Homework 1 Solutions

February 8, 2017

1 Problem 1

Composites material properties are usually calibrated in the material coordinate system, with the base vectors denoted as a_i. To carry out analysis, we usually have to set up problem coordinate system denoted as e_i. In the most general case, the material coordinate system can be obtained through three consecutive rotations from the problem coordinate system. Let us assume we rotate e_i about e_1 by θ_1 to become e'_i. This is followed by a rotation of e'_i about e'_2 by θ_2 to become e''_i. Finally, we rotate rotate e''_i about e''_3 by θ_3 to give a_i. Find the direction cosine matrix β_{ij} such that $e_i = \beta_{ij} a_j$.

1.1 Solution

With the convention that

$$Q_{ij} = \cos(x'_i, x_j)$$

We know that

$$e'_i = Q_{ij} e_j$$

We can denote the rotation matrix, Q_{ij} for each successive transformation with a superscript, giving the following equations

$$e'_i = Q_{ij} e_j \qquad e''_i = Q^2_{ij} e'_j \qquad a_i = Q^3_{ij} e''_j$$

We desire to relate a_i to e_i directly, with one effective rotation, β_{ij}. We can do this by substituting the three equations

$$a_i = Q^3_{ij} Q^2_{jk} Q^3_{kl} e_l = [Q^3][Q^2][Q^1]e$$

In this problem, however, we are tasked with finding the inverse relationship, $e_i = \beta_{ij} a_j$, left multiply by Q^{3T}, Q^{2T}, Q^{1T} in that order.

$$Q^1_{ji} Q^2_{kj} Q^3_{lk} a_l = e_l = [Q^{1T}][Q^{2T}][Q^{3T}]$$

to find $\beta_{ij} = Q^1_{ji} Q^2_{kj} Q^3_{lk}$

We can expand this symbolically

```
In [15]: import sympy as sm #python symbolic math library
         sm.init_printing() #render output
         t1, t2, t3 = sm.symbols(r'\theta_1 \theta_2 \theta_3') #rotation angles
         Q1 = sm.Matrix([[1,0,0],
                         [0,sm.cos(t1),sm.sin(t1)],
```

```
                     [0,-sm.sin(t1),sm.cos(t1)]])
#Q1*sm.Matrix([0,0,1]) #check rotation signs
Q2 = sm.Matrix([[sm.cos(t2),0,-sm.sin(t2)],
                [0,1,0],
                [sm.sin(t2),0,sm.cos(t2)]])
#Q2*sm.Matrix([1,0,0]) #check rotation signs
Q3 = sm.Matrix([[sm.cos(t3), sm.sin(t3),0],
                [-sm.sin(t3), sm.cos(t3),0],
                [0,0,1]])
#Q3*sm.Matrix([0,1,0]) #check rotation signs
beta = Q1.T*Q2.T*Q3.T #.T gives transpose in python
beta
```

Out[15]:

$$\left[\begin{matrix}\cos{\left(\theta_{2} \right)} \cos{\left(\theta_{3} \right)} & - \sin{\left(\theta_{3} \right)} \cos{\left(\theta_{2} \right)} & \sin{\left(\theta_{2} \right)}\\\sin{\left(\theta_{1} \right)} \sin{\left(\theta_{2} \right)} \cos{\left(\theta_{3} \right)} + \sin{\left(\theta_{3} \right)} \cos{\left(\theta_{1} \right)} & - \sin{\left(\theta_{1} \right)} \sin{\left(\theta_{2} \right)} \sin{\left(\theta_{3} \right)} + \cos{\left(\theta_{1} \right)} \cos{\left(\theta_{3} \right)} & - \sin{\left(\theta_{1} \right)} \cos{\left(\theta_{2} \right)}\\\sin{\left(\theta_{1} \right)} \sin{\left(\theta_{3} \right)} - \sin{\left(\theta_{2} \right)} \cos{\left(\theta_{1} \right)} \cos{\left(\theta_{3} \right)} & \sin{\left(\theta_{1} \right)} \cos{\left(\theta_{3} \right)} + \sin{\left(\theta_{2} \right)} \sin{\left(\theta_{3} \right)} \cos{\left(\theta_{1} \right)} & \cos{\left(\theta_{1} \right)} \cos{\left(\theta_{2} \right)}\end{matrix}\right]$$

