*(a-1)+i,ndof*(b-1)+k);
          end
        end
      end
    end
   end
end
%
%==================== Assemble the global traction vector ============
%
function r =
globaltraction(ncoord,ndof,nnodes,ndload,coords,nelnodes,elident,connect,dloa
ds,dofs)

   r = zeros(ndof*nnodes,1);
   traction = zeros(ndof,1);

   for load = 1:ndload
%
%     Extract the coords of the nodes on the appropriate element face
%
      lmn = dloads(1,load);
      face = dloads(2,load);
      n = nelnodes(lmn);
      ident = elident(lmn);
      nfnodes = nfacenodes(ncoord,n,ident,face);
      nodelist = facenodes(ncoord,n,ident,face);
      lmncoord = zeros(ncoord,nfnodes);
```

```
      for a = 1:nfnodes
        for i = 1:ncoord
          lmncoord(i,a) = coords(i,connect(nodelist(a),dloads(1,load)));
        end
        for i = 1:ndof
          lmndof(i,a) = dofs(ndof*(connect(nodelist(a),dloads(1,load))-1)+i);
        end
      end
%
%    Compute the element load vector
%
     for i = 1:ndof
       traction(i) = dloads(i+2,load);
     end

     rel = eldload(ncoord,ndof,nfnodes,ident,lmncoord,traction);
%
%    Assemble the element load vector into global vector
%
     for a = 1:nfnodes
       for i = 1:ndof
         rw = (connect(nodelist(a),dloads(1,load))-1)*ndof+i;
         r(rw) = r(rw) + rel((a-1)*ndof+i);
       end
     end

   end
end

function print_results(outfile, ...
    nprops,materialprops,ncoord,ndof,nnode,coords, ...
    nelem,maxnodes,connect,nelnodes,elident, ...
    nfix,fixnodes,ndload,dloads,dofs)
%      Print nodal displacements, element strains && stresses:a file

fprintf(outfile,'Nodal Displacements: \n');
   if (ndof == 2)
     fprintf(outfile,' Node      Coords         u1       u2 \n');
     for i = 1:nnode
      fprintf(outfile,'%3d %8.4f %8.4f %8.4f %8.4f\n', ...
                                i,coords(1,i),coords(2,i),dofs(2*i-
1),dofs(2*i));
     end
   elseif (ndof == 3)
     fprintf(outfile,' Node            Coords            u1       u2       u3
\n');
     for i = 1:nnode
      fprintf(outfile,'%3d %8.4f %8.4f %8.4f %8.4f %8.4f %8.4f \n', ...
                    i,coords(1,i),coords(2,i),coords(3,i),dofs(3*i-
2),dofs(3*i-1),dofs(3*i));
     end
   end

   fprintf(outfile,'\n\n Strains and Stresses \n');
```

```
   lmncoord = zeros(ncoord,maxnodes);
   displacements = zeros(ndof,maxnodes);

%
%   Loop over all the elements
%
   for lmn = 1:nelem

    fprintf(outfile,' \n Element; %d ',lmn);
    if (ncoord == 2)
    fprintf(outfile,'  \n int pt    Coords          e_11      e_22     e_12
s_11       s_22      s_12 \n');

    elseif (ncoord == 3)
    fprintf(outfile,'\n int pt         Coords            e_11      e_22
e_33      e_12       e_13      e_23      s_11      s_22      s_33      s_12
s_13      s_23 \n');
    end
%
%   Extract coords of nodes, DOF for the current element
%
      for a = 1:nelnodes(lmn)
        for i = 1:ncoord
          lmncoord(i,a) = coords(i,connect(a,lmn));
        end
        for i = 1:ndof
          displacements(i,a) = dofs(ndof*(connect(a,lmn)-1)+i);
        end
      end
      n = nelnodes(lmn);
      ident = elident(lmn);

      npoints = numberofintegrationpoints(ncoord,n);
      dNdx = zeros(n,ncoord);
      dxdxi = zeros(ncoord,ncoord);
      strain = zeros(ndof,ncoord);
      xi = zeros(ncoord,1);
      x = zeros(ncoord,1);
%
%  Set up integration points
%
      xilist = integrationpoints(ncoord,n,npoints);
%
%  Loop over the integration points
%
     for intpt = 1:npoints

%     Compute shape functions && derivatives wrt local coords
%
       for i = 1:ncoord
         xi(i) = xilist(i,intpt);
       end
       N = shapefunctions(n,ncoord,ident,xi);
       dNdxi = shapefunctionderivs(n,ncoord,ident,xi);
%
```

