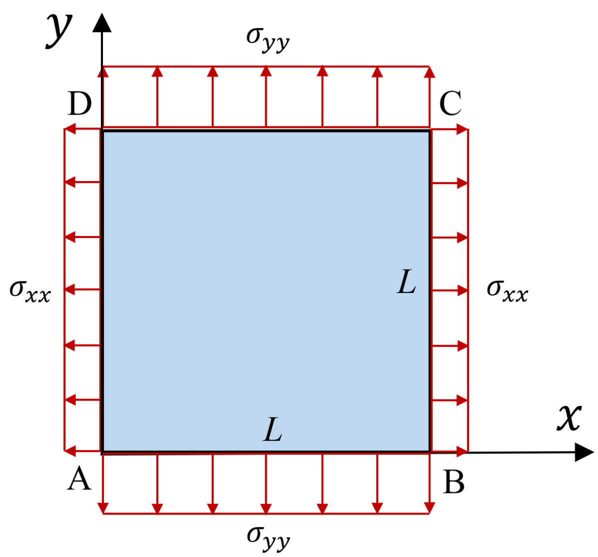
# Homeworks for SM-I course

## Homework 4: Understanding stresses, strains and constitutive relations

Consider a thin square wing spar made of aluminum alloy that can be treated as a homogeneous isotropic material with Young's modulus $E$ and Poisson's ratio $\nu$. It is subjected to uniform stress loading along all edges, as shown in the figure below. Because it is thin, and no force is applied in the $z$-direction, it can be treated a plane stress problem in x-y plane. The data are given as $E = 80GPa$, $\nu = 1/3$,$\sigma_{yy} = 150MPa$, and $L = 100mm$.




1. Derive the formulas for computing the strain components.

2. Determine the stress load $\sigma_{xx}$ (in terms of $\sigma_{yy}$), under the condition that wing spar width remains unchanged. Compute the numerical value for given data.

3. Determine the strain and elongation in the $y$-direction, under the same condition given in 2). Compute the numerical value for given data.

4. Using the coordinate transformation rule, determine the elongation of the diagonal AC, under the same condition given in 2).

5. Determine the area change of the wing spar, under the same condition given in 2).

In [ ]:

```
# Place curse in this cell, and press Ctrl+Enter to import dependences.
import sys                          # for accessing the computer system
sys.path.append('../grbin/')  # Change to the directory in your system

from commonImports import *       # Import dependences from '../grbin/'
import grcodes as gr                  # Import the module of the author
#importlib.reload(gr)             # When grcodes is modified, reload it

from continuum_mechanics import vector
init_printing(use_unicode=True)       # For latex-like quality printing
np.set_printoptions(precision=4,suppress=True)  # Digits in print-outs
```

1.

$\epsilon_{xx} = \sigma_{xx}/E$
$\epsilon_{yy} = \sigma_{yy}/E$
$\epsilon_{yy} = -v\epsilon_{xx}$

Equations:
$\sigma_{yy}$, $E$, and $v$ are given

$\epsilon_{yy} = \sigma_{yy}/E$
$\epsilon_{xx} = -v(\sigma_{yy}/E)$

2.

$\epsilon_{xx} = \sigma_{xx}/E$
$\epsilon_{xx} = -v(\sigma_{yy}/E)$
$\sigma_{xx}/E = -v(\sigma_{yy}/E)$
$\sigma_{xx} = -v\sigma_{yy}$

In [ ]:

```
v = 1/3
Syy = 150 # MPa

Sxx = -1*v*Syy

print("Stress in xx: %3.2f MPa" % (Sxx))
```

```
Stress in xx: -50.00 MPa
```

3.

$$\epsilon_{yy} = \frac{\Delta L}{L}$$

$$\epsilon_{yy} = \frac{1}{E}\sigma_{yy}$$

$$\frac{\Delta L}{L} = \frac{1}{E}\sigma_{yy}$$

$$\Delta L = \frac{L}{E}\sigma_{yy}$$

In [ ]:
```
E = 80 * (10^3) # MPa

Eyy = (1/E)*Syy

print("Strain in yy: %3.2f" % (Eyy))

L = 100 # mm

deltaL = (L/E)*Syy

print("Deformation in yy: %3.2f mm" % (deltaL))
```

```
Strain in yy: 0.21
Deformation in yy: 20.83 mm
```

4.