1.2 Problem 2

Consider a unidirectional fiber-reinforced composite as linearly elastic, orthotropic materials with

Property	Value
E_1	20 Mpsi
E_2	1.4 Mpsi
E_3	2.5 Mpsi
G_{12}	0.8 Mpsi
G_{13}	2.0 Mpsi
G_{23}	1.5 Mpsi
ν_{12}	0.2
ν_{13}	0.3
ν_{23}	0.25

Where the 1-direction is in the fiber direction, the 2-direction is normal to the fibers in the plane, and the 3-direction is normal to the plane. Four $45°$ laminae are used to make a box beam ($45°$ refers to the rotation about the outward normal at each wall). Compute the 6x6 stiffness matrix for each wall of the box beam in the global coordinate system.

1.3 Solution

First, we calculate the orthotropic stiffness matrix for uni-directional fibers.

```
In [5]: import numpy as np
        E1, E2, E3, G12, G13, G23, v12, v13, v23 = 20e6, 1.4e6, 2.5e6, 0.8e6,2.0e6,
        S = np.array([[1/E1,-v12/E1,-v13/E1,0,0,0],
                      [-v12/E1,1/E2,-v23/E2,0,0,0],
```

```
                  [-v13/E1,-v23/E2,1/E3,0,0,0],
                  [0,0,0,1/G23,0,0],
                  [0,0,0,0,1/G13,0],
                  [0,0,0,0,0,1/G12]])
C = np.linalg.inv(S)
np.round(C*1e-6,decimals=2) #in Mpsi
```

```
Out[5]: array([[ 20.41,   0.54,   1.01,   0.  ,   0.  ,   0.  ],
               [  0.54,   1.59,   0.73,   0.  ,   0.  ,   0.  ],
               [  1.01,   0.73,   2.86,   0.  ,   0.  ,   0.  ],
               [  0.  ,   0.  ,   0.  ,   1.5 ,   0.  ,   0.  ],
               [  0.  ,   0.  ,   0.  ,   0.  ,   2.  ,   0.  ],
               [  0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.8 ]])
```

Now we need to calculate the rotations for each wall. First I will define some common rotation functions to build R_σ consistently for each wall.

```
In [17]: def qij_x(theta):
             """
             rotation tensor about x-axis by some theta
             input: theta (angle in radians)
             output: qij (3x3 rotation tensor)
             """
             qij = np.array([[1,0,0],
                             [0,np.cos(theta),np.sin(theta)],
                             [0,-np.sin(theta),np.cos(theta)]])
             return qij

         def qij_y(theta):
             """
             rotation tensor abo|ut y-axis by some theta
             input: theta (angle in radians)
             output: qij (3x3 rotation tensor)
             """
             qij = np.array([[np.cos(theta), 0, -np.sin(theta)],
                             [0,1,0],
                             [np.sin(theta),0,np.cos(theta)]])
             return qij

         def qij_z(theta):
             """
             rotation tensor about z-axis by some theta
             input: theta (angle in radians)
             output: qij (3x3 rotation tensor)
             """
             qij = np.array([[np.cos(theta),np.sin(theta),0],
                             [-np.sin(theta),np.cos(theta),0],
                             [0,0,1]])
```

```
        return qij

    def R_sigma(q):
        """
        rotation matrix for engineering notation given some rotation tensor, c
        note: uses convention for sigma = [s11, s22, s33, s23, s13, s12]
        input: q (3x3 rotation tensor)
        output: R_s (6x6 rotation matrix)
        """
        R_s = np.array([[q[0,0]**2,q[0,1]**2,q[0,2]**2,2*q[0,1]*q[0,2],2*q[0,(
                         [q[1,0]**2,q[1,1]**2,q[1,2]**2,2*q[1,1]*q[1,2],2*q[
                         [q[2,0]**2,q[2,1]**2,q[2,2]**2,2*q[2,1]*q[2,2],2*q[
                         [q[1,0]*q[2,0], q[1,1]*q[2,1], q[1,2]*q[2,2],q[1,2]
                         [q[0,0]*q[2,0], q[0,1]*q[2,1], q[0,2]*q[2,2],q[0,2]
                         [q[0,0]*q[1,0], q[0,1]*q[1,1], q[0,2]*q[1,2],q[0,2]
        return R_s
```

