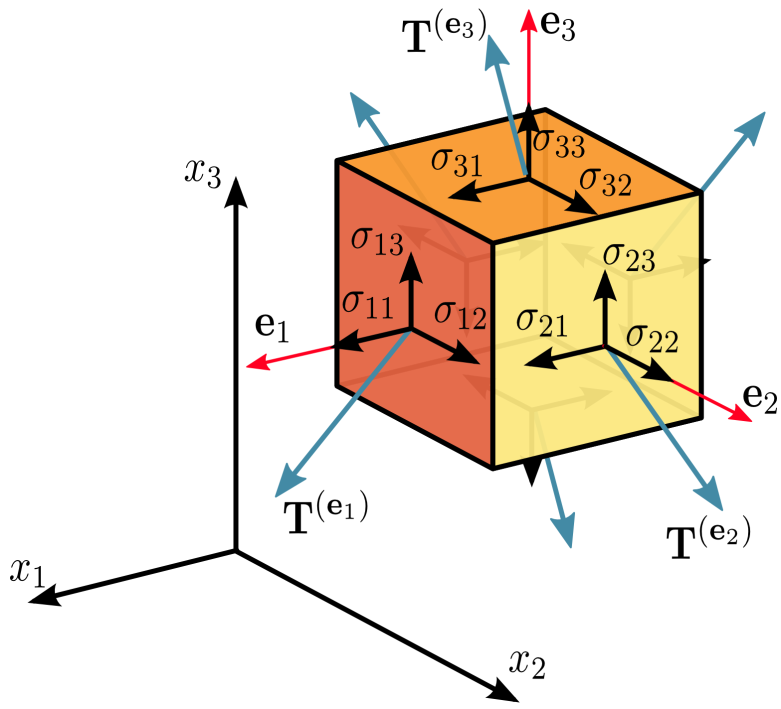
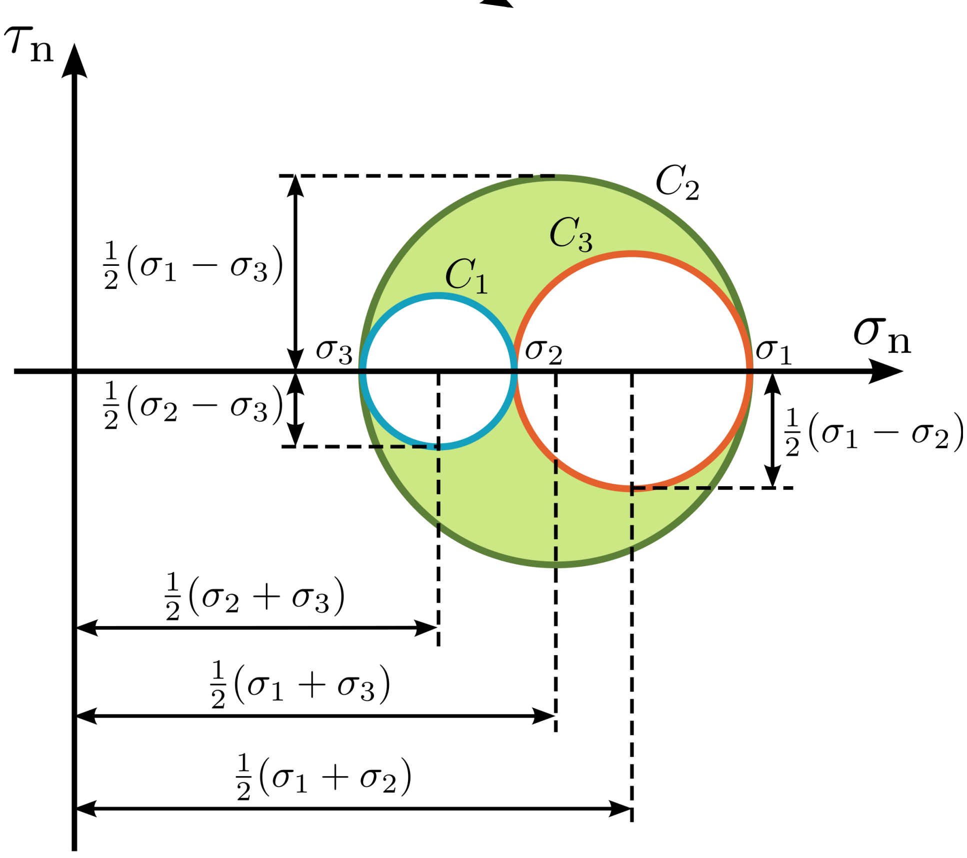
Principle Normal Stresses 3D Triaxial





Hydrostatic and deviatoric components The stress tensor can be separated into two components. One component is a hydrostatic or dilatational stress that acts to change the volume of the material only; the other is the deviatoric stress that acts to change the shape only.

$$\begin{pmatrix} \sigma_{11} & \sigma_{12} & \sigma_{31} \\ \sigma_{12} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{23} & \sigma_{33} \end{pmatrix} = \begin{pmatrix} \sigma_H & 0 & 0 \\ 0 & \sigma_H & 0 \\ 0 & 0 & \sigma_H \end{pmatrix} + \begin{pmatrix} \sigma_{11} - \sigma_H & \sigma_{12} & \sigma_{31} \\ \sigma_{12} & \sigma_{22} - \sigma_H & \sigma_{23} \\ \sigma_{31} & \sigma_{23} & \sigma_{33} - \sigma_H \end{pmatrix}$$

where the hydrostatic stress is given by

$$\sigma_H = (\sigma_1 + \sigma_2 + \sigma_3)/3$$

very nice interactive mohrs

From Wikipedia The Cauchy stress tensor at a particular material point are known with respect to a coordinate system. The Mohr circle is then used to determine graphically the stress components acting on a rotated coordinate system (i.e., acting on a differently oriented plane passing through that point). The abscissa σ_{n} and ordinate τ_{n} of each point on the circle, are the magnitudes of the normal stress and shear stress components, respectively, acting on the rotated coordinate system. In other words, the circle is the locus of points that represent the state of stress on individual planes at all their orientations, where the axes represent the principal axes of the stress element.

$$\boldsymbol{\sigma} = \sigma_{ij} = \left[\mathbf{T}^{(\mathbf{e}_1)} \quad \mathbf{T}^{(\mathbf{e}_2)} \quad \mathbf{T}^{(\mathbf{e}_3)} \right] = \begin{bmatrix} \sigma_{11} & \sigma_{21} & \sigma_{31} \\ \sigma_{12} & \sigma_{22} & \sigma_{32} \\ \sigma_{13} & \sigma_{23} & \sigma_{33} \end{bmatrix}$$

$$\begin{bmatrix} T_1^{(n)} \\ T_2^{(n)} \\ T_3^{(n)} \end{bmatrix} = \begin{bmatrix} \sigma_{11} & \sigma_{21} & \sigma_{31} \\ \sigma_{12} & \sigma_{22} & \sigma_{32} \\ \sigma_{13} & \sigma_{23} & \sigma_{33} \end{bmatrix} \cdot \begin{bmatrix} n_1 \\ n_2 \\ n_3 \end{bmatrix} \quad \text{or} \quad \mathbf{T}^{(\mathbf{n})} = \sigma_{ij} \cdot \mathbf{n}$$

$$M = \begin{bmatrix} S_{xx} & S_{yx} & S_{zx} \\ S_{xy} & S_{yy} & S_{yz} \\ S_{xz} & S_{yz} & S_{zz} \end{bmatrix} = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{zx} \\ \tau_{xy} & \sigma_y & \tau_{zx} \\ \tau_{xz} & \tau_{xy} & \sigma_z \end{bmatrix}$$

Explain why its the shaded area outside of small circles!!!!!!!

