Problem set 2: in TKT4150 Biomechanics and TTK4170 Modelling and identification of biological systems

AUTHOR:

10 September 2016

Exercise 1: Cauchy's stress theorem used on the foot of a running human

a) Rewrite the Cauchy stress theorem in index notation.

Solution.

$$\begin{Bmatrix} t_1 \\ t_2 \\ t_3 \end{Bmatrix} = \begin{bmatrix} T_{11} & T_{12} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ T_{31} & T_{32} & T_{33} \end{bmatrix} \begin{Bmatrix} n_1 \\ n_2 \\ n_3 \end{Bmatrix} \Rightarrow t_i = T_{ij} n_j \tag{1}$$

b) The foot of a running human is exposed to a force from the ground, as is seen in Figure 1.
Let's say the stress matrix corresponding to a point on the foot where the foot hits the ground is assumed to be

$$\mathbf{T} = \begin{bmatrix} 0 & 20 & 0 \\ 20 & -40 & 0 \\ 0 & 0 & 0 \end{bmatrix} MPa \tag{2}$$

Find an estimate of the angle, α, on which the force from the ground acts on the foot.

Solution. Let's first find the traction between the foot and the ground:

$$\mathbf{t} = \mathbf{T} \cdot \mathbf{n} = \begin{bmatrix} 0 & 20 & 0 \\ 20 & -40 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} 0 \\ -1 \\ 0 \end{Bmatrix} MPa = \begin{Bmatrix} -20 \\ 40 \\ 0 \end{Bmatrix} MPa \tag{3}$$

Then, if we assume that the angle of this traction and the angle of the net force is equal, we get this value for α:

$$\alpha = \arctan\left(\frac{40}{20}\right) = \arctan(2) \approx 63\,\mathrm{deg} \tag{4}$$

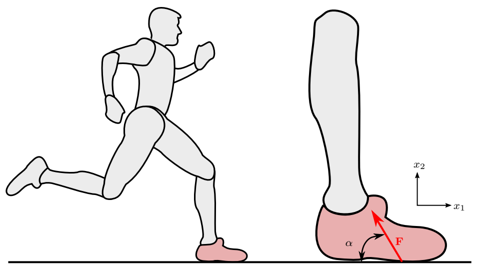


Figure 1: Running human.

Exercise 2: Cauchy equations

a) Use the Cauchy equations (1.89) and the fact that a fluid at rest has an isotropic state of stress (1.86), to derive the formula for the pressure in a still body of fluid as a function of the vertical distance z from a free surface (assume a pressure p_0 at the surface).

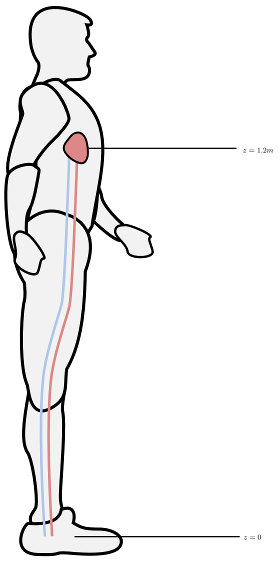


Figure 2: Sketch of arteries and veins between heart and foot.

Solution. We calculate the first term of the Cauchy equation:

$$T_{ij,j} = \frac{\partial}{\partial x_j}(T_{ij}) = \frac{\partial}{\partial x_j}(-p\delta_{ij}) = -\frac{\partial p}{\partial x_j}\delta_{ij} \tag{5}$$

$$= -\frac{\partial p}{\partial x_1}\delta_{i1} - \frac{\partial p}{\partial x_2}\delta_{i2} - \frac{\partial p}{\partial x_3}\delta_{i3} \tag{6}$$

Because the pressure is independent of the x- or y-coordinate, the equation reduces to

$$T_{ij,j} = -\frac{\partial p}{\partial x_3} \quad \text{for i=3} \tag{7}$$

$$= 0 \quad \text{otherwise.} \tag{8}$$

Assuming no external acceleration of the container, Cauchy equation for component x_3 (or z) becomes

$$-\frac{\partial p}{\partial z} + \rho g = 0 \tag{9}$$

which further leads to

$$\nabla p = \frac{\partial p}{\partial z} = \rho g \tag{10}$$

By integrating this, we get

$$p(z) = \int \rho g dz = \rho g z + C \tag{11}$$

By setting $p(z=0) = p_0$, we get $C = p_0$, and thus

$$p(z) = \rho g z + p_0 \tag{12}$$

b) The average static pressure $p - p_1$, where p_1 is the atmospheric pressure, may in veins (p_v) and arteries (p_a) near the heart region be set equal to

$$p_{vh} = 1kPa, \quad p_{ah} = 10kPa \tag{13}$$

Use this to make an estimate of p_v and p_a in the foot of a standing human, when the elevation between the foot and the heart is equal to $1.2m$, as indicated in Figure 2. The density of blood is assumed $\rho = 1000kg/m^3$.