```
%     Compute the coords of the integration point
%
      for i = 1:ncoord
        x(i) = 0.;
        for a = 1:n
          x(i) = x(i) + lmncoord(i,a)*N(a);
        end
      end
%
%     Compute the jacobian matrix && its determinant
%
      for i = 1:ncoord
        for j = 1:ncoord
          dxdxi(i,j) = 0.;
          for a = 1:n
            dxdxi(i,j) = dxdxi(i,j) + lmncoord(i,a)*dNdxi(a,j);
          end
        end
      end

      dxidx = inv(dxdxi);
%
%     Convert shape function derivatives:derivatives wrt global coords
%
      for a = 1:n
        for i = 1:ncoord
          dNdx(a,i) = 0.;
          for j = 1:ncoord
            dNdx(a,i) = dNdx(a,i) + dNdxi(a,j)*dxidx(j,i);
          end
        end
      end
%
%     Compute the (infinitesimal) strain by differentiating displacements
%
      for i = 1:ncoord
         for j = 1:ncoord
            strain(i,j) = 0.;
            for a = 1:n
              strain(i,j) = strain(i,j) +
0.5*(displacements(i,a)*dNdx(a,j)+displacements(j,a)*dNdx(a,i));
            end
          end
      end

      stress = materialstress(ndof,ncoord,strain,materialprops);

      if (ncoord == 2)

      fprintf(outfile,'%5d %7.4f %7.4f %9.4f %9.4f %9.4f %9.4f %9.4f %9.4f
\n', ...

intpt,x(1),x(2),strain(1,1),strain(2,2),strain(1,2),stress(1,1),stress(2,2),s
tress(1,2));
```

```
      elseif (ncoord == 3)

      fprintf(outfile,'%5d %7.4f %7.4f %7.4f %9.4f %9.4f %9.4f %9.4f %9.4f
%9.4f %9.4f %9.4f %9.4f %9.4f %9.4f %9.4f \n',...
              intpt,x(1),x(2),x(3), ...

strain(1,1),strain(2,2),strain(3,3),strain(1,2),strain(1,3),strain(2,3), ...

stress(1,1),stress(2,2),stress(3,3),stress(1,2),stress(1,3),stress(2,3));
      end
     end
   end
end

%==============================Function read_input_file ================
 function [nprops,materialprops,ncoord,ndof,nnode,coords,nelem,maxnodes, ...
                connect,nelnodes,elident,nfix,fixnodes,ndload,dloads] =
read_input_file(infile)

 cellarray=textscan(infile,'%s');
 %cellarray=fopen(infile,'%s');

%  Total no. material parameters, && list of parameters
%
   nprops = str2num(cellarray{1}{2});
   materialprops = zeros(nprops,1);
   cellno = 2;
   for i = 1:nprops
     cellno = cellno + 2;
     materialprops(i) = str2num(cellarray{1}{cellno});
   end;
%
%    no. coords (1:3), no. DOF, no. nodes && nodal coordinates
%
   cellno = cellno + 2;
   ncoord = str2num(cellarray{1}{cellno});
   cellno = cellno + 2;
   ndof = str2num(cellarray{1}{cellno});
   cellno = cellno + 2;
   nnode = str2num(cellarray{1}{cellno});

   coords = zeros(ncoord,nnode);
   cellno = cellno + 1;
   for i = 1 : nnode
     for j = 1 : ncoord
       cellno = cellno + 1;
       coords(j,i) = str2num(cellarray{1}{cellno});
     end;
   end;
%
%    No. elements && connectivity
%
   cellno = cellno + 2;
   nelem = str2num(cellarray{1}{cellno});
   cellno = cellno + 2;
   maxnodes = str2num(cellarray{1}{cellno});
```