We will start by considering the face that intersects the positive x_2 axis. Since the lamina is orthotropic, there are multiple local coordinate systems that would give the same properties.

Recall that, as defined in Problem 1, we have $e_i = \beta_{ij} a_j$, so we can use β_{ij} from Problem 1 to directly relate the local coordinate system (fiber direction with the fibers pointing along x_1' and the outward normal being x_3') to the global coordinate system.

We can use the β defined in Problem 1 by first rotating about x_1 until x_3 is the normal, then rotating by $45°$ about the normal (x_3').

Starting at the right wall, we have $\theta_1 = 270$ (or -90) and $\theta_3 = 45$ (with $\theta_2 = 0$).

We can check this rotation by multiplying β by (1,0,0), (0,1,0) and (0,0,1) (the unit vectors in the prime coordinates), this should give back global coordinates of (1,0,-1)/$\sqrt{2}$, (-1,0,-1)/$\sqrt{2}$, and (0,1,0) respectively.

```
In [28]: beta_rightwall = beta.subs([(t1,-sm.pi/2),(t2,0),(t3,sm.pi/4)])
         #beta_rightwall*sm.Matrix([0,0,1])
```

Now we need to be careful about the R_σ as defined. The convention used for this R_σ is that R_σ is a function of Q_{ij}, and gives the relationship

$$C' = R_\sigma C R_\sigma^T$$

However, we already know C' and desire to find C in the global reference. We can acheive this by either re-writing our equation, or calculating R_σ with $Q^{-1} = Q^T = \beta$

```
In [70]: #convert beta to numeric python library
         B = np.array(beta_rightwall.tolist()).astype(np.float64)
         R = R_sigma(B)
         print np.round(np.dot(R,np.dot(C,R.T))*1e-6,decimals=2)

[[ 6.57  0.87  4.97  0.   -4.7   0.  ]
 [ 0.87  2.86  0.87  0.   -0.14  0.  ]
 [ 4.97  0.87  6.57  0.   -4.7   0.  ]
 [ 0.    0.    0.    1.75  0.   -0.25]
```

```
 [-4.7   -0.14 -4.7    0.     5.23   0.   ]
 [ 0.     0.     0.    -0.25   0.     1.75]]
```

We can now look at the top wall, where x_3 is already normal, so we simply set $\theta_3 = 45°$ with $\theta_1 = \theta_2 = 0$

```
In [71]: beta_topwall = beta.subs([(t1,0),(t2,0),(t3,sm.pi/4)])
         #print beta_topwall*sm.Matrix([0,0,1]) #check rotations
         B = np.array(beta_topwall.tolist()).astype(np.float64)
         R = R_sigma(B)
         print np.round(np.dot(R,np.dot(C,R.T))*1e-6,decimals=2)
```

```
[[ 6.57   4.97   0.87   0.     0.     4.7 ]
 [ 4.97   6.57   0.87   0.     0.     4.7 ]
 [ 0.87   0.87   2.86   0.     0.     0.14]
 [ 0.     0.     0.     1.75   0.25   0.  ]
 [ 0.     0.     0.     0.25   1.75   0.  ]
 [ 4.7    4.7    0.14   0.     0.     5.23]]
```

