Homework 2 Solutions

March 1, 2017

0.1 Problem 1

In this problem we plot the effect of volume fraction for a perfectly aligned short-fiber composite and compare between three models, Eshelby, Mori-Tanaka, and Halpin-Tsai.

For both the Eshelby and Mori-Tanaka models we will need to calculate the Eshelby Tensor.

```
In [1]: import numpy as np

        #material properties
        Em = 10e9 #matrix stiffness
        nu_m = 0.40 #matrix poisson's ratio

        Ef = 200e9 #fiber stiffness
        nu_f = 0.2 #fiber poisson's ratio

        s = 50. #fiber aspect ratio

        #eshelby tensor
        I1 = (2.*s/(s**2.-1.)**1.5)*(s*(s**2.-1.)**.5-np.arccosh(s))
        Q = 3./(8.*(1-nu_m))
        R = (1.-2*nu_m)/(8*(1-nu_m))
        T = Q*(4.-3.*I1)/(3.*(s**2.-1.))
        I3 = 4.-2.*I1
        S = np.zeros((6,6))
        S[0,0] = Q + R*I1 + 0.75*T
        S[1,1] = S[0,0]
        S[2,2] = 4./3.*Q + R*I3 + 2*s**2*T
        S[0,1] = Q/3 - R*I1 + 4.*T/3.
        S[1,0] = S[0,1]
        S[0,2] = -R*I1 - s**2*T
        S[1,2] = S[0,2]
        S[2,0] = -R*I3 - T
        S[2,1] = S[2,0]
        S[5,5] = Q/3. + R*I1 + T/4.
        S[3,3] = 2.*R - I1*R/2. - (1.+s**2)*T/2.
        S[4,4] = S[3,3]
```

We also need to assemble our fiber and matrix properties into a stiffness tensor for each.

```
In [2]: #fiber stiffness tensor
        Cf = np.zeros((6,6))
        Cf[0:3,0:3] = nu_f*np.ones((3,3))
        Cf = Cf + (1-2*nu_f)*np.eye(6)
        Cf[3:,3:] = Cf[3:,3:]/2
        Cf = Ef/((1+nu_f)*(1-2*nu_f))*Cf

        #matrix stiffness tensor
        Cm = np.zeros((6,6))
        Cm[0:3,0:3] = nu_m*np.ones((3,3))
        Cm = Cm + (1-2*nu_m)*np.eye(6)
        Cm[3:,3:] = Cm[3:,3:]/2
        Cm = Em/((1+nu_m)*(1-2*nu_m))*Cm
```

Now we need to calculate the strain concentration tensors for both Eshelby and Mori-Tanaka methods. For Mori-Tanaka, we will have a different strain concentration tensor for each volume fraction, so we also define an array of volume fractions.

```
In [3]: vf = np.linspace(0,.5)

        #Eshelby strain concentration
        A_E = np.linalg.inv(np.eye(6)+np.dot(np.dot(S,np.linalg.inv(Cm)),(Cf-Cm)))

        #Mori-Tanaka strain concentration
        #initiate 6x6 matrix for each vf value
        A_mt = np.zeros((len(vf),6,6))
        i=0 #track index

        for v in vf: #loop through volume fraction values
            A_tot = A_E*v
            A_mt[i,:,:] = np.dot(A_E,np.linalg.inv(A_tot+(1-v)*np.eye(6)))
            i = i+1
```

Now we can calculate the stiffness tensor for both the Eshelby and Mori-Tanaka methods.

```
In [4]: #make array of stiffnesses and compliances
        C_E = np.zeros((len(vf),6,6))
        C_mt = np.zeros((len(vf),6,6))
        S_E = np.zeros((len(vf),6,6))
        S_mt = np.zeros((len(vf),6,6))

        #calculate stiffness at each volme fraction
        for i in range(len(vf)):
            C_E[i,:,:] = Cm + vf[i]*np.dot((Cf-Cm),A_E)
            C_mt[i,:,:] = Cm + vf[i]*np.dot((Cf-Cm),A_mt[i,:,:])
            S_E[i,:,:] = np.linalg.inv(C_E[i,:,:])
            S_mt[i,:,:] = np.linalg.inv(C_mt[i,:,:])
```