Solution. For connected fluid columns, we can use the formula which was derived above: $p(z) = p_0 + pgz$.

$$p_a(1.2m) = p_{ah} + \rho g h = 10kPa + 1000\frac{kg}{m^3} \cdot 9.81\frac{m}{s^2} \cdot 1.2m = 22kPa \tag{14}$$

$$p_v(1.2m) = p_{vh} + \rho g h = 1kPa + 1000\frac{kg}{m^3} \cdot 9.81\frac{m}{s^2} \cdot 1.2m = 13kPa \tag{15}$$

Exercise 3: Computational Prinicpal Stress Analisys

For the following stress matrix:

$$\mathbf{T} = \begin{bmatrix} 90 & -30 & 0 \\ -30 & 120 & -30 \\ 0 & -30 & 90 \end{bmatrix} MPa \tag{16}$$

you will calculate the principal stresses, and directions, as well as use `Python`, `Matlab` or another computational tool of your choice to calculate these quantities as well. The file `exercise_6.ipynb` is an IPython Notebook to help you get started if you would like. You can upload this to https://tmpnb.org and get started if you do not have access to `Python` on your local computer. Remember that to save it you will need to download the notebook, as https://tmpnb.org does not provide permanent storage.

a) Establish the principal stresses σ_1, σ_2 and σ_3, and the corresponding directions $\mathbf{n}_1$, $\mathbf{n}_2$ and $\mathbf{n}_3$ with hand-calculations.

Hint. Remember that $\|\mathbf{T}\|$ is the **Frobenius Norm**. It may be easier to solve the cubic polynomial $\det(\sigma\mathbf{1} - \mathbf{T}) = 0$ using cofactor expansion to find the determinant as opposed to the formula in terms of invariants.

b) Find the numerical values for the three stress invariants I, II and III, using `Python` or `Matlab`.

Hint. Remember that $\|\mathbf{T}\|$ is the **Frobenius Norm**.

c) Use the invariants to determine the characteristic polynomial from the eigenvalue problem. Furthermore, plot the polynomial and graphically determine the principal stresses.

d) Use `Python` or `Matlab` to find the solutions of the characteristic polynomial.

e) Use the linear algebra library of `Python` or `Matlab` to directly solve for the principal stresses and corresponding directions.

f) Plot the principal direction vectors $\mathbf{n}_i$ (i=1,2,3) using `Python` or `Matlab`. (Make sure it is possible to see how they are oriented relative to each other. It may be necessary to have two figures from different angles.)

g) The maximum shear stress is found in an orientation between the directions of the smallest and largest principal stresses, σ_1 and σ_3, and in the plane normal to the direction of σ_2. Determine the orientation where the stress element gets the largest shear stress, and the numerical value of this stress. Plot the found normal vector together with the vectors in the previous problem.

Tensor_calculations_solution

September 19, 2017

```
In [1]: %matplotlib inline
        import numpy as np
        from matplotlib import pyplot
```

$$\mathbf{T} = \begin{bmatrix} 90 & -30 & 0 \\ -30 & 120 & -30 \\ 0 & -30 & 90 \end{bmatrix} MPa \tag{1}$$

```
In [2]: T= np.array([[90, -30, 0],
                     [-30, 120, -30],
                     [0, -30, 90]])
        print("T={}".format(T))

T=[[ 90 -30   0]
 [-30 120 -30]
 [  0 -30  90]]
```

a) Establish the principal stresses σ_1, σ_2 and σ_3, and the corresponding directions $\mathbf{n}_1$, $\mathbf{n}_2$ and $\mathbf{n}_3$ with hand-calculations.

Cofactor expansion along row 1 gives

$$\det(\mathbf{T} - \lambda\mathbf{1}) = (90-\lambda)\{(120-\lambda)(90-\lambda) - (-30)^2\} - (-30)\{(-30)(90-\lambda) - 0(-30)\} \tag{2}$$

Expanding terms we see that

$$\begin{aligned} \det(\mathbf{T} - \lambda\mathbf{1}) &= (120-\lambda)(90-\lambda)^2 - 900(90-\lambda) + -900(90-\lambda) &(3)\\ &= (90-\lambda)\{(120-\lambda)(90-\lambda) - 1800\} &(4) \end{aligned}$$

Clearly $\lambda = 90$ is on zero, and the others occur when $(120-\lambda)(90-\lambda) - 1800 = 0$. This quadratic equation has two real roots at 150 and 60. Thus the principal stresses are $\sigma_1 = 150$MPa, $\sigma_2 = 90$MPa, and $\sigma_3 = 60$MPa.