```
   connect = zeros(maxnodes,nelem);
   nelnodes = zeros(nelem,1);
   elident = zeros(nelem,1);
   cellno = cellno + 3;
   for i = 1 : nelem
     cellno = cellno + 1;
     elident(i) = str2num(cellarray{1}{cellno});
     cellno = cellno + 1;
     nelnodes(i) = str2num(cellarray{1}{cellno});
     for j = 1 : nelnodes(i)
       cellno = cellno + 1;
       connect(j,i) = str2num(cellarray{1}{cellno});
     end
   end
%
%    No. nodes with prescribed displacements, with the prescribed
displacements
%
   cellno = cellno + 2;
   nfix = str2num(cellarray{1}{cellno});
   cellno = cellno + 3;
   fixnodes = zeros(3,nfix);
   for i = 1 : nfix
      cellno = cellno + 1;
      fixnodes(1,i) = str2num(cellarray{1}{cellno});
      cellno = cellno + 1;
      fixnodes(2,i) = str2num(cellarray{1}{cellno});
      cellno = cellno + 1;
      fixnodes(3,i) = str2num(cellarray{1}{cellno});
   end;
%
%    No. loaded element faces, with the loads
%
    cellno = cellno + 2;
    ndload = str2num(cellarray{1}{cellno});
    cellno = cellno + 3;
    dloads = zeros(2+ndof,ndload);
    for i = 1 : ndload
       cellno = cellno + 1;
       dloads(1,i) = str2num(cellarray{1}{cellno});
       cellno = cellno + 1;
       dloads(2,i) = str2num(cellarray{1}{cellno});
       for j = 1 : ndof
         cellno = cellno + 1;
         dloads(j+2,i) = str2num(cellarray{1}{cellno});
       end;
    end;

 end
  function plotmesh(coords,ncoord,nnode,connect,nelem,elident,nelnodes,color)
% Function to plot a mesh.
    f2D_3 = [1,2,3];
    f2D_4 = [1,2,3,4];
    f2D_6 = [1,4,2,5,3,6];
    f2D_8 = [1,5,2,6,3,7,4,8];
    f3D_4 = [[1,2,3];[1,4,2];[2,4,3];[3,4,1]];
    f3D_10 = [[1,5,2,6,3,7];[1,8,4,9,2,5];[2,9,4,10,3,6];[3,10,4,8,1,7]];
```

```
    f3D_8 = [[1,2,3,4];[5,8,7,6];[1,5,6,2];[2,3,7,6];[3,7,8,4];[4,8,5,1]];
    f3D_20 = [[1,9,2,10,3,11,4,12];[5,16,8,15,7,14,6,13];
              [1,17,5,13,6,18,2,9];[2,18,6,14,7,19,3,10];
              [3,19,7,15,8,20,4,11];[4,20,8,16,5,17,1,12]];

   hold on
   if (ncoord==2)  % Plot a 2D mesh
       for lmn = 1:nelem
           for i = 1:nelnodes(lmn)
               x(i,1:2) = coords(1:2,connect(i,lmn));
           end
           scatter(x(:,1),x(:,2),'MarkerFaceColor','r');
           if (nelnodes(lmn)==3)

patch('Vertices',x,'Faces',f2D_3,'FaceColor','none','EdgeColor',color);
           elseif (nelnodes(lmn)==4)

patch('Vertices',x,'Faces',f2D_4,'FaceColor','none','EdgeColor',color);
           elseif (nelnodes(lmn)==6)

patch('Vertices',x,'Faces',f2D_6,'FaceColor','none','EdgeColor',color);
           elseif (nelnodes(lmn)==8 || nelnodes(lmn)==9)

patch('Vertices',x,'Faces',f2D_8,'FaceColor','none','EdgeColor',color);
           end
       end
   elseif (ncoord==3) % Plot a 3D mesh
       for lmn = 1:nelem
           for i = 1:nelnodes(lmn)
               x(i,1:3) = coords(1:3,connect(i,lmn));
           end
           scatter3(x(:,1),x(:,2),x(:,3),'MarkerFaceColor','r');
           if (nelnodes(lmn)==4)

patch('Vertices',x,'Faces',f3D_4,'FaceColor','none','EdgeColor',color);
           elseif (nelnodes(lmn)==10)

patch('Vertices',x,'Faces',f3D_10,'FaceColor','none','EdgeColor',color);
           elseif (nelnodes(lmn)==8)

patch('Vertices',x,'Faces',f3D_8,'FaceColor','none','EdgeColor',color);
           elseif (nelnodes(lmn)==20)