Characteristic polynomial equation

$$\sigma^3 - I_1\sigma^2 + I_2\sigma - I_3 = 0$$

Principle Scalar Invariants

$$\begin{aligned} I_1 = &\ \sigma_{11} + \sigma_{22} + \sigma_{33} \\ = &\ \sigma_{kk} \\ I_2 = &\ \begin{vmatrix} \sigma_{22} & \sigma_{23} \\ \sigma_{32} & \sigma_{33} \end{vmatrix} + \begin{vmatrix} \sigma_{11} & \sigma_{13} \\ \sigma_{31} & \sigma_{33} \end{vmatrix} + \begin{vmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{vmatrix} \\ = &\ \sigma_{11}\sigma_{22} + \sigma_{22}\sigma_{33} + \sigma_{11}\sigma_{33} - \sigma_{12}^2 - \sigma_{23}^2 - \sigma_{13}^2 \\ = &\ \frac{1}{2}\left(\sigma_{ii}\sigma_{jj} - \sigma_{ij}\sigma_{ji}\right) \\ I_3 = &\ \begin{vmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{vmatrix} \\ = &\ \sigma_{11}\sigma_{22}\sigma_{33} + 2\sigma_{12}\sigma_{23}\sigma_{31} - \sigma_{12}^2\sigma_{33} - \sigma_{23}^2\sigma_{11} - \sigma_{13}^2\sigma_{22} \end{aligned}$$

$$\sigma^3 - A\sigma^2 + B\sigma - C = 0$$

Polynomial coefficient (A)

$$= \sigma_x + \sigma_y + \sigma_z$$

$$= \sigma_1^{'} + \sigma_2^{'} + \sigma_3^{'}$$

Polynomial coefficient (B)

$$= \sigma_x\sigma_y + \sigma_y\sigma_z + \sigma_x\sigma_z - \tau_{xy}^2 - \tau_{yz}^2 - \tau_{xz}^2$$
$$= \sigma_1^{'}\sigma_2^{'} + \sigma_2^{'}\sigma_3^{'} + \sigma_1^{'}\sigma_3^{'}$$

polynomial coefficient (C)

$$= \sigma_x\sigma_y\sigma_z + 2\tau_{xy}\tau_{yz}\tau_{xz} - \sigma_x(\tau_{yz})^2 - \sigma y(\tau_{xz})^2 - \sigma z(\tau_{xy})^2$$
$$= \sigma'1\sigma'2\sigma'3$$

---

principal stress

$$\sigma_1 = max(\sigma_1^{'}, \sigma_2^{'}, \sigma_3)$$
$$\sigma_2 = A - \sigma_1^{'} - \sigma_2^{'}$$
$$\sigma_3 = min(\sigma_1^{'}, \sigma_2^{'}, \sigma_3^{'})$$

max shear stress

$$\tau_{max1} = (\sigma_2 - \sigma_3)/2$$
$$\tau_{max2} = (\sigma_1 - \sigma_3)/2$$
$$\tau_{max3} = (\sigma_1 - \sigma_2)/2$$

Failure Theory

DUCTILE yeild as a function of **Yield Strength**

> (*MSS Tresca*) or (*Hill*) or (*Garson*)

> (*Distortion Energy = Von Mises = Octahedral Shear Stress Energy*)

BRITTLE fracture as function of **Ultimate Strength**

> (*Rankine*) or (*Brittle Coulumb-Mohr*) or (*Modified-Mohr*)

Rankine (Maximum Principle Stress theory) BRITTLE

easy, but not great

$$\sigma_1 = \sigma_Y, \sigma_U$$
$$\sigma_3 = -\sigma_Y, -\sigma_U$$

Maximum Normal Stress Theory BRITTLE

Failure (i.e. yielding, fracture) is expected to occur if the maximum normal stress in the part exceeds the maximum normal stress in test specimen at failure (yielding, fracture). where σ_t and σ_c are test specimen tensile and compressive strengths, respectively.

$$\max(\sigma_1, \sigma_2, \sigma_3) \geq \sigma_c$$
$$\min(\sigma_1, \sigma_2, \sigma_3) \leq \sigma_c$$

Maximum Shearing Stress Theory (MSS) DUCTILE

Here failure is predicted to occur if the maximum shearing stress (the principal shearing stress) is equal to or exceeds the maximum shear stress in a test specimen at failure. τ_c is the maximum shearing stress in the test specimen, and for a uniaxial tension test the max shear stress occurs at 45 degrees to the applied load direction and is equal to half of the first principal stress, which is the nominal tensile stress.

This theory applies well to ductile materials and to ductile yielding.

$$\max(|\tau_1|, |\tau_2|, |\tau_3|) \geq \tau_c$$

Distortion Energy Theory (von Mises) DUCTILE

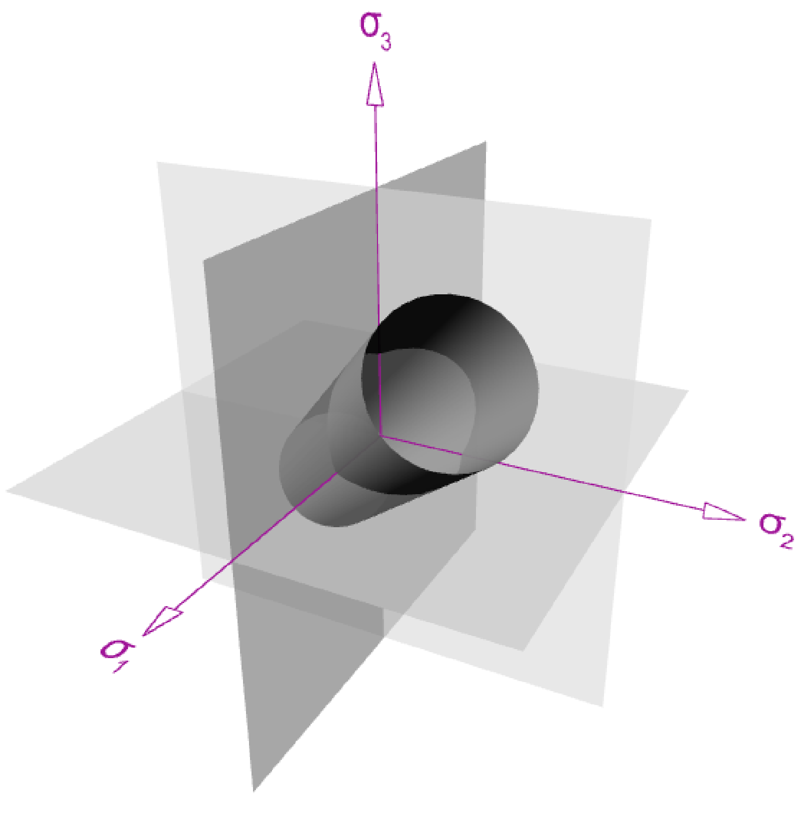
This theory postulates that the distortion energy - that which contributes to shape change and not to change in volume - affects the failure of the part. Dilation energy, that which only changes the volume, does not contribute to the failure of the part. The latter is produced by hydrostatic stress, which has been shown to not induce yielding or fracture (none under compression, but fracture under high levels of tensile stress).