To find the principle directions we must solve $(\mathbf{T} - \sigma_i\mathbf{1})\mathbf{n}_i = \mathbf{0}$ Thus we have for $\sigma_2 = 90$

$$\begin{bmatrix} 0 & -30 & 0 \\ -30 & 30 & -30 \\ 0 & -30 & 0 \end{bmatrix} \begin{bmatrix} n_1 \\ n_2 \\ n_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{5}$$

which simplifies to

$$\begin{bmatrix} -30n_2 \\ -30n_1 + 30n_2 - 30n_3 \\ -30n_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{6}$$

Thus $n_2 = 0$ and we solve $-30n_1 - 30n_3 = 0$ to see that $n_1 = -n_3$ so the eigenvector $\mathbf{n}_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$.
Similarly you may find the other two eigen vectors.

b) Find the numerical values for the three stress invariants I, II and III.

```
In [3]: I = np.trace(T)
        II = 0.5*(np.trace(T)**2 - np.trace(np.dot(T,T)))
        III = np.linalg.det(T)
        print(I,II,III)
```

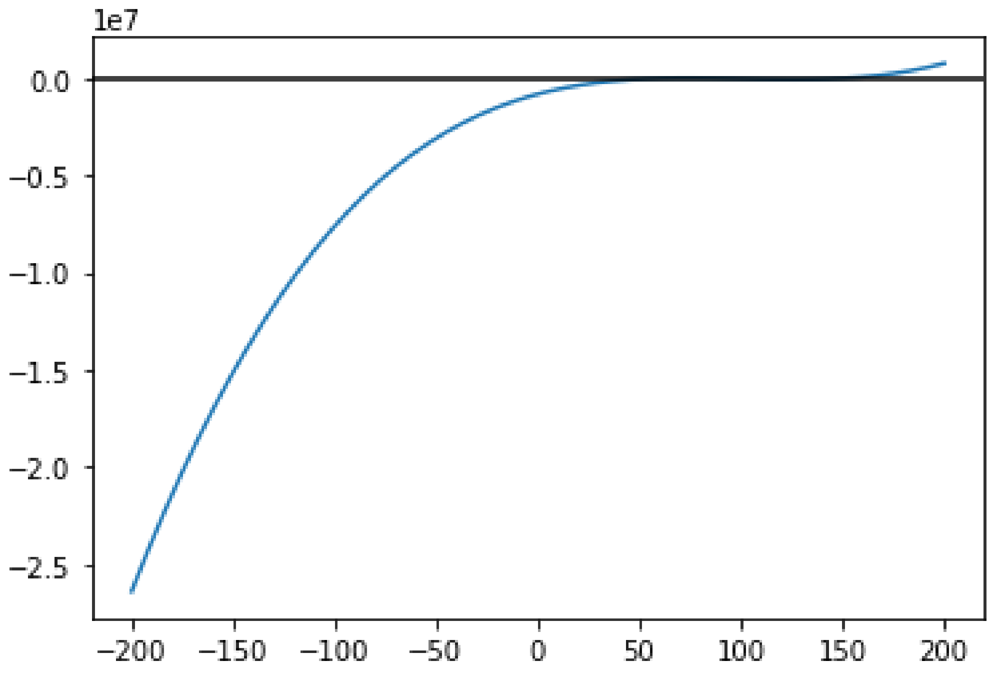
```
300 27900.0 810000.0
```

c) Use the invariants to establish the characteristic polynomial from the eigenvalue problem. Furthermore, plot the polynomial and graphically determine the principal stresses.

```
In [4]: polynomial = lambda x: x**3 - I*x**2 + II*x - III
```