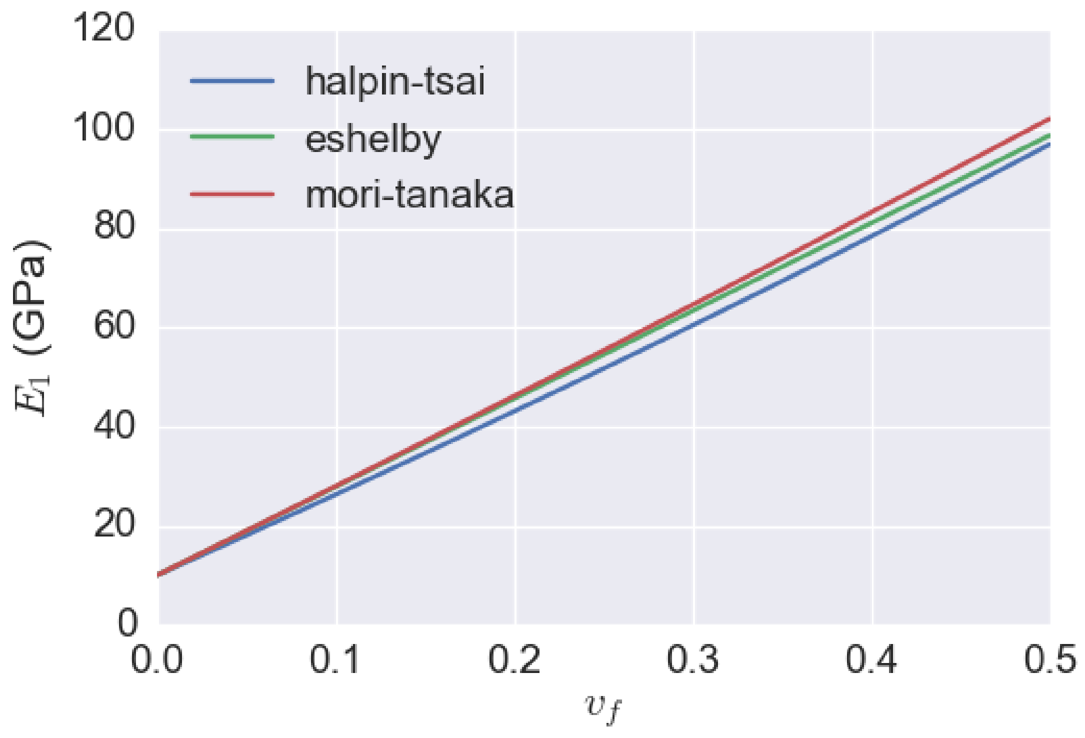
Now we also calcualte stiffness using the Halpin-Tsai method

```
In [5]: #axial stiffness
        zeta = 2.*s
        eta = (Ef/Em-1.)/(Ef/Em+zeta)
        E1_ht = Em*(1.+zeta*eta*vf)/(1.-eta*vf)

        #transverse stiffness
        zeta = 2.
        eta = (Ef/Em-1)/(Ef/Em+zeta)
        E2_ht = Em*(1.+zeta*eta*vf)/(1.-eta*vf)

        #shear stiffness
        Gf = Ef/(2.*(1.+nu_f)) #isotropic material shear stiffness
        Gm = Em/(2.*(1.+nu_m)) #isotropic material shear stiffness
        zeta = 1.
        eta = (Gf/Gm-1)/(Gf/Gm+zeta)
        G12_ht = Gm*(1.+zeta*eta*vf)/(1.-eta*vf)
```