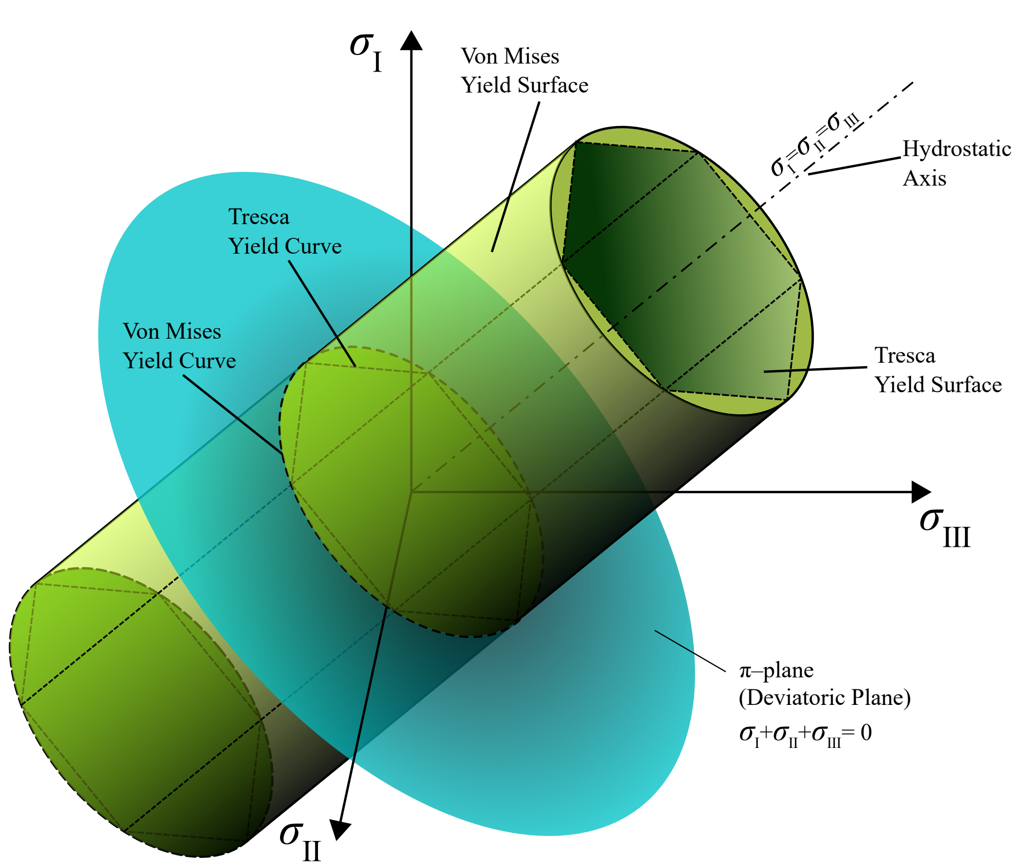
$$\frac{(\sigma_1 - \sigma_2)^2 + (\sigma_2 - \sigma_3)^2 + (\sigma_3 - \sigma_1)^2}{2} \geq \sigma_f^2$$

The left side of the equation when (sigma_f) is solved is the von Mises stress:

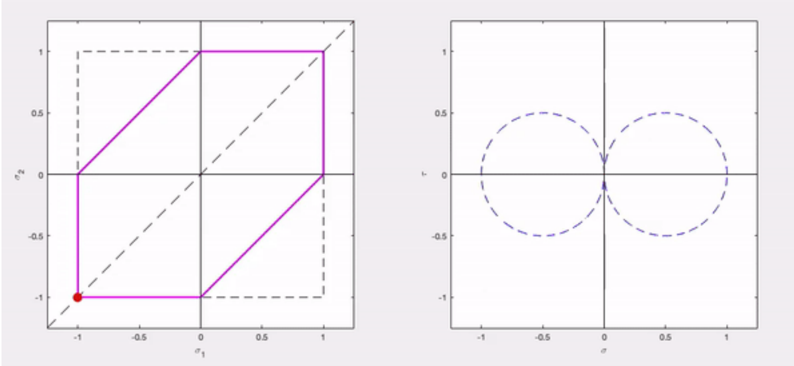
$$\sigma_{vm} = \sqrt{\frac{(\sigma_1 - \sigma_2)^2 + (\sigma_2 - \sigma_3)^2 + (\sigma_3 - \sigma_1)^2}{2}}$$

The yield and failure surface can be plotted, obtaining an infinite cylinder. According to the theory all stress points that lie within the surface do not produce failure and those outside do.