patch('Vertices',x,'Faces',f3D_20,'FaceColor','none','EdgeColor',color);
           end
            title('green undeformed plot, red is deformed')
       end
   end
   axis equal
   hold off
end
```

Appendix

Results for 0,0,0 angle:

Nodal Displacements:

Node	Coords		u1	u2	u3	
1	0.0000	-1.0000	0.0000	-0.0000	0.0000	-0.0000
2	1.0000	-1.0000	0.0000	0.0000	0.0000	-0.0000
3	2.0000	-1.0000	0.0000	0.0000	0.0000	-0.0000
4	3.0000	-1.0000	0.0000	0.0000	0.0000	-0.0000
5	0.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000
6	1.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000
7	2.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000
8	3.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000
9	0.0000	1.0000	0.0000	0.0000	-0.0000	-0.0000
10	1.0000	1.0000	0.0000	0.0000	-0.0000	-0.0000
11	2.0000	1.0000	0.0000	0.0000	-0.0000	-0.0000
12	3.0000	1.0000	0.0000	0.0000	-0.0000	-0.0000
13	0.0000	-1.0000	1.0000	0.0000	-0.0000	-0.0000
14	1.0000	-1.0000	1.0000	0.0000	0.0000	-0.0000
15	2.0000	-1.0000	1.0000	0.0000	0.0000	-0.0000
16	3.0000	-1.0000	1.0000	0.0000	0.0000	-0.0000
17	0.0000	0.0000	1.0000	-0.0000	-0.0000	-0.0000
18	1.0000	0.0000	1.0000	0.0000	-0.0000	-0.0000
19	2.0000	0.0000	1.0000	0.0000	-0.0000	-0.0000
20	3.0000	0.0000	1.0000	0.0000	-0.0000	-0.0000
21	0.0000	1.0000	1.0000	-0.0000	-0.0000	-0.0000
22	1.0000	1.0000	1.0000	0.0000	-0.0000	-0.0000
23	2.0000	1.0000	1.0000	0.0000	-0.0000	-0.0000
24	3.0000	1.0000	1.0000	0.0000	-0.0000	-0.0000

Strains and Stresses

Element; 1

int pt	Coords	e_11	e_22	e_33	e_12	e_13	e_23	s_11	s_22	s_33	s_12	s_13	s_23
1	0.2113 -0.7887 0.2113	0.0000	-0.0000	-0.0000	0.0000	0.0000	-0.0000	1.0000	-0.0000	0.0000	0.0000	0.0000	-0.0000
2	0.7887 -0.7887 0.2113	0.0000	-0.0000	-0.0000	-0.0000	0.0000	-0.0000	1.0000	0.0000	0.0000	-0.0000	0.0000	-0.0000
3	0.2113 -0.2113 0.2113	0.0000	-0.0000	-0.0000	0.0000	-0.0000	-0.0000	1.0000	-0.0000	-0.0000	0.0000	-0.0000	-0.0000
4	0.7887 -0.2113 0.2113	0.0000	-0.0000	-0.0000	0.0000	-0.0000	-0.0000	1.0000	-0.0000	-0.0000	0.0000	-0.0000	-0.0000
5	0.2113 -0.7887 0.7887	0.0000	-0.0000	-0.0000	0.0000	0.0000	-0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	-0.0000
6	0.7887 -0.7887 0.7887	0.0000	-0.0000	-0.0000	0.0000	0.0000	-0.0000	1.0000	0.0000	-0.0000	0.0000	0.0000	-0.0000
7	0.2113 -0.2113 0.7887	0.0000	-0.0000	-0.0000	0.0000	-0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	-0.0000	0.0000
8	0.7887 -0.2113 0.7887	0.0000	-0.0000	-0.0000	0.0000	-0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	-0.0000	0.0000

Element; 2

int pt	Coords	e_11	e_22	e_33	e_12	e_13	e_23	s_11	s_22	s_33	s_12	s_13	s_23
1	1.2113 -0.7887 0.2113	0.0000	-0.0000	-0.0000	0.0000	-0.0000	0.0000	1.0000	-0.0000	-0.0000	0.0000	-0.0000	0.0000
2	1.7887 -0.7887 0.2113	0.0000	-0.0000	-0.0000	0.0000	-0.0000	0.0000	1.0000	-0.0000	-0.0000	0.0000	-0.0000	0.0000
3	1.2113 -0.2113 0.2113	0.0000	-0.0000	-0.0000	-0.0000	0.0000	-0.0000	1.0000	0.0000	0.0000	-0.0000	0.0000	-0.0000

4 1.7887 -0.2113 0.2113 0.0000 -0.0000 -0.0000 0.0000 0.0000 -0.0000 1.0000 0.0000 0.0000 0.0000 0.0000 -0.0000