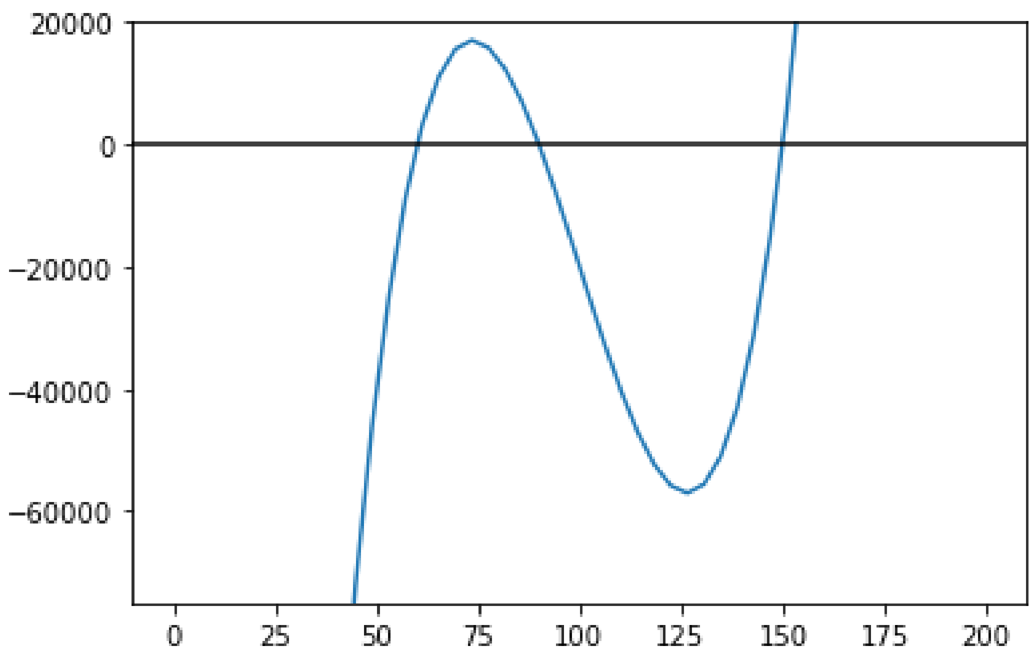
```
In [5]: from matplotlib import pyplot as plt
        lower_bound = -200
        upper_bound = 200
        n_pts = 50
        xvals = np.linspace(lower_bound,upper_bound,n_pts) # Define points at which to evaluate
        plt.plot(xvals,polynomial(xvals)) # Plot the polynomial at those points
        plt.axhline(0, color="black")
```

```
Out[5]: <matplotlib.lines.Line2D at 0x7f4935046748>
```



Adjust the limits to better see the zeros

```
In [6]: xvals = np.linspace(0,upper_bound,n_pts)
        plt.plot(xvals,polynomial(xvals))
        plt.ylim(-75000,20000)
        plt.axhline(0, color="black")

Out[6]: <matplotlib.lines.Line2D at 0x7f493199fac8>
```



d) Use `fsolve` from the `scipy.optimize` library to find the solutions of the characteristic polynom.

```
In [7]: from scipy import optimize as opt
        initial_guess = 50
        print(opt.fsolve(polynomial,initial_guess))
        initial_guess = 100
        print(opt.fsolve(polynomial,initial_guess))
        initial_guess = 160
        print(opt.fsolve(polynomial,initial_guess))
```

```
[ 60.]
[ 90.]
[ 150.]
```

e) Use `eig` to directly solve for the principal stresses and corresponding directions.

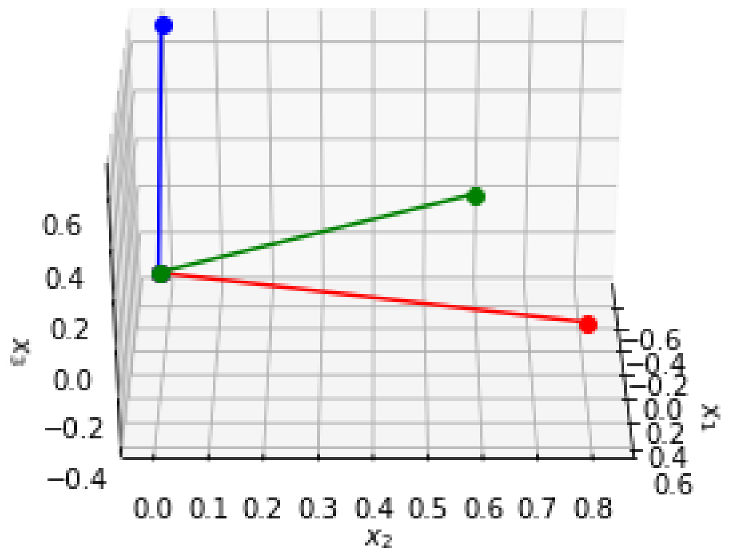
```
In [8]: lambdas, ns = np.linalg.eig(T)
        print(ns)
```

```
[[ -4.08248290e-01  -7.07106781e-01   5.77350269e-01]
 [  8.16496581e-01   1.35927168e-16   5.77350269e-01]
 [ -4.08248290e-01   7.07106781e-01   5.77350269e-01]]
```