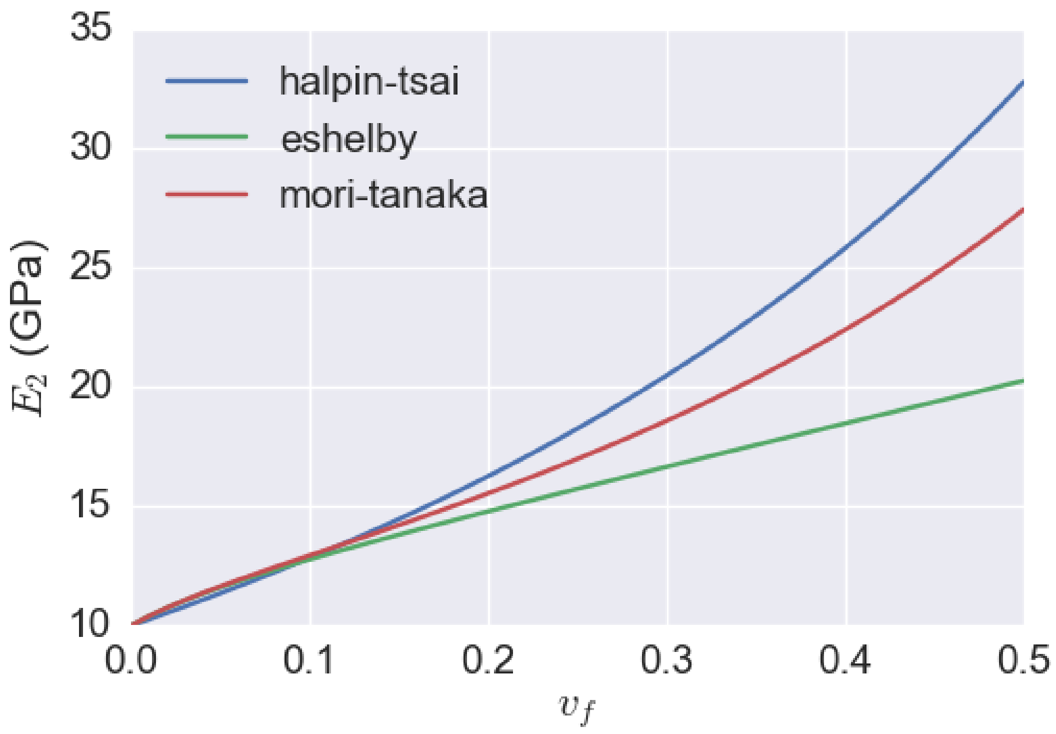
And we extract the appropriate stiffnesses for Eshelby and Mori-Tanaka to plot

```
In [6]: #Eshelby
        E1_E = 1./S_E[:,2,2] #fibers in 3 direction
        E2_E = 1./S_E[:,1,1]
        G12_E = 1./S_E[:,3,3]

        #Mori-Tanaka
        E1_mt = 1./S_mt[:,2,2] #fibers in 3 direction
        E2_mt = 1./S_mt[:,1,1]
        G12_mt = 1./S_mt[:,3,3]
```

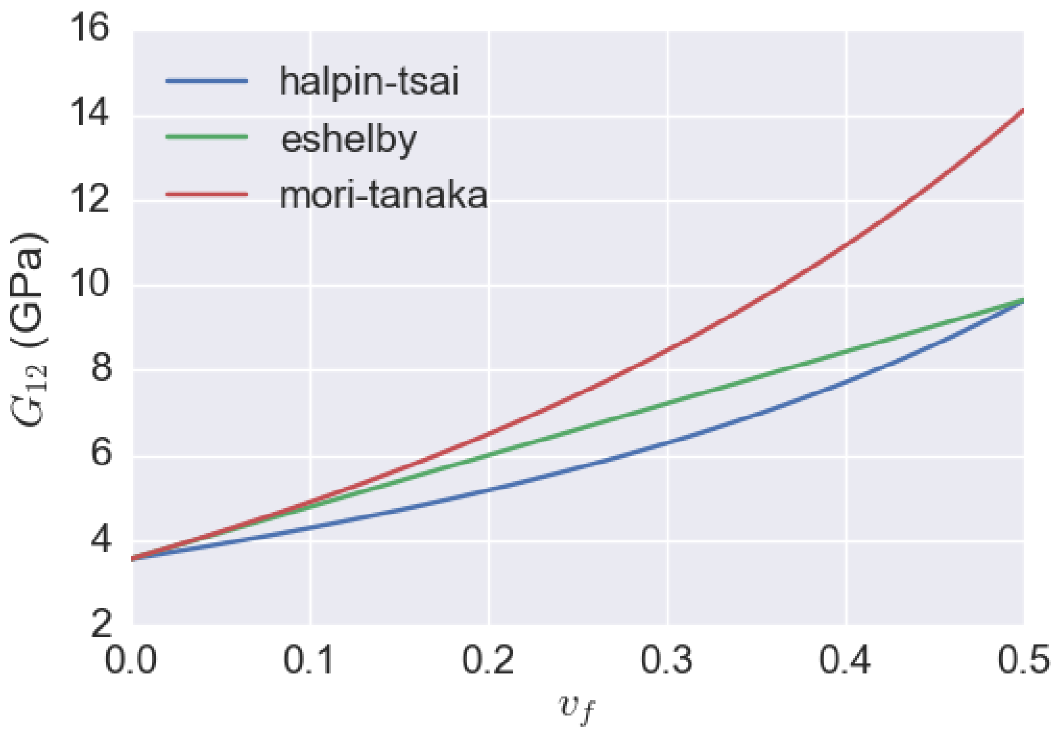
```
In [7]: import numpy as np
        from matplotlib import pyplot as plt
        #optional, for prettier default color scheme
        import seaborn as sb
        #make fonts bigger
        sb.set(font_scale=1.5)
        #only needed for "live notebooks" such as this demo
        %matplotlib inline

        plt.plot(vf,E1_ht*1e-9,label='halpin-tsai')
        plt.plot(vf,E1_E*1e-9,label='eshelby')
        plt.plot(vf,E1_mt*1e-9,label='mori-tanaka')
        plt.legend(loc='best')
        plt.xlabel('$v_f$')
        plt.ylabel('$E_1$ (GPa)')

Out[7]: <matplotlib.text.Text at 0xa2846d8>
```



```
In [8]: plt.plot(vf,E2_ht*1e-9,label='halpin-tsai')
        plt.plot(vf,E2_E*1e-9,label='eshelby')
        plt.plot(vf,E2_mt*1e-9,label='mori-tanaka')
        plt.legend(loc='best')
        plt.xlabel('$v_f$')
        plt.ylabel('$E_2$ (GPa)')

Out[8]: <matplotlib.text.Text at 0xb52f2e8>
```



```
In [9]: plt.plot(vf,G12_ht*1e-9,label='halpin-tsai')
        plt.plot(vf,G12_E*1e-9,label='eshelby') #correct for engineering/tensor str
        plt.plot(vf,G12_mt*1e-9,label='mori-tanaka')
        plt.legend(loc='best')
        plt.xlabel('$v_f$')
        plt.ylabel('$G_{12}$ (GPa)')

Out[9]: <matplotlib.text.Text at 0xb8365f8>
```