5 1.2113 -0.7887 0.7887 0.0000 -0.0000 -0.0000 0.0000 -0.0000 0.0000 1.0000 -0.0000 -0.0000 0.0000 -0.0000 0.0000

6 1.7887 -0.7887 0.7887 0.0000 -0.0000 -0.0000 -0.0000 0.0000 0.0000 1.0000 0.0000 0.0000 -0.0000 0.0000 0.0000

7 1.2113 -0.2113 0.7887 0.0000 -0.0000 -0.0000 0.0000 -0.0000 0.0000 1.0000 -0.0000 -0.0000 0.0000 -0.0000 0.0000

8 1.7887 -0.2113 0.7887 0.0000 -0.0000 -0.0000 0.0000 -0.0000 0.0000 1.0000 -0.0000 -0.0000 0.0000 -0.0000 0.0000

Element; 3

int pt Coords e_11 e_22 e_33 e_12 e_13 e_23 s_11 s_22 s_33 s_12 s_13 s_23

1 2.2113 -0.7887 0.2113 0.0000 -0.0000 -0.0000 0.0000 0.0000 -0.0000 1.0000 0.0000 0.0000 0.0000 0.0000 -0.0000

2 2.7887 -0.7887 0.2113 0.0000 -0.0000 -0.0000 0.0000 0.0000 -0.0000 1.0000 -0.0000 -0.0000 0.0000 0.0000 -0.0000

3 2.2113 -0.2113 0.2113 0.0000 -0.0000 -0.0000 0.0000 0.0000 0.0000 1.0000 0.0000 -0.0000 0.0000 0.0000 0.0000

4 2.7887 -0.2113 0.2113 0.0000 -0.0000 -0.0000 0.0000 -0.0000 0.0000 1.0000 0.0000 0.0000 0.0000 -0.0000 0.0000

5 2.2113 -0.7887 0.7887 0.0000 -0.0000 -0.0000 -0.0000 0.0000 -0.0000 1.0000 0.0000 0.0000 -0.0000 0.0000 -0.0000

6 2.7887 -0.7887 0.7887 0.0000 -0.0000 -0.0000 0.0000 0.0000 -0.0000 1.0000 -0.0000 -0.0000 0.0000 0.0000 -0.0000

7 2.2113 -0.2113 0.7887 0.0000 -0.0000 -0.0000 -0.0000 0.0000 -0.0000 1.0000 0.0000 -0.0000 -0.0000 0.0000 -0.0000

8 2.7887 -0.2113 0.7887 0.0000 -0.0000 -0.0000 0.0000 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000

Element; 4

int pt	Coords			e_11	e_22	e_33	e_12	e_13	e_23	s_11	s_22	s_33	s_12	s_13	s_23
1	0.2113	0.2113	0.2113	0.0000	-0.0000	-0.0000	-0.0000	-0.0000	0.0000	1.0000	-0.0000	-0.0000	-0.0000	-0.0000	0.0000
2	0.7887	0.2113	0.2113	0.0000	-0.0000	-0.0000	-0.0000	-0.0000	0.0000	1.0000	-0.0000	-0.0000	-0.0000	-0.0000	0.0000
3	0.2113	0.7887	0.2113	0.0000	-0.0000	-0.0000	-0.0000	-0.0000	-0.0000	1.0000	0.0000	-0.0000	-0.0000	-0.0000	-0.0000
4	0.7887	0.7887	0.2113	0.0000	-0.0000	-0.0000	-0.0000	-0.0000	-0.0000	1.0000	-0.0000	-0.0000	-0.0000	-0.0000	-0.0000
5	0.2113	0.2113	0.7887	0.0000	-0.0000	-0.0000	-0.0000	-0.0000	0.0000	1.0000	0.0000	0.0000	-0.0000	-0.0000	0.0000
6	0.7887	0.2113	0.7887	0.0000	-0.0000	-0.0000	-0.0000	-0.0000	-0.0000	1.0000	0.0000	0.0000	-0.0000	-0.0000	-0.0000
7	0.2113	0.7887	0.7887	0.0000	-0.0000	-0.0000	-0.0000	-0.0000	-0.0000	1.0000	0.0000	0.0000	-0.0000	-0.0000	-0.0000
8	0.7887	0.7887	0.7887	0.0000	-0.0000	-0.0000	0.0000	-0.0000	-0.0000	1.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000

Element; 5

int pt	Coords			e_11	e_22	e_33	e_12	e_13	e_23	s_11	s_22	s_33	s_12	s_13	s_23
1	1.2113	0.2113	0.2113	0.0000	-0.0000	-0.0000	0.0000	0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2	1.7887	0.2113	0.2113	0.0000	-0.0000	-0.0000	0.0000	0.0000	-0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	-0.0000
3	1.2113	0.7887	0.2113	0.0000	-0.0000	-0.0000	0.0000	0.0000	-0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	-0.0000