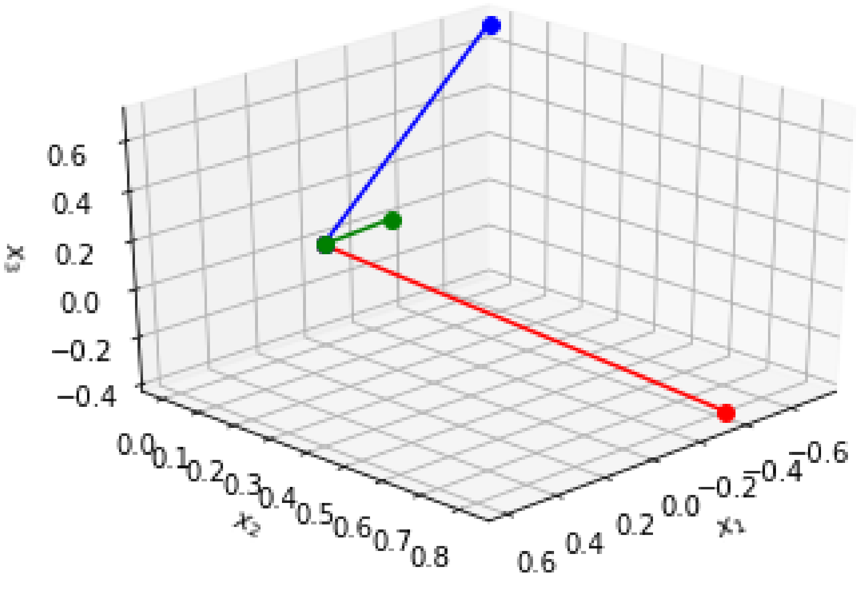
f) Plot the principal direction vectors $\mathbf{n}_i$ (i=1,2,3).

```
In [9]: from mpl_toolkits.mplot3d import Axes3D
        #plt.quiver(origin[:,0],ns[:,0])
        #ax.quiver(0,0,0,ns[0,0],ns[0,1],ns[0,2],pivot="tip")
        #ax.quiver(0,0,0,ns[1,0],ns[1,1],ns[1,2],pivot="tip")
        #ax.quiver(0,0,0,ns[2,0],ns[2,1],ns[2,2],pivot="tip")
        # fig = plt.figure()
        # ax = fig.gca(projection='3d')
        # ax.quiver(ns[0,0],ns[0,1],ns[0,2],ns[0,0],ns[0,1],ns[0,2],   # data
        #           length=1,                              # arrow length
        #           color='red'                           # arrow colour
        #           )
        # ax.quiver(ns[1,0],ns[1,1],ns[1,2],ns[1,0],ns[1,1],ns[1,2],   # data
        #           length=1,                              # arrow length
        #           color='blue'                            # arrow colour
        #           )
        # ax.quiver(ns[2,0],ns[2,1],ns[2,2],ns[2,0],ns[2,1],ns[2,2],   # data
        #           length=1,                              # arrow length
        #           color='green'                            # arrow colour
        #             )
        # ax.set_xlim(-1,1)
        # ax.set_ylim(-1,1)
        # ax.set_zlim(-1,1)
        fig = plt.figure()
        ax = fig.gca(projection='3d')
        ax.plot([0,ns[0,0]],[0,ns[1,0]],[0,ns[2,0]],'-ro')
        ax.plot([0,ns[0,1]],[0,ns[1,1]],[0,ns[2,1]],'-bo')
        ax.plot([0,ns[0,2]],[0,ns[1,2]],[0,ns[2,2]],'-go')
        ax.view_init(elev=30, azim=0)
        ax.set_xlabel(r'$x_1$')
        ax.set_ylabel(r'$x_2$')
        ax.set_zlabel(r'$x_3$')
        fig = plt.figure()
        ax = fig.gca(projection='3d')
        ax.plot([0,ns[0,0]],[0,ns[1,0]],[0,ns[2,0]],'-ro')
        ax.plot([0,ns[0,1]],[0,ns[1,1]],[0,ns[2,1]],'-bo')
        ax.plot([0,ns[0,2]],[0,ns[1,2]],[0,ns[2,2]],'-go')
        ax.view_init(elev=30, azim=45)
        ax.set_xlabel(r'$x_1$')
        ax.set_ylabel(r'$x_2$')
        ax.set_zlabel(r'$x_3$')
        fig = plt.figure()
        ax = fig.gca(projection='3d')
        ax.plot([0,ns[0,0]],[0,ns[1,0]],[0,ns[2,0]],'-ro')
        ax.plot([0,ns[0,1]],[0,ns[1,1]],[0,ns[2,1]],'-bo')
        ax.plot([0,ns[0,2]],[0,ns[1,2]],[0,ns[2,2]],'-go')
        ax.view_init(elev=30, azim=90)
        ax.set_xlabel(r'$x_1$')
        ax.set_ylabel(r'$x_2$')
        ax.set_zlabel(r'$x_3$')
        fig = plt.figure()
        ax = fig.gca(projection='3d')
        ax.plot([0,ns[0,0]],[0,ns[1,0]],[0,ns[2,0]],'-ro')
        ax.plot([0,ns[0,1]],[0,ns[1,1]],[0,ns[2,1]],'-bo')
        ax.plot([0,ns[0,2]],[0,ns[1,2]],[0,ns[2,2]],'-go')
        ax.view_init(elev=70, azim=150)
        ax.set_xlabel(r'$x_1$')
        ax.set_ylabel(r'$x_2$')
        ax.set_zlabel(r'$x_3$')
```