In [ ]:
```
def apply_symmetry(C4, key = "all", tol=1.e-2):
    if key == "all" or key == "ij":
        for k in range(3):
            for l in range(3):
                for i in range(3):
                    for j in range(i+1,3):
                        if abs(C4[j,i,k,l]) <= tol:
                            C4[j,i,k,l]=C4[i,j,k,l]

    if key == "all" or key == "kl":
        for k in range(3):
            for l in range(k+1,3):
                for i in range(3):
                    for j in range(3):
                        if abs(C4[i,j,l,k]) <= tol:
                            C4[i,j,l,k]=C4[i,j,k,l]

    if key == "all" or key == "ijkl":
        for k in range(3):
            for l in range(3):
                for i in range(k+1,3):
                    for j in range(l+1,3):
                        if abs(C4[i,j,k,l]) <= tol:
                            C4[i,j,k,l]=C4[k,l,i,j]
    return C4
```

In [ ]:
```
def C2toC4(C2):
    '''To convert C(6,6) matrix (the Voigt notation) to
       4th tensor C(3,3,3,3).'''
    C4 = np.zeros((3,3,3,3))                        #Initialization

    # Pass over all C(6,6) to parts of C(3,3,3,3)
    C4[0,0,0,0],C4[0,0,1,1],C4[0,0,2,2] = C2[0,0],C2[0,1],C2[0,2]
    C4[0,0,1,2],C4[0,0,0,2],C4[0,0,0,1] = C2[0,3],C2[0,4],C2[0,5]

    C4[1,1,0,0],C4[1,1,1,1],C4[1,1,2,2] = C2[1,0],C2[1,1],C2[1,2]
    C4[1,1,1,2],C4[1,1,0,2],C4[1,1,0,1] = C2[1,3],C2[1,4],C2[1,5]

    C4[2,2,0,0],C4[2,2,1,1],C4[2,2,2,2] = C2[2,0],C2[2,1],C2[2,2]
    C4[2,2,1,2],C4[2,2,0,2],C4[2,2,0,1] = C2[2,3],C2[2,4],C2[2,5]

    C4[1,2,0,0],C4[1,2,1,1],C4[1,2,2,2] = C2[3,0],C2[3,1],C2[3,2]
    C4[1,2,1,2],C4[1,2,0,2],C4[1,2,0,1] = C2[3,3],C2[3,4],C2[3,5]

    C4[0,2,0,0],C4[0,2,1,1],C4[0,2,2,2] = C2[4,0],C2[4,1],C2[4,2]
    C4[0,2,1,2],C4[0,2,0,2],C4[0,2,0,1] = C2[4,3],C2[4,4],C2[4,5]

    C4[0,1,0,0],C4[0,1,1,1],C4[0,1,2,2] = C2[5,0],C2[5,1],C2[5,2]
    C4[0,1,1,2],C4[0,1,0,2],C4[0,1,0,1] = C2[5,3],C2[5,4],C2[5,5]

    # Imporse (minor) symmetric conditions
    apply_symmetry(C4, key = "all", tol=1.e-4)

    return C4
```

In [ ]:
```
def C4toC2(C4):
    '''To convert 4th tensor C(3,3,3,3) to C(6,6) matrix
       (the Voigt notation).'''
    C2 = np.zeros((6,6))
    C2[0,0],C2[0,1],C2[0,2]=C4[0,0,0,0],C4[0,0,1,1],C4[0,0,2,2]
    C2[0,3],C2[0,4],C2[0,5]=C4[0,0,1,2],C4[0,0,0,2],C4[0,0,0,1]

    C2[1,0],C2[1,1],C2[1,2]=C4[1,1,0,0],C4[1,1,1,1],C4[1,1,2,2]
    C2[1,3],C2[1,4],C2[1,5]=C4[1,1,1,2],C4[1,1,0,2],C4[1,1,0,1]

    C2[2,0],C2[2,1],C2[2,2]=C4[2,2,0,0],C4[2,2,1,1],C4[2,2,2,2]
    C2[2,3],C2[2,4],C2[2,5]=C4[2,2,1,2],C4[2,2,0,2],C4[2,2,0,1]

    C2[3,0],C2[3,1],C2[3,2]=C4[1,2,0,0],C4[1,2,1,1],C4[1,2,2,2]
    C2[3,3],C2[3,4],C2[3,5]=C4[1,2,1,2],C4[1,2,0,2],C4[1,2,0,1]

    C2[4,0],C2[4,1],C2[4,2]=C4[0,2,0,0],C4[0,2,1,1],C4[0,2,2,2]
    C2[4,3],C2[4,4],C2[4,5]=C4[0,2,1,2],C4[0,2,0,2],C4[0,2,0,1]

    C2[5,0],C2[5,1],C2[5,2]=C4[0,1,0,0],C4[0,1,1,1],C4[0,1,2,2]
    C2[5,3],C2[5,4],C2[5,5]=C4[0,1,1,2],C4[0,1,0,2],C4[0,1,0,1]

    return C2
```