0.2 Problem 2

In this problem we use orientation averaging to find the stiffness of a randomly oriented composite. With orientation averaging, we only need to find the uni-directional stiffness, so for Eshelby and Mori-Tanaka we can use the stiffness we already calculated in problem one.

To use orientation averaging with only a second-order orientation tensor, we need to estimate the fourth-order tensor with some closure approximation. Here I will show the linear and quadratic closures, the hybrid would simply converge to the linear closure.

First I define a function for convenience which converts a fourth-order tensor into engineering notation, represented by a 6x6 matrix.

```
In [10]: def tens2eng(tens):
             """
             returns 4th-order tensor in engineering notation
             """
             A = np.zeros((6,6))
             #mapping
             code = {0:(0,0),1:(1,1),2:(2,2),3:(1,2),4:(0,2),5:(0,1)}
             for i in range(6):
                 for j in range(6):
                     A[i,j] = tens[code[i][0],code[i][1],code[j][0],code[j][1]]
             return A
```

Now I will define the orientation averaging procedure as a function

```
In [11]: def ornavg(C,A4,a2):
             B1 = C[0,0] + C[2,2] - 2*C[1,2] -4*C[4,4]
             B2 = C[1,2] - C[0,1]
             B3 = C[4,4] - 0.5*(C[0,0]-C[0,1])
             B4 = C[0,1]
             B5 = 0.5*(C[0,0]-C[0,1])
             #map between tensor and engineering notation
             code = {0:(0,0),1:(1,1),2:(2,2),3:(1,2),4:(0,2),5:(0,1)}
             C_avg = np.zeros((6,6))
             for i in range(6):
                 for j in range(6):
                     C_avg[i,j] = B1*A4[i,j] + B2*(
                     a2[code[i][0],code[i][1]]*(code[j][0]==code[j][1]) +
                         a2[code[j][0],code[j][1]]*(code[i][0]==code[i][1])) + B3*
                     a2[code[i][0],code[j][0]]*(code[i][1]==code[j][1]) +
                     a2[code[i][0],code[j][1]]*(code[i][1]==code[j][0]) +
                     a2[code[i][1],code[j][0]]*(code[i][0]==code[j][1]) +
                     a2[code[i][1],code[j][1]]*(code[i][0]==code[j][0])) + B4*(
                     (code[i][0]==code[i][1])*(code[j][0]==code[j][1])) + B5*(
                     (code[i][0]==code[j][0])*(code[i][1]==code[j][1]) +
                     (code[i][0]==code[j][1])*(code[i][1]==code[j][0]))
             return C_avg
```