And for the left wall we set $\theta_1 = 90$ with $\theta_2 = 0$ and $\theta_3 = 45$

```
In [72]: beta_leftwall = beta.subs([(t1,sm.pi/2),(t2,0),(t3,sm.pi/4)])
         #beta_leftwall*sm.Matrix([0,1,0]) #check rotations
         B = np.array(beta_leftwall.tolist()).astype(np.float64)
         R = R_sigma(B)
         C_left = np.dot(R,np.dot(C,R.T))#save for problem 3
         print np.round(C_left*1e-6,decimals=2)
```

```
[[ 6.57   0.87   4.97   0.     4.7    0.  ]
 [ 0.87   2.86   0.87   0.     0.14   0.  ]
 [ 4.97   0.87   6.57   0.     4.7    0.  ]
 [ 0.     0.     0.     1.75   0.     0.25]
 [ 4.7    0.14   4.7    0.     5.23   0.  ]
 [ 0.     0.     0.     0.25   0.     1.75]]
```

Finally for the bottom wall we have $\theta_1 = 180$, $\theta_2 = 0$ and $\theta_3 = 45$

```
In [73]: beta_bottomwall = beta.subs([(t1,sm.pi),(t2,0),(t3,sm.pi/4)])
         #beta_bottomwall*sm.Matrix([0,1,0]) #check rotations
         B = np.array(beta_bottomwall.tolist()).astype(np.float64)
         R = R_sigma(B)
         print np.round(np.dot(R,np.dot(C,R.T))*1e-6,decimals=2)
```

```
[[ 6.57   4.97   0.87   0.     0.    -4.7 ]
 [ 4.97   6.57   0.87   0.     0.    -4.7 ]
 [ 0.87   0.87   2.86   0.     0.    -0.14]
 [ 0.     0.     0.     1.75  -0.25   0.  ]
 [ 0.     0.     0.    -0.25   1.75   0.  ]
 [-4.7   -4.7   -0.14   0.     0.     5.23]]
```

2 Problem 3

To find R_σ and R_ϵ we start by multiplying out the transformation relationship.

In [74]:
```
#define symbols for stress tensor
s1, s2, s3, s4, s5, s6 = sm.symbols('\sigma_1 \sigma_2 \sigma_3 \sigma_4 \
s = sm.Matrix([[s1, s6, s5],
              [s6,s2,s4],
               [s5,s4,s3]]) #symmetric stress tensor in engineering notati
Q = sm.Matrix(3, 3, lambda i,j:sm.symbols('Q_{%d%d}' % (i+1,j+1))) #symbo
sp = Q*s*Q.T
sp = sp.expand() #multiply out terms
#write matrix in vector form in correct order
spe = sm.Matrix([[sp[0,0]],
                [sp[0,1]],
                [sp[1,1]],
                [sp[0,2]],
                [sp[1,2]],
                [sp[2,2]]]) #reshape to vector for engineering notation
spe #notice that terms are not necessarily sorted, nor are they combined
```

Out[74]:

$$\left[\begin{matrix}Q_{11}^{2}\sigma_{1} + 2Q_{11}Q_{12}\sigma_{6} + 2Q_{11}Q_{13}\sigma_{5} + Q_{12}^{2}\sigma_{2} + 2Q_{12}Q_{13}\sigma_{4} + Q_{13}^{2}\sigma_{3}\\Q_{11}Q_{21}\sigma_{1} + Q_{11}Q_{22}\sigma_{6} + Q_{11}Q_{23}\sigma_{5} + Q_{12}Q_{21}\sigma_{6} + Q_{12}Q_{22}\sigma_{2} + Q_{12}Q_{23}\sigma_{4} + Q_{13}Q_{21}\sigma_{5} + Q_{13}Q_{22}\sigma_{4} + Q_{13}Q_{23}\sigma_{3}\\Q_{21}^{2}\sigma_{1} + 2Q_{21}Q_{22}\sigma_{6} + 2Q_{21}Q_{23}\sigma_{5} + Q_{22}^{2}\sigma_{2} + 2Q_{22}Q_{23}\sigma_{4} + Q_{23}^{2}\sigma_{3}\\Q_{11}Q_{31}\sigma_{1} + Q_{11}Q_{32}\sigma_{6} + Q_{11}Q_{33}\sigma_{5} + Q_{12}Q_{31}\sigma_{6} + Q_{12}Q_{32}\sigma_{2} + Q_{12}Q_{33}\sigma_{4} + Q_{13}Q_{31}\sigma_{5} + Q_{13}Q_{32}\sigma_{4} + Q_{13}Q_{33}\sigma_{3}\\Q_{21}Q_{31}\sigma_{1} + Q_{21}Q_{32}\sigma_{6} + Q_{21}Q_{33}\sigma_{5} + Q_{22}Q_{31}\sigma_{6} + Q_{22}Q_{32}\sigma_{2} + Q_{22}Q_{33}\sigma_{4} + Q_{23}Q_{31}\sigma_{5} + Q_{23}Q_{32}\sigma_{4} + Q_{23}Q_{33}\sigma_{3}\\Q_{31}^{2}\sigma_{1} + 2Q_{31}Q_{32}\sigma_{6} + 2Q_{31}Q_{33}\sigma_{5} + Q_{32}^{2}\sigma_{2} + 2Q_{32}Q_{33}\sigma_{4} + Q_{33}^{2}\sigma_{3}\end{matrix}\right]$$

Now we need to write this as a matrix equation by factoring out like terms.

In [75]:
```
#here we will sort and combine terms
R_s = sm.zeros(6) #first fill with zeros
cols = [s1,s6,s2,s5,s4,s3] #define term order
#loop through all values of R_s
for i in range(6):
    row = spe[i] #find row of matrix to work with
    for j in range(6):
        col = cols[j]
        temp = sm.collect(row,col,evaluate=False) #collect terms in row wh
        R_s[i,j] = temp[col] #factor out term from col, and insert it into
R_s
```

Out[75]:

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

We can find the transformation for engineering strain by expanding the transformation for strain, but replacing the shear strain terms ϵ_{12} with engineering strain divided by 2 $\gamma_{12}/2$. We can then multiply both sides of the shear strain equations by two to find the transformation in terms of the engineering strain.

```
In [76]: #define strain tensor symbols
         e1, e2, e3, e4, e5, e6 = sm.symbols('\epsilon_1 \epsilon_2 \epsilon_3 \eps
         e = sm.Matrix([[e1, e6, e5],
                        [e6,e2,e4],
                          [e5,e4,e3]])
         #engineering strain symbols
         g12, g13, g23 = sm.symbols('\gamma_{12} \gamma_{13} \gamma_{23}')
         #substitute engineering strain
         e=e.subs([(e4,g23/2),(e5,g13/2),(e6,g12/2)])
         e
```

Out[76]:

$$\begin{bmatrix} \epsilon_1 & \frac{\gamma_{12}}{2} & \frac{\gamma_{13}}{2} \\ \frac{\gamma_{12}}{2} & \epsilon_2 & \frac{\gamma_{23}}{2} \\ \frac{\gamma_{13}}{2} & \frac{\gamma_{23}}{2} & \epsilon_3 \end{bmatrix}$$

```
In [77]: #transformation equation
         ep = Q*e*Q.T
         ep = ep.expand()
         ep
```