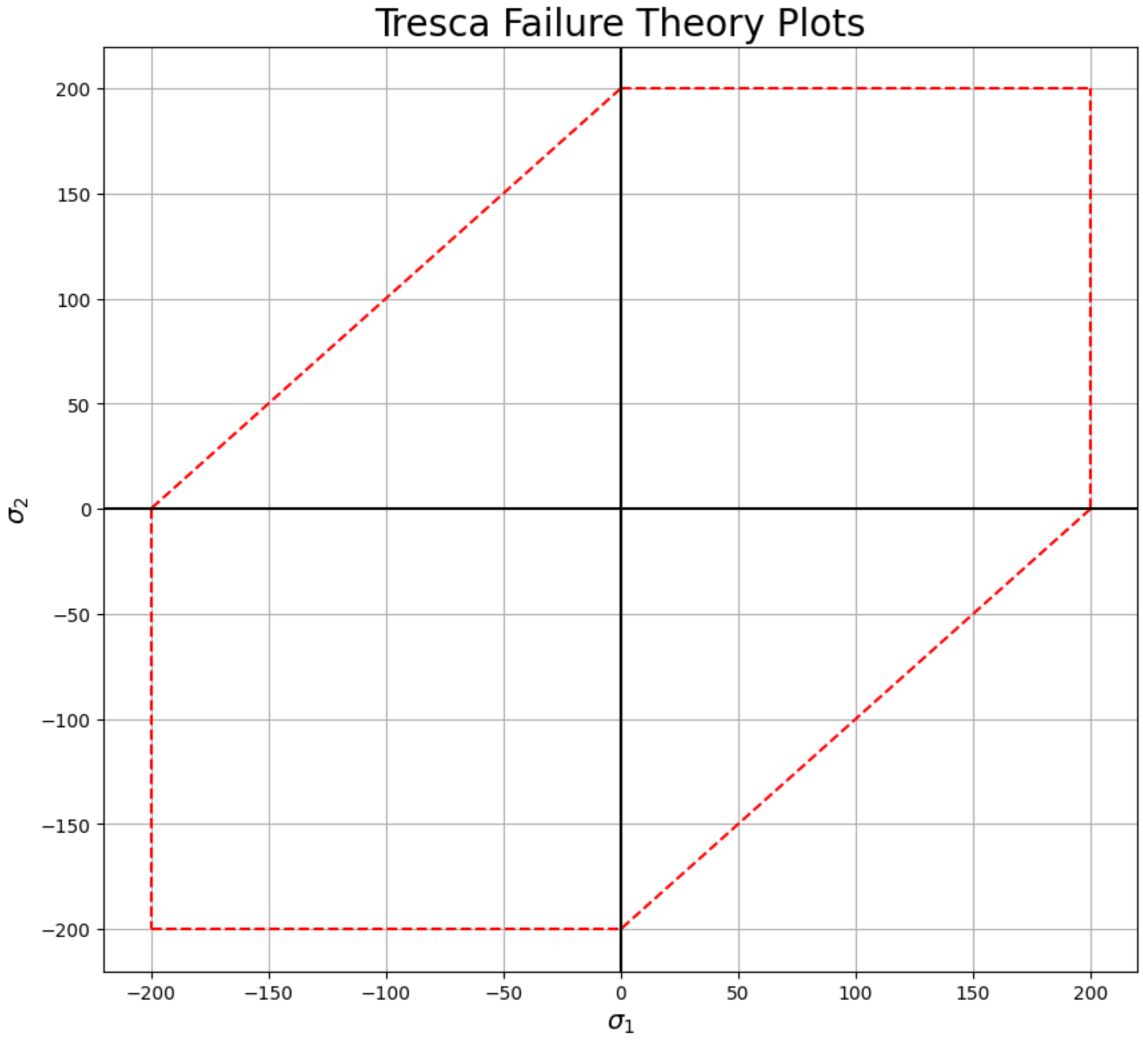
(ductile - can irgnore hydrostatic?)