And finally we will compute the linear and quadratic closure approxmiations

```
In [12]: #Uni-directional stiffness tensor
         C_mt_uni = C_mt[-1,:,:]
         C_E_uni = C_E[-1,:,:]

         #linear closure
         a2 = 0.33333*np.eye(3)
         a4_l = np.zeros((3,3,3,3))
         for i in range(3):
             for j in range(3):
                 for k in range(3):
                     for l in range(3):
                         a4_l[i,j,k,l] = -1./35.*((i==j)*(k==l) + (i==k)*(j==l) +
                         a2[i,j]*(k==l) + a2[i,k]*(j==l) +a2[i,l]*(j==k) + a2[k,l]*
                         a2[j,l]*(i==k) + a2[j,k]*(i==l))
         A4_l = tens2eng(a4_l)
         C_mt_l = ornavg(C_mt_uni,A4_l,a2)
         C_E_l = ornavg(C_E_uni,A4_l,a2)

         #quadratic closure
         a4_q = np.zeros((3,3,3,3))
         for i in range(3):
             for j in range(3):
                 for k in range(3):
```

```
                for l in range(3):
                    a4_q[i,j,k,l] = a2[i,j]*a2[k,l]
    A4_q = tens2eng(a4_q)
    C_mt_q = ornavg(C_mt_uni, A4_q,a2)
    C_E_q = ornavg(C_E_uni, A4_q,a2)
```

We would expect the stiffness to be the same in all three directions for a perfectly 3D-random composite, we can check if that is true

```
In [13]: S_mt_l = np.linalg.inv(C_mt_l)
         S_E_l = np.linalg.inv(C_E_l)
         S_mt_q = np.linalg.inv(C_mt_q)
         S_E_q = np.linalg.inv(C_E_q)

         print 'Mori-tanaka, linear'
         print np.array([1/S_mt_l[0,0], 1/S_mt_l[1,1], 1/S_mt_l[2,2]])*1e-9

         print 'Eshelby, linear'
         print np.array([1/S_E_l[0,0], 1/S_E_l[1,1], 1/S_E_l[2,2]])*1e-9

         print 'Mori-tanaka, quadratic'
         print np.array([1/S_mt_q[0,0], 1/S_mt_q[1,1], 1/S_mt_q[2,2]])*1e-9

         print 'Eshelby, quadratic'
         print np.array([1/S_E_q[0,0], 1/S_E_q[1,1], 1/S_E_q[2,2]])*1e-9
```

```
Mori-tanaka, linear
[ 42.72129622  42.72129622  42.72129622]
Eshelby, linear
[ 34.85523147  34.85523147  34.85523147]
Mori-tanaka, quadratic
[ 33.33129101  33.33129101  33.33129101]
Eshelby, quadratic
[ 23.59922809  23.59922809  23.59922809]
```

Interestingly, we find that although the stiffness is equal across all three axes no matter which closure approximation is used, the quadratic closure predicts a significantly lower stiffness, we can also check the shear stiffnesses as a comparison.

```
In [14]: print 'Mori-tanaka, linear'
         print np.array([1/S_mt_l[3,3], 1/S_mt_l[4,4], 1/S_mt_l[5,5]])*1e-9

         print 'Eshelby, linear'
         print np.array([1/S_E_l[3,3], 1/S_E_l[4,4], 1/S_E_l[5,5]])*1e-9

         print 'Mori-tanaka, quadratic'
         print np.array([1/S_mt_q[3,3], 1/S_mt_q[4,4], 1/S_mt_q[5,5]])*1e-9
```

```
        print 'Eshelby, quadratic'
        print np.array([1/S_E_q[3,3], 1/S_E_q[4,4], 1/S_E_q[5,5]])*1e-9
```

```
Mori-tanaka, linear
[ 16.37265737  16.37265737  16.37265737]
Eshelby, linear
[ 13.21484283  13.21484283  13.21484283]
Mori-tanaka, quadratic
[ 12.3669996  12.3669996  12.3669996]
Eshelby, quadratic
[ 8.5671465  8.5671465  8.5671465]
```

Once again we find isotropic properties, with the quadratic closure approximation significantly lower than the linear. For truly random, we expect the linear approximation to be correct, but the large difference indicates the error that can be introduced from an improper closure approximation.

```
In [ ]:
```