Out[77]:

$$\Bigg[\begin{matrix} Q_{11}^2\epsilon_1 + Q_{11}Q_{12}\gamma_{12} + Q_{11}Q_{13}\gamma_{13} + Q_{12}^2\epsilon_2 + Q_{12}Q_{13}\gamma_{23} + Q_{13}^2\epsilon_3 & Q \\ Q_{11}Q_{21}\epsilon_1 + \frac{Q_{11}Q_{22}}{2}\gamma_{12} + \frac{Q_{11}Q_{23}}{2}\gamma_{13} + \frac{Q_{12}Q_{21}}{2}\gamma_{12} + Q_{12}Q_{22}\epsilon_2 + \frac{Q_{12}Q_{23}}{2}\gamma_{23} + \frac{Q_{13}Q_{21}}{2}\gamma_{13} + \frac{Q_{13}Q_{22}}{2}\gamma_{23} + Q_{13}Q_{23}\epsilon_3 & Q \\ Q_{11}Q_{31}\epsilon_1 + \frac{Q_{11}Q_{32}}{2}\gamma_{12} + \frac{Q_{11}Q_{33}}{2}\gamma_{13} + \frac{Q_{12}Q_{31}}{2}\gamma_{12} + Q_{12}Q_{32}\epsilon_2 + \frac{Q_{12}Q_{33}}{2}\gamma_{23} + \frac{Q_{13}Q_{31}}{2}\gamma_{13} + \frac{Q_{13}Q_{32}}{2}\gamma_{23} + Q_{13}Q_{33}\epsilon_3 & Q \end{matrix}$$

Notice that the shear equations show the transformation for ϵ'_{12}, or $\gamma'_{12}/2$. Since we want a transformation for γ'_{12}, we will multiply both sides of those equations by 2.

```
In [78]: #write matrix in vector form in correct order
         epe = sm.Matrix([[ep[0,0]],
                          [2*ep[0,1]],
                          [ep[1,1]],
```

```
                    [2*ep[0,2]],
                    [2*ep[1,2]],
                    [ep[2,2]]]) #reshape to vector for engineering notation
epe #notice that terms are not necessarily sorted, nor are they combined
```

Out[78]:

$$\left[\begin{array}{c} Q_{11}^2\epsilon_1 + Q_{11}Q_{12}\gamma_{12} + Q_{11}Q_{13}\gamma_{13} + Q_{12}^2\epsilon_2 + Q_{12}Q_{13}\gamma_{23} + Q_{13}^2\epsilon_3 \\ 2Q_{11}Q_{21}\epsilon_1 + Q_{11}Q_{22}\gamma_{12} + Q_{11}Q_{23}\gamma_{13} + Q_{12}Q_{21}\gamma_{12} + 2Q_{12}Q_{22}\epsilon_2 + Q_{12}Q_{23}\gamma_{23} + Q_{13}Q_{21}\gamma_{13} + Q_{13}Q_{22}\gamma_{23} + 2Q_{13}Q \\ Q_{21}^2\epsilon_1 + Q_{21}Q_{22}\gamma_{12} + Q_{21}Q_{23}\gamma_{13} + Q_{22}^2\epsilon_2 + Q_{22}Q_{23}\gamma_{23} + Q_{23}^2\epsilon_3 \\ 2Q_{11}Q_{31}\epsilon_1 + Q_{11}Q_{32}\gamma_{12} + Q_{11}Q_{33}\gamma_{13} + Q_{12}Q_{31}\gamma_{12} + 2Q_{12}Q_{32}\epsilon_2 + Q_{12}Q_{33}\gamma_{23} + Q_{13}Q_{31}\gamma_{13} + Q_{13}Q_{32}\gamma_{23} + 2Q_{13}Q \\ 2Q_{21}Q_{31}\epsilon_1 + Q_{21}Q_{32}\gamma_{12} + Q_{21}Q_{33}\gamma_{13} + Q_{22}Q_{31}\gamma_{12} + 2Q_{22}Q_{32}\epsilon_2 + Q_{22}Q_{33}\gamma_{23} + Q_{23}Q_{31}\gamma_{13} + Q_{23}Q_{32}\gamma_{23} + 2Q_{23}Q \\ Q_{31}^2\epsilon_1 + Q_{31}Q_{32}\gamma_{12} + Q_{31}Q_{33}\gamma_{13} + Q_{32}^2\epsilon_2 + Q_{32}Q_{33}\gamma_{23} + Q_{33}^2\epsilon_3 \end{array}\right.$$