In [ ]:
```python
def E_SnC3Dsp(E1, E2, E3, m12, m13, m23, G23, G13, G12):
    '''Compute the S and C matrix in Voigt notation for given Young's
    moduli and Poisson's ratios of orthotropic materials for 3D prolems.
    '''
    S = sp.zeros(6,6)                #initialization
    m21, m31, m32 = m12/E1*E2, m13/E1*E3, m23/E2*E3
    # compute the compliance matrix S
    S[0,0], S[1,1], S[2,2] = 1/E1, 1/E2, 1/E3
    S[0,1], S[0,2], S[1,2] = -m21/E2, -m31/E2, -m32/E3
    S[3,3], S[4,4], S[5,5] = 1/G23, 1/G13, 1/G12
    S[1,0], S[2,0], S[2,1] = S[0,1], S[0,2], S[1,2]

    # compute C matrix
    C = S.inv()

    return C, S
```

In [ ]:
```python
def transferM(theta, about = 'z'):
    '''Create a transformation matrix for coordinate transformation (numpy)\
    Input theta: rotation angle in degree \
          about: the axis of the rotation is about \
    Return: numpy array of transformation matrix of shape (3,3)'''
    from scipy.stats import ortho_group

    n = 3        # 3-dimensonal problem
    c, s = np.cos(np.deg2rad(theta)), np.sin(np.deg2rad(theta))
    #T = np.zeros((n,n))

    if about == 'z':
        # rotates about z by theta
        T = np.array([[ c, s, 0.],
                      [-s, c, 0.],
                      [0.,0., 1.]])
    elif about == 'y':
        # rotates about y by theta
        T = np.array([[ c, 0.,-s],
                      [0., 1.,0.],
                      [s, 0., c]])
    elif about == 'x':
        # rotates about x by theta
        T = np.array([[ 1.,0., 0.],
                      [ 0., c, s],
                      [ 0.,-s, c]])
    else: # randomly generated unitary matrix->transformation matrix, no theta
        T = ortho_group.rvs(dim=n)          # Generate a random matrix
        T[2,:] = np.cross(T[0,:], T[1,:])   # Enforce the righ-hand rule

    return T, about
```

In [ ]:
```python
G = 0.5*E/(1+v)

C0, S = E_SnC3Dsp(E,E,E,v,v,v,G,G,G)

C0
```

Out[ ]: $$\begin{bmatrix} 1080.0 & 540.0 & 540.0 & 0 & 0 & 0 \\ 540.0 & 1080.0 & 540.0 & 0 & 0 & 0 \\ 540.0 & 540.0 & 1080.0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 270.0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 270.0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 270.0 \end{bmatrix}$$

In [ ]:
```
T, _ = transferM(45, about = 'random')
```

In [ ]:
```
def Tensor4_transfer(T,C4):

    C4 = np.tensordot( T, C4, axes=([1],[0]))  # contract i
    C4 = np.tensordot( T, C4, axes=([1],[1]))  # contract j
    C4 = np.tensordot(C4,  T, axes=([3],[1]))  # contract l
    C4 = np.tensordot(C4,  T, axes=([2],[1]))  # contract k

    return C4
```

In [ ]:
```
C4 = C2toC4(C0)

Cp = Tensor4_transfer(T,C4)

C2 = C4toC2(Cp)

C2
```

Out[ ]:
```
array([[1080.,  540.,  540.,    0.,    0.,    0.],
       [ 540., 1080.,  540.,    0.,    0.,    0.],
       [ 540.,  540., 1080.,    0.,    0.,    0.],
       [   0.,    0.,    0.,  270.,    0.,    0.],
       [   0.,    0.,    0.,    0.,  270.,    0.],
       [   0.,    0.,    0.,    0.,    0.,  270.]])
```

In [ ]:
```
# Validation

originalAC = np.sqrt((L**2)+(L**2))

newAC = np.sqrt((L**2)+((L+deltaL)**2))

print("The change in length AC is %3.2f mm" % (newAC-originalAC))
```

```
The change in length AC is 15.42 mm
```

5.

In [ ]:
```
areachange = L*deltaL

print("The change in area is %3.2f mm^2" % (areachange))
```

```
The change in area is 2083.33 mm^2
```