4	1.7887	0.7887	0.2113	0.0000	-0.0000	-0.0000	0.0000	0.0000	-0.0000	1.0000	-0.0000	-0.0000	0.0000	0.0000	-0.0000
5	1.2113	0.2113	0.7887	0.0000	-0.0000	-0.0000	0.0000	0.0000	-0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	-0.0000
6	1.7887	0.2113	0.7887	0.0000	-0.0000	-0.0000	0.0000	-0.0000	0.0000	1.0000	-0.0000	-0.0000	0.0000	-0.0000	0.0000

7 1.2113 0.7887 0.7887 0.0000 -0.0000 -0.0000 0.0000 0.0000 -0.0000 1.0000 -0.0000 -0.0000 0.0000 0.0000 -0.0000

8 1.7887 0.7887 0.7887 0.0000 -0.0000 -0.0000 0.0000 0.0000 0.0000 1.0000 -0.0000 -0.0000 0.0000 0.0000 0.0000

Element; 6

int pt Coords e_11 e_22 e_33 e_12 e_13 e_23 s_11 s_22 s_33 s_12 s_13 s_23

1 2.2113 0.2113 0.2113 0.0000 -0.0000 -0.0000 0.0000 0.0000 -0.0000 1.0000 -0.0000 -0.0000 0.0000 0.0000 -0.0000

2 2.7887 0.2113 0.2113 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 1.0000 0.0000 0.0000 -0.0000 -0.0000 0.0000

3 2.2113 0.7887 0.2113 0.0000 -0.0000 -0.0000 0.0000 0.0000 -0.0000 1.0000 0.0000 0.0000 0.0000 0.0000 -0.0000

4 2.7887 0.7887 0.2113 0.0000 -0.0000 -0.0000 0.0000 0.0000 -0.0000 1.0000 -0.0000 -0.0000 0.0000 0.0000 -0.0000

5 2.2113 0.2113 0.7887 0.0000 -0.0000 -0.0000 0.0000 0.0000 0.0000 1.0000 0.0000 -0.0000 0.0000 0.0000 0.0000

6 2.7887 0.2113 0.7887 0.0000 -0.0000 -0.0000 0.0000 0.0000 0.0000 1.0000 -0.0000 -0.0000 0.0000 0.0000 0.0000

7 2.2113 0.7887 0.7887 0.0000 -0.0000 -0.0000 0.0000 -0.0000 0.0000 1.0000 0.0000 0.0000 0.0000 -0.0000 0.0000

8 2.7887 0.7887 0.7887 0.0000 -0.0000 -0.0000 0.0000 -0.0000 0.0000 1.0000 -0.0000 -0.0000 0.0000 -0.0000 0.0000

Results for 45,45,45

Nodal Displacements:

Node Coords u1 u2 u3

1 0.0000 -1.0000 0.0000 -0.0000 -0.0000 -0.0000

2 1.0000 -1.0000 0.0000 0.0000 -0.0000 -0.0000

3 2.0000 -1.0000 0.0000 0.0000 -0.0000 -0.0000

4 3.0000 -1.0000 0.0000 0.0000 -0.0000 -0.0000

5 0.0000 0.0000 0.0000 -0.0000 -0.0000 0.0000

6	1.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000
7	2.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000
8	3.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000
9	0.0000	1.0000	0.0000	0.0000	-0.0000	0.0000
10	1.0000	1.0000	0.0000	0.0000	-0.0000	0.0000
11	2.0000	1.0000	0.0000	0.0000	-0.0000	-0.0000
12	3.0000	1.0000	0.0000	0.0000	-0.0000	-0.0000
13	0.0000	-1.0000	1.0000	0.0000	0.0000	-0.0000
14	1.0000	-1.0000	1.0000	0.0000	0.0000	-0.0000
15	2.0000	-1.0000	1.0000	0.0000	0.0000	-0.0000
16	3.0000	-1.0000	1.0000	0.0000	-0.0000	-0.0000
17	0.0000	0.0000	1.0000	0.0000	-0.0000	-0.0000
18	1.0000	0.0000	1.0000	0.0000	-0.0000	-0.0000
19	2.0000	0.0000	1.0000	0.0000	-0.0000	-0.0000
20	3.0000	0.0000	1.0000	0.0000	-0.0000	-0.0000
21	0.0000	1.0000	1.0000	0.0000	-0.0000	-0.0000
22	1.0000	1.0000	1.0000	0.0000	-0.0000	-0.0000
23	2.0000	1.0000	1.0000	0.0000	-0.0000	-0.0000
24	3.0000	1.0000	1.0000	0.0000	-0.0000	-0.0000

Strains and Stresses

Element; 1

int pt	Coords		e_11	e_22	e_33	e_12	e_13	e_23	s_11	s_22	s_33	s_12	s_13	s_23
1	0.2113	-0.7887	0.2113	0.0000	-0.0000	-0.0000	-0.0000	-0.0000	0.0000	1.0759	0.0057	0.0233	-0.0001	0.0486	-0.0265