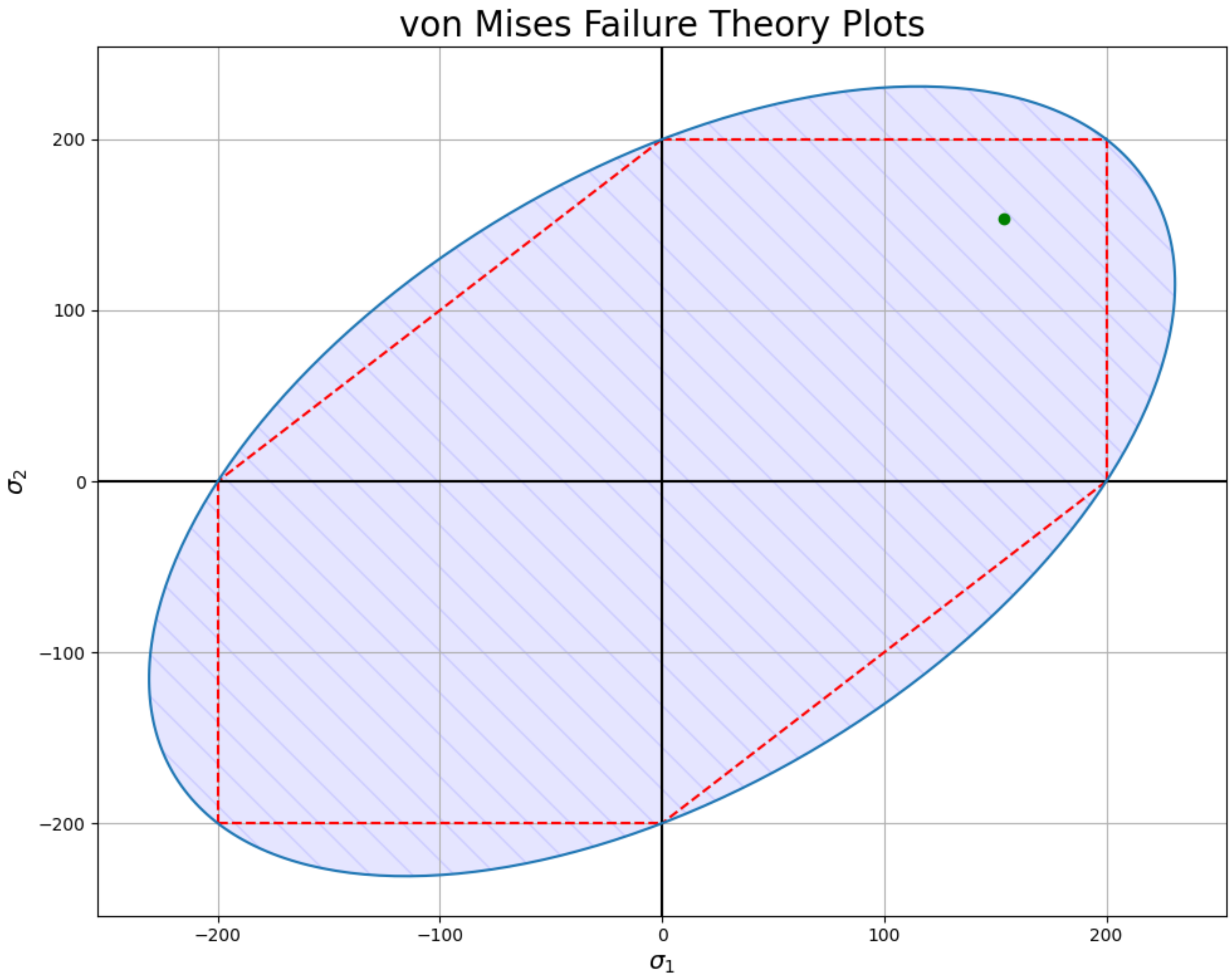


2D Tresca plot

In [ ]:

```python
import numpy as np
import matplotlib.pyplot as plt

sy = 200  # yeild strength [MPa]
s1 = 125  # principal stress 1, s1 [MPa]
s2 = 90   # principal stress 2, s2 [MPa]

class MaterialFailure():
    def __init__(self,sy,s1,s2):
        ''' INPUT INITIAL INFORMATION
                σ_y : sigma y
                σ_1 : sigma 1
                σ_2 : sigma 2
        '''
        self.sy = sy
        self.s1 = s1
        self.s2 = s2

    def TrescaCriterionPlot(self):
        plt.figure(figsize=(10,9))
        plt.title("Tresca Failure Theory Plots",fontsize=20)
        plt.xlabel(r'$\sigma_1$',fontsize=14)
        plt.ylabel(r'$\sigma_2$',fontsize=14)
        # plot dashed line for maximum shear stress criterion (Tresca yield criterion)
        plt.plot([self.sy,self.sy],[0,sy],'r--')
        plt.plot([0,self.sy],[self.sy,self.sy],'r--')
        plt.plot([-self.sy,0],[-self.sy,-self.sy],'r--')
        plt.plot([-self.sy,-self.sy],[0,-self.sy],'r--')
        plt.plot([0,-self.sy],[self.sy,0],'r--')
        plt.plot([self.sy,0],[0,-self.sy],'r--')

        plt.axhline(color = 'k')
        plt.axvline(color = 'k')
        plt.grid()
        plt.show()

    def vonMisesCriterionPlot(self):
        s_von = np.sqrt(self.s1**2 + self.s2**2 -(self.s1-self.s2))

        a = np.sqrt(2)*self.sy
        b = np.sqrt(2/3)*self.sy

        alpha = np.linspace(0,2*np.pi,360)
        theta = np.pi/4 # 45degree

        #--before rotation matrix--#
        # x = a*np.cos(alpha)
        # y = b*np.sin(alpha)

        #--after rotation matrix--#
        x = (a*np.cos(alpha)*np.cos(theta)) - (b*np.sin(alpha)*np.sin(theta))
        y = (a*np.cos(alpha)*np.sin(theta)) + (b*np.sin(alpha)*np.cos(theta))

        # set graph size
        plt.figure(figsize=(10,8))
        plt.title("von Mises Failure Theory Plots",fontsize=20)
        plt.xlabel(r'$\sigma_1$',fontsize=14)
        plt.ylabel(r'$\sigma_2$',fontsize=14)

        plt.fill_between(x,y,color='blue',hatch='\\',alpha=0.1)

        # plot dashed line for maximum shear stress criterion (Tresca yield criterion)
        plt.plot([self.sy,self.sy],[0,sy],'r--')
        plt.plot([0,self.sy],[self.sy,self.sy],'r--')
        plt.plot([-self.sy,0],[-self.sy,-self.sy],'r--')
        plt.plot([-self.sy,-self.sy],[0,-self.sy],'r--')
        plt.plot([0,-self.sy],[self.sy,0],'r--')
        plt.plot([self.sy,0],[0,-self.sy],'r--')

        # Create horizontal and vertical lines at center
        plt.axhline(color = 'k')
        plt.axvline(color = 'k')

        # check if von Mises stress in the region
        plt.scatter(s_von,s_von,color = 'g')
        # show gridline
        plt.grid()
        plt.tight_layout()
        plt.plot(x,y)
        plt.show()

if __name__ == '__main__':
    x = MaterialFailure(sy,s1,s2)
    x.TrescaCriterionPlot()
    x.vonMisesCriterionPlot()
```





[stress tensor] = [hydrostatic] + [deviatoric]

In [ ]:
```
import numpy as np

def calculate_tensors(stress_tensor):
    # Ensure the input is a numpy array
    stress_tensor = np.array(stress_tensor)
    # Calculate the hydrostatic tensor
    hydrostatic_tensor = np.trace(stress_tensor) / 3 * np.eye(3)
    # Calculate the deviatoric tensor
    deviatoric_tensor = stress_tensor - hydrostatic_tensor
    return hydrostatic_tensor, deviatoric_tensor

#stress_tensor = [[96, 0, 0], [0, -76, 0], [0, 0, 100]]  #INPUT
# stress_tensor = [[10, 20, 30], [20, 40, 50], [30, 50, 60]] #INPUT
# stress_tensor = [[10, 20, 30], [20, 40, 50], [30, 50, 60]] #INPUT
stress_tensor = [[40, 15, 35], [15, 30,25], [35, 25, 20]] #INPUT
hydrostatic_tensor, deviatoric_tensor = calculate_tensors(stress_tensor)

print("Stress tensor:")
print(stress_tensor)
print("\nHydrostatic Tensor:")
print(hydrostatic_tensor)
print("\nDeviatoric Tensor:")
print(deviatoric_tensor)
```

```
Stress tensor:
[[40, 15, 35], [15, 30, 25], [35, 25, 20]]

Hydrostatic Tensor:
[[30.  0.  0.]
 [ 0. 30.  0.]
 [ 0.  0. 30.]]

Deviatoric Tensor:
[[ 10.  15.  35.]
 [ 15.   0.  25.]
 [ 35.  25. -10.]]
```

this one isn't working, but it worked at some point and it uses the dot product

In [ ]:
```
#create first matrix
#A = np.array([[96, 0, 0], [0, -76, 0], [0, 0, 100]])
#A = np.array([[400, 0, 0], [0, 380, 0], [0, 0, 350]])
A = np.array([[40, 15, 35], [15, 30,25], [35, 25, 20]])
print(" A :")
print(A)
```

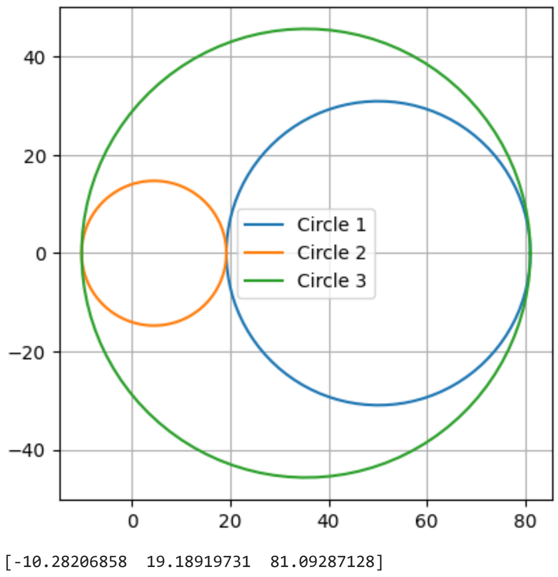
```python
print(" ")

#hydrostatic
B = np.sum(A)/-8
BB= np.dot(B,np.identity(3))
#create second matrix
#B = np.array([[5,6,1],[7,8,1]])
print("Hydrostatic B :")
print(BB)
print(" ")

# adding two matrix
print('A - I B')
C = np.add(A,BB)
print(C)
```

```
 A :
[[40 15 35]
 [15 30 25]
 [35 25 20]]

Hydrostatic B :
[[-30.   0.   0.]
 [  0. -30.   0.]
 [  0.   0. -30.]]

A - I B
[[ 10.  15.  35.]
 [ 15.   0.  25.]
 [ 35.  25. -10.]]
```