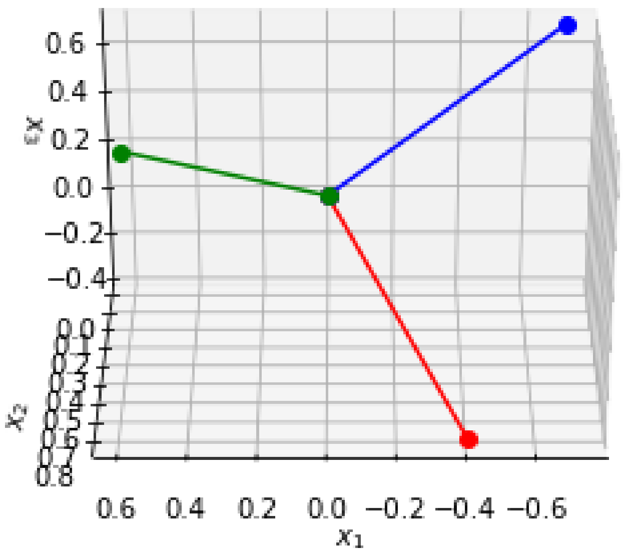
```
Out[9]: <matplotlib.text.Text at 0x7f4959e20780>
```

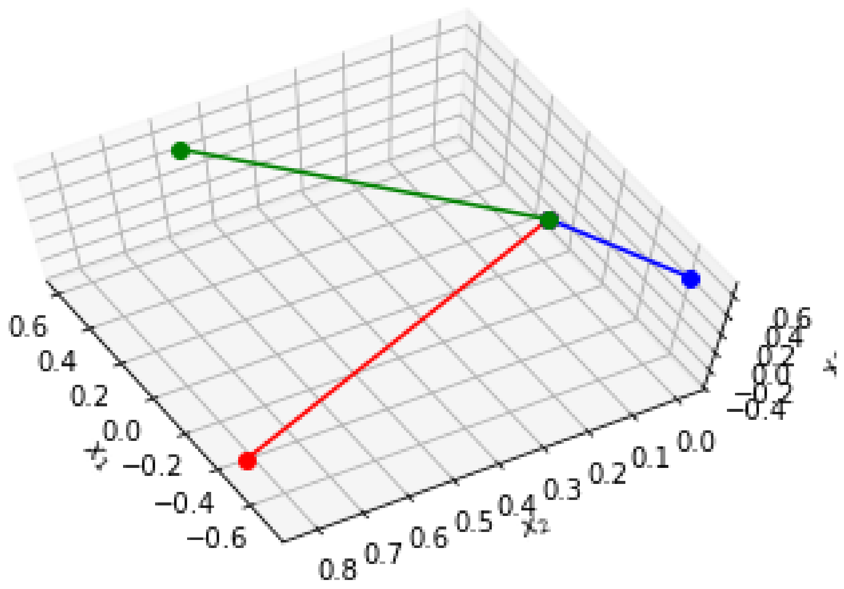


x_3
x_2
x_1
0.6
0.4
0.2
0.0
−0.2
−0.4
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8
−0.6 −0.4 −0.2 0.0 0.2 0.4 0.6