2 0.7887 -0.7887 0.2113 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 1.0851 0.0393 0.0428 -0.0019 0.0183 -0.0101

3 0.2113 -0.2113 0.2113 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 1.0604 0.0020 0.0187 0.0032 0.0473 -0.0119

4 0.7887 -0.2113 0.2113 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 1.0697 0.0320 0.0365 0.0026 0.0181 -0.0055

5 0.2113 -0.7887 0.7887 0.0000 -0.0000 -0.0000 0.0000 -0.0000 0.0000 0.9657 -0.0104 -0.0230 0.0145 0.0393 -0.0198

6 0.7887 -0.7887 0.7887 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 0.9411 -0.0320 -0.0352 0.0131 0.0102 -0.0061

7 0.2113 -0.2113 0.7887 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 0.9525 -0.0127 -0.0254 0.0057 0.0389 -0.0040

8 0.7887 -0.2113 0.7887 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 0.9280 -0.0379 -0.0393 0.0056 0.0110 -0.0002

Element; 2

int pt Coords e_11 e_22 e_33 e_12 e_13 e_23 s_11 s_22 s_33 s_12 s_13 s_23

1 1.2113 -0.7887 0.2113 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 1.0030 0.0002 0.0009 0.0016 0.0005 -0.0018

2 1.7887 -0.7887 0.2113 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 1.0032 0.0015 0.0008 0.0007 0.0005 -0.0002

3 1.2113 -0.2113 0.2113 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 1.0014 0.0001 0.0021 0.0016 0.0010 -0.0009

4 1.7887 -0.2113 0.2113 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 1.0006 0.0001 0.0002 0.0009 0.0006 -0.0001

5 1.2113 -0.7887 0.7887 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 1.0066 0.0060 0.0041 0.0040 -0.0000 -0.0013

6 1.7887 -0.7887 0.7887 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 1.0039 0.0024 0.0013 0.0024 0.0002 -0.0001

7 1.2113 -0.2113 0.7887 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 1.0022 0.0041 0.0037 0.0030 0.0001 -0.0002

8 1.7887 -0.2113 0.7887 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 0.9984 -0.0006 -0.0009 0.0017 -0.0000 0.0002

Element; 3

int pt	Coords		e_11	e_22	e_33	e_12	e_13	e_23	s_11	s_22	s_33	s_12	s_13	s_23
1	2.2113	-0.7887	0.2113	0.0000	-0.0000	-0.0000	-0.0000	-0.0000	0.0000	1.0005	0.0003	-0.0006	0.0006	-0.0000	0.0004
2	2.7887	-0.7887	0.2113	0.0000	-0.0000	-0.0000	-0.0000	-0.0000	0.0000	1.0008	0.0004	0.0001	0.0004	0.0001	0.0001
3	2.2113	-0.2113	0.2113	0.0000	-0.0000	-0.0000	-0.0000	-0.0000	0.0000	0.9999	0.0000	-0.0005	0.0006	-0.0001	0.0002
4	2.7887	-0.2113	0.2113	0.0000	-0.0000	-0.0000	-0.0000	-0.0000	0.0000	1.0000	-0.0001	-0.0002	0.0003	-0.0000	0.0001
5	2.2113	-0.7887	0.7887	0.0000	-0.0000	-0.0000	-0.0000	-0.0000	0.0000	1.0002	-0.0004	-0.0009	0.0003	0.0002	0.0004
6	2.7887	-0.7887	0.7887	0.0000	-0.0000	-0.0000	-0.0000	-0.0000	0.0000	1.0010	0.0004	0.0002	0.0001	0.0002	0.0001
7	2.2113	-0.2113	0.7887	0.0000	-0.0000	-0.0000	-0.0000	-0.0000	0.0000	0.9998	-0.0006	-0.0008	0.0004	0.0000	0.0003
8	2.7887	-0.2113	0.7887	0.0000	-0.0000	-0.0000	-0.0000	-0.0000	0.0000	1.0003	0.0000	-0.0000	0.0002	0.0001	0.0001

Element; 4

int pt	Coords		e_11	e_22	e_33	e_12	e_13	e_23	s_11	s_22	s_33	s_12	s_13	s_23