3D Morh's Circle Plot

In [ ]:

```python
# @title
import numpy as np
import matplotlib.pyplot as plt

def mohrs_circle_3d(stress_tensor):
    # Calculate the principal stresses
    principal_stresses = np.linalg.eigvalsh(stress_tensor)
    s1, s2, s3 = np.sort(principal_stresses)[::-1]

    # Calculate the center and radius of the three Mohr's Circles
    center1 = (s1 + s2) / 2
    radius1 = (s1 - s2) / 2
    center2 = (s2 + s3) / 2
    radius2 = (s2 - s3) / 2
    center3 = (s1 + s3) / 2
    radius3 = (s1 - s3) / 2

    # Generate points on the three Mohr's Circles
    theta = np.linspace(0, 2 * np.pi, 100)
    x1 = center1 + radius1 * np.cos(theta)
    y1 = radius1 * np.sin(theta)
    x2 = center2 + radius2 * np.cos(theta)
    y2 = radius2 * np.sin(theta)
    x3 = center3 + radius3 * np.cos(theta)
    y3 = radius3 * np.sin(theta)

    # Plot the three Mohr's Circles
    fig, ax = plt.subplots()
    ax.plot(x1, y1, label='Circle 1')
    ax.plot(x2, y2, label='Circle 2')
    ax.plot(x3, y3, label='Circle 3')
    ax.legend()
    ax.set_aspect('equal')
    ax.grid(True)
    plt.show()

    return principal_stresses

#stress_tensor = np.array([[14,8,6],[8,12,-10],[6,-10,10]])
#stress_tensor = np.array([[100, -50, 0], [-50, -100, 0], [0, 0, -150]])
#stress_tensor = np.array([[96, 0, 0], [0, -76, 0], [0, 0, 100]])
stress_tensor = np.array([[40, 15, 35], [15, 30,25], [35, 25, 20]])  #EXAMPLE XXXXXXXXXXXXX
#stress_tensor = np.array([[400, 0, 0], [0, 300, 0], [0, 0, 150]])
#stress_tensor = np.array([[80, 25, 0], [25, -50, 0], [0, 0, 0]])      #PLANAR


#mohrs_circle_3d(stress_tensor)
principal_stresses=mohrs_circle_3d(stress_tensor)
print(principal_stresses)
```




```
[-10.28206858  19.18919731  81.09287128]
```

3D calculator with `linalg.eigvalsh(stress_tensor)`

In [ ]:

```python
import numpy as np

def principal_stresses(stress_tensor):
    principal_stresses = np.linalg.eigvalsh(stress_tensor)
    # Sort the principal stresses in descending order
    principal_stresses = np.sort(principal_stresses)[::-1]
    return principal_stresses

# Example usage:
#stress_tensor = np.array([[100, -50, 0], [-50, -100, 0], [0, 0, -150]])
#stress_tensor = np.array([[400, 0, 0], [0, 380, 0], [0, 0, 350]])
#stress_tensor = np.array([[40, 15, 35], [15, 30,25], [35, 25, 20]])
#stress_tensor = np.array([[50, 20, 0], [20, -30, 0], [0, 0, 40]])
#stress_tensor = np.array([[14,8,6],[8,12,-10],[6,-10,10]])
stress_tensor = np.array([[80, 25, 0.0], [25, -50, 0], [0, 0, 0]])  #PLANAR
print(stress_tensor)
print("gives principle stresses:", principal_stresses(stress_tensor))
```

```
[[ 80.  25.   0.]
 [ 25. -50.   0.]
 [  0.   0.   0.]]
gives principle stresses: [ 84.6419   0.      -54.6419]
```

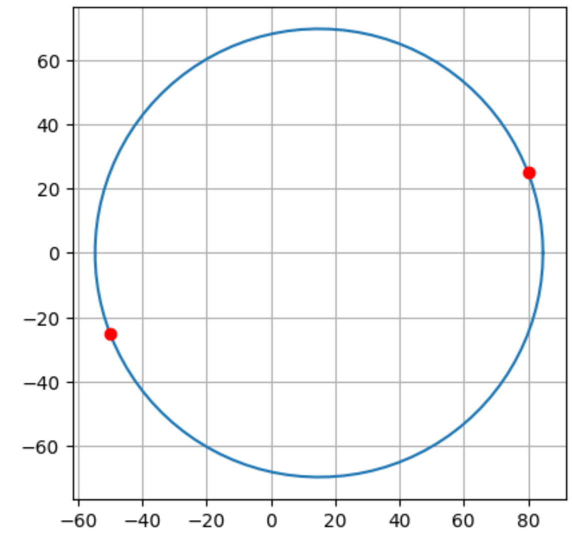
2D Mohr's Circle Plot

plane stress

```
In [ ]: #Given
sigma_x = 80  #ksi, stress along x
sigma_y = -50  #stress along y
tou_xy = 25    #ksi, stress along xy

#Calculation Construction of the circle
import math
sigma_avg = (sigma_x+sigma_y)/2.0
R = math.sqrt((-sigma_x+sigma_avg)**2 + (tou_xy)**2)
#Principal Stresses
sigma2 = -R+sigma_avg
sigma1 = R+sigma_avg
theta_p2 = math.atan((-tou_xy)/(-sigma_x+sigma_avg))
theta_p2 = theta_p2/2*(180/math.pi)

#Display
print('The first principal stress is            = ',round(sigma1,2),"ksi")
print('The second principal stress is           = ',round(sigma2,2),"ksi")
print('The direction of the principal plane is  = ',theta_p2,"degree")
```

```
The first principal stress is            =  84.64 ksi
The second principal stress is           =  -54.64 ksi
The direction of the principal plane is  =  10.518755512710909 degree
```

```
In [ ]: # @title
import numpy as np
import matplotlib.pyplot as plt

def mohrs_circle(sigma_x, sigma_y, tau_xy):
    # Calculate the center and radius of Mohr's Circle
    center = (sigma_x + sigma_y) / 2
    radius = np.sqrt(((sigma_x - sigma_y) / 2)**2 + tau_xy**2)

    # Generate points on Mohr's Circle
    theta = np.linspace(0, 2 * np.pi, 100)
    x = center + radius * np.cos(theta)
    y = radius * np.sin(theta)

    # Plot Mohr's Circle
    fig, ax = plt.subplots()
    ax.plot(x, y)

    # Plot the input values as X and Y points on the circle
    ax.plot([sigma_x, sigma_y], [tau_xy, -tau_xy], 'ro')

    ax.set_aspect('equal')
    ax.grid(True)
    plt.show()

# Example usage:
mohrs_circle(80, -50, 25)    ##INPUT~~~~~~~~~~~~~~~~~~~~
```