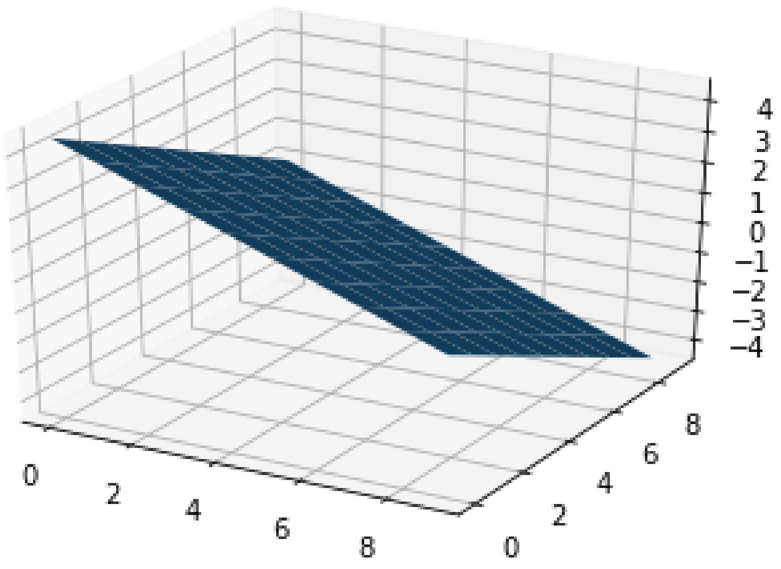
x_3
x_2
x_1
0.6
0.4
0.2
0.0
−0.2
−0.4
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8
0.6 0.4 0.2 0.0 −0.2 −0.4 −0.6





g) The maximum shear stress is found in an orientation between the directions of the smallest and largest principal stresses, σ_1 and σ_3, and in the plane normal to the direction of σ_2. Determine the orientation where the stress element gets the largest shear stress, and the numerical value of this stress. Plot the found normal vector together with the vectors in f).

```
In [10]: # See <http://stackoverflow.com/questions/3461869/plot-a-plane-based-on-a-normal-vector
         import numpy as np
         import matplotlib.pyplot as plt
         from mpl_toolkits.mplot3d import Axes3D

         point  = np.array([1, 2, 3])
         normal = np.array([1, 1, 2])

         # a plane is a*x+b*y+c*z+d=0
         # [a,b,c] is the normal. Thus, we have to calculate
         # d and we're set
         d = -point.dot(normal)

         # create x,y
         xx, yy = np.meshgrid(range(10), range(10))

         # calculate corresponding z
         z = (-normal[0] * xx - normal[1] * yy - d) * 1. /normal[2]

         # plot the surface
         plt3d = plt.figure().gca(projection='3d')
         plt3d.plot_surface(xx, yy, z)
         plt.show()
```