Now we can sort terms and factor into a matrix equation

```
In [79]: R_e = sm.zeros(6) #first fill with zeros
         cols = [e1,g12,e2,g13,g23,e3] #define term order
         #loop through all values of R_s
         for i in range(6):
             row = epe[i] #find row of matrix to work with
             for j in range(6):
                 col = cols[j]
                 temp = sm.collect(row,col,evaluate=False) #collect terms in row wh
                 R_e[i,j] = temp[col] #factor out term from col, and insert it into
         R_e
```

Out[79]:

$$\left[\begin{array}{cccccc} Q_{11}^2 & Q_{11}Q_{12} & Q_{12}^2 & Q_{11}Q_{13} & Q_{12}Q_{13} & Q_{13}^2 \\ 2Q_{11}Q_{21} & Q_{11}Q_{22} + Q_{12}Q_{21} & 2Q_{12}Q_{22} & Q_{11}Q_{23} + Q_{13}Q_{21} & Q_{12}Q_{23} + Q_{13}Q_{22} & 2Q_{13}Q_{23} \\ Q_{21}^2 & Q_{21}Q_{22} & Q_{22}^2 & Q_{21}Q_{23} & Q_{22}Q_{23} & Q_{23}^2 \\ 2Q_{11}Q_{31} & Q_{11}Q_{32} + Q_{12}Q_{31} & 2Q_{12}Q_{32} & Q_{11}Q_{33} + Q_{13}Q_{31} & Q_{12}Q_{33} + Q_{13}Q_{32} & 2Q_{13}Q_{33} \\ 2Q_{21}Q_{31} & Q_{21}Q_{32} + Q_{22}Q_{31} & 2Q_{22}Q_{32} & Q_{21}Q_{33} + Q_{23}Q_{31} & Q_{22}Q_{33} + Q_{23}Q_{32} & 2Q_{23}Q_{33} \\ Q_{31}^2 & Q_{31}Q_{32} & Q_{32}^2 & Q_{31}Q_{33} & Q_{32}Q_{33} & Q_{33}^2 \end{array}\right]$$

To verify that $(R_\epsilon)^{-1} = R_\sigma^T$ for $\theta_1 = 20$, $\theta_2 = 45$ and $\theta_3 = 60$ we can substitute into the beta we found previously, calculate $Q = \beta^T$, then substitute values for both R_ϵ and R_σ.

```
In [80]: b3 = beta.subs([(t1,sm.pi/9.),(t2,sm.pi/4),(t3,sm.pi/3)])
         q = np.array(b3.tolist()).astype(np.float64)
         q = q.T
         #add new matrix to store values for numeric R_e and R_s
         R_s_vals = R_s
         R_e_vals = R_e
         #substitute into R_s and R_e
         for i in range(3):
             for j in range(3):
```

```
            R_e_vals = R_e_vals.subs(Q[i,j],q[i,j])
            R_s_vals = R_s_vals.subs(Q[i,j],q[i,j])
        R_s_vals = np.array(R_s_vals.tolist()).astype(np.float64)
        R_e_vals = np.array(R_e_vals.tolist()).astype(np.float64)
        np.round(np.linalg.inv(R_e_vals)-R_s_vals.T,decimals=2)
```

```
Out[80]: array([[ 0.,  0.,  0.,  0.,  0.,  0.],
                [ 0.,  0.,  0.,  0.,  0.,  0.],
                [ 0.,  0.,  0.,  0.,  0.,  0.],
                [ 0.,  0.,  0.,  0.,  0.,  0.],
                [ 0.,  0.,  0.,  0.,  0.,  0.],
                [ 0.,  0.,  0.,  0.,  0.,  0.]])
```