3D number solver

```
In [ ]: import numpy as np
import matplotlib.pyplot as plt

def principal_stresses(sigma_x, sigma_y, sigma_z, tau_xy, tau_yz, tau_zx):
    # Calculate the principal stresses
    A = 1
    B = -(sigma_x + sigma_y + sigma_z)
    C = (sigma_x * sigma_y) + (sigma_y * sigma_z) + (sigma_z * sigma_x) - (tau_xy ** 2) - (tau_yz ** 2) - (tau_zx ** 2)
    D = -(sigma_x * sigma_y * sigma_z) + (sigma_x * tau_yz ** 2) + (sigma_y * tau_zx ** 2) + (sigma_z * tau_xy ** 2) - (2 * tau_xy * tau_yz * tau_zx)
    roots = np.roots([A, B, C, D])
    return roots

def von_mises(sigma_x, sigma_y, sigma_z, tau_xy, tau_yz, tau_zx):
    # Calculate the von Mises distortion energy theory
    s1, s2, s3 = principal_stresses(sigma_x, sigma_y, sigma_z, tau_xy, tau_yz, tau_zx)
    return np.sqrt((s1 - s2)**2 + (s2 - s3)**2 + (s3 - s1)**2 + 6*(tau_xy**2 + tau_yz**2 + tau_zx**2))

# Take input for triaxial stress values
sigma_x = float(input("Enter the value of Sigma X: "))
sigma_y = float(input("Enter the value of Sigma Y: "))
sigma_z = float(input("Enter the value of Sigma Z: "))
tau_xy = float(input("Enter the value of Tau XY: "))
tau_yz = float(input("Enter the value of Tau YZ: "))
tau_zx = float(input("Enter the value of Tau ZX: "))

# Calculate principal stresses and von Mises distortion energy theory
s1, s2, s3 = principal_stresses(sigma_x, sigma_y, sigma_z, tau_xy, tau_yz, tau_zx)
vm_stress = von_mises(sigma_x, sigma_y, sigma_z, tau_xy, tau_yz, tau_zx)

# Print results
print(f"\nPrincipal Stresses:\nS1: {s1:.4f}\nS2: {s2:.4f}\nS3: {s3:.4f}")
print(f"\nVon Mises Distortion Energy Theory:\n{vm_stress:.4f}")
```

```
Enter the value of Sigma X: 80
Enter the value of Sigma Y: -50
Enter the value of Sigma Z: 0.01
Enter the value of Tau XY: 25
Enter the value of Tau YZ: 0
Enter the value of Tau ZX: 0

Principal Stresses:
S1: 84.6419
S2: -54.6419
S3: 0.0100

Von Mises Distortion Energy Theory:
182.4812
```

3D solver by robsiegwart

```
In [ ]: import numpy as np

# Set the print options to 4 decimal places
np.set_printoptions(precision=4)

S_xx = 80
S_yy = -50
S_zz = 0.01
S_xy = 25
S_zx = 0
S_yz = 0
```

```
S = np.array([ [S_xx, S_xy, S_zx],
               [S_xy, S_yy, S_yz],
               [S_zx, S_yz, S_zz] ])
```

In [ ]:
```
e_val, e_vec = np.linalg.eig(S)
print(str(e_val) + '\n' + '\n' + str(e_vec))
```

```
[ 8.4642e+01 -5.4642e+01  1.0000e-02]

[[ 0.9832 -0.1826  0.    ]
 [ 0.1826  0.9832  0.    ]
 [ 0.      0.      1.    ]]
```

In [ ]:
```
p3, p2, p1 = np.sort(e_val)   # sort smallest to largest
print(f"\nPrincipal Stresses:\nS1: {p1:.4f}\nS2: {p2:.4f}\nS3: {p3:.4f}")
```

```
Principal Stresses:
S1: 84.6419
S2: 0.0100
S3: -54.6419
```

In [ ]:
```
tau1 = (p1-p3)/2
tau2 = (p1-p2)/2
tau3 = (p2-p3)/2
print(f"\n tau1", tau1)
print(f"\n tau2", tau2)
print(f"\n tau3", tau3)
```

```
 tau1 69.64194138592059

 tau2 42.315970692960285

 tau3 27.325970692960297
```

References

https://www.robsiegwart.com/principal-stresses-in-3D.html

https://www.robsiegwart.com/failure-theories.html

https://www.purdue.edu/freeform/me323/animations-and-demonstrations/failure-boundaries-and-mohrs-circle/

https://pantelisliolios.com/principal-stresses-and-invariants/

https://www.doitpoms.ac.uk/tlplib/metal-forming-1/printall.php

---

ARCHIVE

3d Interactive plot of 2 Vectors

using `plotly` to make an interactive 3d graph

In [1]:
```
import plotly.graph_objs as go

# Define the vectors
V1 = [1, 2, 3]
V2 = [3, 1, 2]

# Create the plot
fig = go.Figure()

# Add the vectors to the plot
fig.add_trace(go.Scatter3d(x=[0, V1[0]], y=[0, V1[1]], z=[0, V1[2]], mode='lines', line=dict(width=5, color='red')))
fig.add_trace(go.Scatter3d(x=[0, V2[0]], y=[0, V2[1]], z=[0, V2[2]], mode='lines', line=dict(width=5, color='blue')))

# Add the origin and axes to the plot
fig.add_trace(go.Scatter3d(x=[0], y=[0], z=[0], mode='markers', marker=dict(size=5, color='black')))
fig.add_trace(go.Scatter3d(x=[0, 5], y=[0, 0], z=[0, 0], mode='lines', line=dict(width=2, color='black')))
fig.add_trace(go.Scatter3d(x=[0, 0], y=[0, 5], z=[0, 0], mode='lines', line=dict(width=2, color='black')))
fig.add_trace(go.Scatter3d(x=[0, 0], y=[0, 0], z=[0, 5], mode='lines', line=dict(width=2, color='black')))

# Show the plot
fig.show()
```

Eigenvalues and eigenvectors of stiffness matrices

has some advanced derivations and interesting approaches

Predefinition

In [ ]:
```
from sympy.utilities.codegen import codegen
from sympy import *
from sympy import init_printing
```

In [ ]:
```
init_printing()
```

In [ ]:
```
r, s, t, x, y, z = symbols('r s t x y z')
k, m, n = symbols('k m n', integer=True)
rho, nu, E = symbols('rho, nu, E')
```

The constitutive model tensor in Voigt notation (plane stress) is

$$C = \frac{E}{(1-\nu^2)}\begin{pmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2)} \end{pmatrix}$$

In [ ]:

```python
K_factor = E/(1 - nu**2)
C = K_factor * Matrix([
        [1, nu, 0],
        [nu, 1, 0],
         [0, 0, (1 - nu)/2]])

C
```

Out[ ]:

$$\begin{bmatrix} \frac{E}{1-\nu^2} & \frac{E\nu}{1-\nu^2} & 0 \\ \frac{E\nu}{1-\nu^2} & \frac{E}{1-\nu^2} & 0 \\ 0 & 0 & \frac{E\left(\frac{1}{2}-\frac{\nu}{2}\right)}{1-\nu^2} \end{bmatrix}$$

Interpolation functions

The shape functions are

In [ ]:

```python
N = S(1)/4*Matrix([(1 + r)*(1 + s),
         (1 - r)*(1 + s),
         (1 - r)*(1 - s),
         (1 + r)*(1 - s)])
N
```

Out[ ]:

$$\begin{bmatrix} \frac{(r+1)(s+1)}{4} \\ \frac{(1-r)(s+1)}{4} \\ \frac{(1-r)(1-s)}{4} \\ \frac{(1-s)(r+1)}{4} \end{bmatrix}$$