1	0.2113	0.2113	0.2113	0.0000	-0.0000	-0.0000	-0.0000	-0.0000	0.0000	1.0689	0.0308	0.0337	0.0043	0.0455	-0.0011
2	0.7887	0.2113	0.2113	0.0000	-0.0000	-0.0000	-0.0000	-0.0000	0.0000	1.0639	0.0355	0.0370	0.0046	0.0174	-0.0009
3	0.2113	0.7887	0.2113	0.0000	-0.0000	-0.0000	-0.0000	-0.0000	0.0000	1.0553	0.0255	0.0300	0.0038	0.0442	0.0054
4	0.7887	0.7887	0.2113	0.0000	-0.0000	-0.0000	-0.0000	-0.0000	0.0000	1.0494	0.0287	0.0314	0.0043	0.0167	0.0036

5 0.2113 0.2113 0.7887 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 0.9329 -0.0313 -0.0369 0.0013 0.0386 0.0053

6 0.7887 0.2113 0.7887 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 0.9215 -0.0369 -0.0395 0.0021 0.0108 0.0046

7 0.2113 0.7887 0.7887 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 0.9210 -0.0354 -0.0394 -0.0014 0.0373 0.0119

8 0.7887 0.7887 0.7887 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 0.9088 -0.0425 -0.0439 -0.0004 0.0100 0.0094

Element; 5

int pt Coords e_11 e_22 e_33 e_12 e_13 e_23 s_11 s_22 s_33 s_12 s_13 s_23

1 1.2113 0.2113 0.2113 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 0.9999 0.0004 0.0028 0.0022 0.0010 0.0008

2 1.7887 0.2113 0.2113 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 0.9994 0.0009 0.0006 0.0014 0.0004 0.0002

3 1.2113 0.7887 0.2113 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 0.9963 -0.0004 0.0026 0.0023 0.0008 0.0036

4 1.7887 0.7887 0.2113 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 0.9952 -0.0012 -0.0009 0.0019 -0.0000 0.0006

5 1.2113 0.2113 0.7887 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 1.0048 0.0133 0.0091 0.0020 -0.0002 0.0021

6 1.7887 0.2113 0.7887 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 0.9963 0.0006 -0.0005 0.0006 -0.0004 0.0008

7 1.2113 0.7887 0.7887 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 0.9990 0.0111 0.0079 -0.0007 -0.0005 0.0052

8 1.7887 0.7887 0.7887 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 0.9899 -0.0028 -0.0030 -0.0017 -0.0009 0.0016

Element; 6

int pt Coords e_11 e_22 e_33 e_12 e_13 e_23 s_11 s_22 s_33 s_12 s_13 s_23

1 2.2113 0.2113 0.2113 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 1.0006 0.0018 0.0007 0.0005 -0.0002 0.0000

2 2.7887 0.2113 0.2113 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 0.9997 0.0002 -0.0000 0.0002 -0.0001 0.0001

3 2.2113 0.7887 0.2113 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 0.9997 0.0013 0.0008 0.0002 -0.0003 -0.0003

4 2.7887 0.7887 0.2113 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 0.9986 -0.0004 -0.0004 -0.0002 -0.0001 0.0001

5 2.2113 0.2113 0.7887 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 1.0000 0.0002 -0.0001 0.0005 -0.0001 0.0001

6 2.7887 0.2113 0.7887 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 1.0002 0.0004 0.0002 0.0003 -0.0000 0.0001

7 2.2113 0.7887 0.7887 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 0.9994 -0.0001 0.0001 0.0006 -0.0003 -0.0003

8 2.7887 0.7887 0.7887 0.0000 -0.0000 -0.0000 -0.0000 -0.0000 0.0000 0.9994 -0.0000 -0.0000 0.0003 -0.0001 0.0001