To transform the stiffness in this new notation, we first need to build the uni-directional stiffness matrix in this notation

```
In [81]: E1, E2, E3, G12, G13, G23, v12, v13, v23 = 20e6, 1.4e6, 2.5e6, 0.8e6,2.0e6
         S_new = np.array([[1/E1,0,-v12/E1,0,0,-v13/E1],
                           [0,1/G12,0,0,0,0],
                           [-v12/E1,0,1/E2,0,0,-v23/E2],
                           [0,0,0,1/G13,0,0],
                           [0,0,0,0,1/G23,0],
                           [-v13/E1,0,-v23/E2,0,0,1/E3]])
         C_new = np.linalg.inv(S_new)
         np.round(C_new*1e-6,decimals=2) #in Mpsi
```

```
Out[81]: array([[ 20.41,   0.  ,   0.54,   0.  ,   0.  ,   1.01],
                [  0.  ,   0.8 ,   0.  ,   0.  ,   0.  ,   0.  ],
                [  0.54,   0.  ,   1.59,   0.  ,   0.  ,   0.73],
                [  0.  ,   0.  ,   0.  ,   2.  ,   0.  ,   0.  ],
                [  0.  ,   0.  ,   0.  ,   0.  ,   1.5 ,   0.  ],
                [  1.01,   0.  ,   0.73,   0.  ,   0.  ,   2.86]])
```

Now we perform the rotation using the β and Q we assembled for the left wall

```
In [82]: b3 = beta.subs([(t1,sm.pi/2),(t2,0),(t3,sm.pi/4)])
         q = np.array(b3.tolist()).astype(np.float64)
         #add new matrix to store values for numeric R_e and R_s
         R_s_vals = R_s
         R_e_vals = R_e
         #substitute into R_s and R_e
         for i in range(3):
             for j in range(3):
                 R_s_vals = R_s_vals.subs(Q[i,j],q[i,j])
         R_s_vals = np.array(R_s_vals.tolist()).astype(np.float64)

         C_left_new = np.dot(R_s_vals,np.dot(C_new,R_s_vals.T))
         print np.round(C_left_new*1e-6,decimals=2)
```

```
[[ 6.57  0.    0.87  4.7   0.    4.97]
 [ 0.    1.75  0.    0.    0.25  0.  ]
 [ 0.87  0.    2.86  0.14  0.    0.87]
 [ 4.7   0.    0.14  5.23  0.    4.7 ]
 [ 0.    0.25  0.    0.    1.75  0.  ]
 [ 4.97  0.    0.87  4.7   0.    6.57]]
```

Now we can confirm that $C' = BCB^T$

```
In [83]: B = np.array([[1,0,0,0,0,0],
                       [0,0,0,0,0,1],
                       [0,1,0,0,0,0],
                       [0,0,0,0,1,0],
                       [0,0,0,1,0,0],
                       [0,0,1,0,0,0]])
         np.round(np.dot(B,np.dot(C_left,B.T))*1e-6,decimals=2)
```

```
Out[83]: array([[ 6.57,  0.  ,  0.87,  4.7 ,  0.  ,  4.97],
                [ 0.  ,  1.75,  0.  ,  0.  ,  0.25,  0.  ],
                [ 0.87,  0.  ,  2.86,  0.14,  0.  ,  0.87],
                [ 4.7 ,  0.  ,  0.14,  5.23,  0.  ,  4.7 ],
                [ 0.  ,  0.25,  0.  ,  0.  ,  1.75,  0.  ],
                [ 4.97,  0.  ,  0.87,  4.7 ,  0.  ,  6.57]])
```