Thus, the interpolation matrix renders

Derivatives interpolation matrix

In [ ]:

```python
dHdr = zeros(2,4)
for i in range(4):
    dHdr[0,i] = diff(N[i],r)
    dHdr[1,i] = diff(N[i],s)

jaco = eye(2)  # Jacobian matrix, identity for now
dHdx = jaco*dHdr

B = zeros(3,8)
for i in range(4):
    B[0, 2*i] = dHdx[0, i]
    B[1, 2*i+1] = dHdx[1, i]
    B[2, 2*i] = dHdx[1, i]
    B[2, 2*i+1] = dHdx[0, i]

B
```

Out[ ]:

$$\begin{bmatrix} \frac{s}{4}+\frac{1}{4} & 0 & -\frac{s}{4}-\frac{1}{4} & 0 & \frac{s}{4}-\frac{1}{4} & 0 & \frac{1}{4}-\frac{s}{4} & 0 \\ 0 & \frac{r}{4}+\frac{1}{4} & 0 & \frac{1}{4}-\frac{r}{4} & 0 & \frac{r}{4}-\frac{1}{4} & 0 & -\frac{r}{4}-\frac{1}{4} \\ \frac{r}{4}+\frac{1}{4} & \frac{s}{4}+\frac{1}{4} & \frac{1}{4}-\frac{r}{4} & -\frac{s}{4}-\frac{1}{4} & \frac{r}{4}-\frac{1}{4} & \frac{s}{4}-\frac{1}{4} & -\frac{r}{4}-\frac{1}{4} & \frac{1}{4}-\frac{s}{4} \end{bmatrix}$$

Being the stiffness matrix integrand

$$K_{\text{int}} = B^T C B$$

In [ ]:

```python
K_int = B.T*C*B
```

Analytic integration

The stiffness matrix is obtained integrating the product of the interpolator-derivatives (displacement-to-strains) matrix with the constitutive tensor and itself, i.e.

$$\begin{aligned} K &= \int_{-1}^{1}\int_{-1}^{1} K_{\text{int}} dr\, ds \\ &= \int_{-1}^{1}\int_{-1}^{1} B^T C\, B\, dr\, ds\ . \end{aligned}$$

In [ ]:

```python
K = zeros(8,8)
for i in range(8):
    for j in range(8):
        K[i,j] = integrate(K_int[i,j], (r,-1,1), (s,-1,1))

simplify(K/K_factor)
```

Out[ ]:

$$\begin{bmatrix} \frac{1}{2}-\frac{\nu}{6} & \frac{\nu}{8}+\frac{1}{8} & -\frac{\nu}{12}-\frac{1}{4} & \frac{3\nu}{8}-\frac{1}{8} & \frac{\nu}{12}-\frac{1}{4} & -\frac{\nu}{8}-\frac{1}{8} & \frac{\nu}{6} & \frac{1}{8}-\frac{3\nu}{8} \\ \frac{\nu}{8}+\frac{1}{8} & \frac{1}{2}-\frac{\nu}{6} & \frac{1}{8}-\frac{3\nu}{8} & \frac{\nu}{6} & -\frac{\nu}{8}-\frac{1}{8} & \frac{\nu}{12}-\frac{1}{4} & \frac{3\nu}{8}-\frac{1}{8} & -\frac{\nu}{12}-\frac{1}{4} \\ -\frac{\nu}{12}-\frac{1}{4} & \frac{1}{8}-\frac{3\nu}{8} & \frac{1}{2}-\frac{\nu}{6} & -\frac{\nu}{8}-\frac{1}{8} & \frac{\nu}{6} & \frac{3\nu}{8}-\frac{1}{8} & \frac{\nu}{12}-\frac{1}{4} & \frac{\nu}{8}+\frac{1}{8} \\ \frac{3\nu}{8}-\frac{1}{8} & \frac{\nu}{6} & -\frac{\nu}{8}-\frac{1}{8} & \frac{1}{2}-\frac{\nu}{6} & \frac{1}{8}-\frac{3\nu}{8} & -\frac{\nu}{12}-\frac{1}{4} & \frac{\nu}{8}+\frac{1}{8} & \frac{\nu}{12}-\frac{1}{4} \\ \frac{\nu}{12}-\frac{1}{4} & -\frac{\nu}{8}-\frac{1}{8} & \frac{\nu}{6} & \frac{1}{8}-\frac{3\nu}{8} & \frac{1}{2}-\frac{\nu}{6} & \frac{\nu}{8}+\frac{1}{8} & -\frac{\nu}{12}-\frac{1}{4} & \frac{3\nu}{8}-\frac{1}{8} \\ -\frac{\nu}{8}-\frac{1}{8} & \frac{\nu}{12}-\frac{1}{4} & \frac{3\nu}{8}-\frac{1}{8} & -\frac{\nu}{12}-\frac{1}{4} & \frac{\nu}{8}+\frac{1}{8} & \frac{1}{2}-\frac{\nu}{6} & \frac{1}{8}-\frac{3\nu}{8} & \frac{\nu}{6} \\ \frac{\nu}{6} & \frac{3\nu}{8}-\frac{1}{8} & \frac{\nu}{12}-\frac{1}{4} & \frac{\nu}{8}+\frac{1}{8} & -\frac{\nu}{12}-\frac{1}{4} & \frac{1}{8}-\frac{3\nu}{8} & \frac{1}{2}-\frac{\nu}{6} & -\frac{\nu}{8}-\frac{1}{8} \\ \frac{1}{8}-\frac{3\nu}{8} & -\frac{\nu}{12}-\frac{1}{4} & \frac{\nu}{8}+\frac{1}{8} & \frac{\nu}{12}-\frac{1}{4} & \frac{3\nu}{8}-\frac{1}{8} & \frac{\nu}{6} & -\frac{\nu}{8}-\frac{1}{8} & \frac{1}{2}-\frac{\nu}{6} \end{bmatrix}$$

We can check some numerical vales for $E = 1$ Pa and $\nu = 1/3$

In [ ]:

```python
K_num = K.subs([(E, 1), (nu, S(1)/3)])
```

In [ ]:

```python
K_num.eigenvects()
```

Out[ ]:

$$\left[\left(0,\ 3,\ \left[\begin{bmatrix}1\\0\\1\\1\\0\\1\\0\\0\end{bmatrix},\ \begin{bmatrix}1\\0\\1\\0\\1\\0\\1\\0\end{bmatrix},\ \begin{bmatrix}-1\\1\\-1\\0\\0\\0\\0\\1\end{bmatrix} \right] \right),\ \left(\frac{1}{2},\ 2,\ \left[\begin{bmatrix}-1\\0\\1\\0\\-1\\0\\1\\0\end{bmatrix},\ \begin{bmatrix}0\\-1\\0\\1\\0\\-1\\0\\1\end{bmatrix} \right] \right),\ \left(\frac{3}{4},\ 2,\ \left[\begin{bmatrix}0\\-1\\-1\\0\\0\\1\\1\\0\end{bmatrix},\ \begin{bmatrix}1\\0\\0\\-1\\-1\\0\\0\\1\end{bmatrix} \right] \right),\ \left(\frac{3}{2},\ 1,\ \left[\begin{bmatrix}-1\\-1\\1\\-1\\1\\1\\-1\\1\end{bmatrix} \right] \right) \right]$$