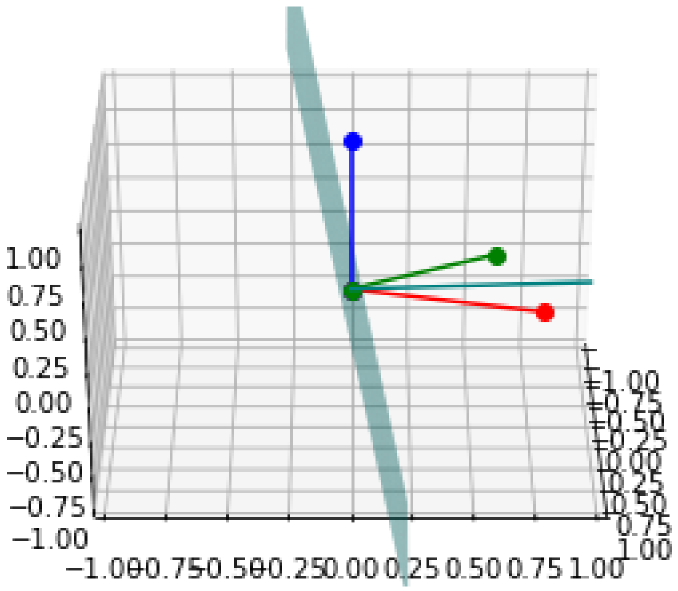


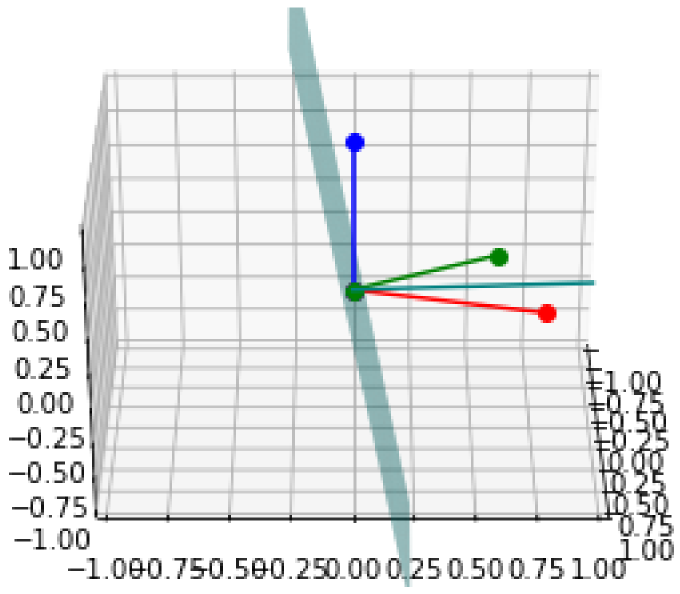
```
In [11]: # See <http://stackoverflow.com/questions/3461869/plot-a-plane-based-on-a-normal-vector
         from mpl_toolkits.mplot3d import Axes3D
```

```
normal_to_max_shear = (ns[:,0]+ns[:,2])/np.linalg.norm(ns[:,0]+ns[:,2])
point  = np.array([0, 0, 0])
normal = normal_to_max_shear
print(normal)

# a plane is a*x+b*y+c*z+d=0
# [a,b,c] is the normal. Thus, we have to calculate
# d and we're set
d = -point.dot(normal)

# create x,y
xvals = np.linspace(-0.25,.25,100)
yvals = np.linspace(-0.25,.25,100)
xx, yy = np.meshgrid(xvals,yvals)

# calculate corresponding z
z = (-normal[0] * xx - normal[1] * yy - d) * 1. /normal[2]

# plot the surface
plt3d = plt.figure().gca(projection='3d')
plt3d.plot_surface(xx, yy, z,linewidth=0.0,color=(0.0,0.5,0.5,0.5))
plt3d.plot([0,ns[0,0]],[0,ns[1,0]],[0,ns[2,0]],'-ro')
plt3d.plot([0,ns[0,1]],[0,ns[1,1]],[0,ns[2,1]],'-bo')
plt3d.plot([0,ns[0,2]],[0,ns[1,2]],[0,ns[2,2]],'-go')
plt3d.plot([0,normal[0]],[0,normal[1]],[0,normal[2]],color=(0.0,0.5,0.5))
plt3d.set_xlim((-1,1))
plt3d.set_ylim(-1,1)
plt3d.set_zlim(-1,1)
plt3d.view_init(elev=30, azim=0)
plt.show()# See <http://stackoverflow.com/questions/3461869/plot-a-plane-based-on-a-nor
from mpl_toolkits.mplot3d import Axes3D


normal_to_max_shear = (ns[:,0]+ns[:,2])/np.linalg.norm(ns[:,0]+ns[:,2])
point  = np.array([0, 0, 0])
normal = normal_to_max_shear

# a plane is a*x+b*y+c*z+d=0
# [a,b,c] is the normal. Thus, we have to calculate
# d and we're set
d = -point.dot(normal)

# create x,y
xvals = np.linspace(-0.25,.25,100)
yvals = np.linspace(-0.25,.25,100)
xx, yy = np.meshgrid(xvals,yvals)
```

```
    # calculate corresponding z
    z = (-normal[0] * xx - normal[1] * yy - d) * 1. /normal[2]

    # plot the surface
    plt3d = plt.figure().gca(projection='3d')
    plt3d.plot_surface(xx, yy, z,linewidth=0.0,color=(0.0,0.5,0.5,0.5))
    plt3d.plot([0,ns[0,0]],[0,ns[1,0]],[0,ns[2,0]],'-ro')
    plt3d.plot([0,ns[0,1]],[0,ns[1,1]],[0,ns[2,1]],'-bo')
    plt3d.plot([0,ns[0,2]],[0,ns[1,2]],[0,ns[2,2]],'-go')
    plt3d.plot([0,normal[0]],[0,normal[1]],[0,normal[2]],color=(0.0,0.5,0.5))
    plt3d.set_xlim((-1,1))
    plt3d.set_ylim(-1,1)
    plt3d.set_zlim(-1,1)
    plt3d.view_init(elev=30, azim=0)
    plt.show()
```

```
[ 0.11957316  0.98559856  0.11957316]
```




1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00