pyLabFEA-Supplementary

March 24, 2020

1 Machine Learning flow rules

This notebook represents supplementary material to the publication "Data-Oriented Constitutive Modeling of Plasticity in Metals" by Alexander Hartmaier in Materials (2020). It demonstrates the training and application of ML flow rules in FEA in form of simple examples with data synthetically produced from standard flow rules, like isotropic J2 (von Mises) or Hill-type anisotropic plasticity and also with Tresca yield functions. FEA is performed with the pyLabFEA package, which - together with further tutorials and examples - is freely available on GitHub (https://github.com/AHartmaier/pyLabFEA.git). This notebook uses the matplotlib (https://matplotlib.org/) library for the visualization of results, and NumPy (http://www.numpy.org) for mathematical operations with arrays. Machine learning algorithms have been adopted from the scikit-learn platform (https://scikit-learn.org/stable/).

Author: Alexander Hartmaier, ICAMS, Ruhr-Universtität Bochum, Germany, March 2020

1.1 Introduction

Machine Learning (ML) algorithms provide a great flexibility to describe aribitrary mathematical functions. At the same time they offer the possibility to handle large data sets and multidimensional features as input. Hence, using ML algorithms as constitutive rules for plastic material behavior offers the possibility to explicitly take into account microstructural information of the material in the constitutive modeling. Furthermore, data resulting from experiment and micromechanical simulations can be hybridized to generate training data sets. The present example is using Support Vector Classification (SVC) as yield function. The SVC algorithm is trained by using deviatoric stresses as input data and the information whether a given stress state leads to purely elastic or rather to elastic-plastic deformation of the material as result data. In this way, a ML yield function is obtained, which can determine whether a given stress state lies inside or outside of the elastic regime of the material. Furthermore, the yield locus, i.e., the hyperplane in stress space on which plastic deformation occurs, can be reconstructed from the SVC, and the gradient on this yield locus can be conveniently calculated. Therefore, the standard formulations of continuum plasticity, as the return mapping algorithm, can be applied in Finite Element Analysis (FEA) in the usual way. Thus, it is demonstrated that the new ML yield function can replace conventional FEA yield functions.

1.2 Theoretical background

The yield function of a material is defined as

$$f = \sigma_{eq} - \sigma_y \,, \tag{1}$$

where plastic deformation sets in at $f = 0$, i.e. when the the equivalent stress σ_{eq} equals the yield strength σ_y of the material. The equivalent stress used here is based on the principal stresses σ_i with $(i = 1, 2, 3)$, as

$$\sigma_{eq}^{\mathrm{J2}} = \sqrt{\frac{1}{2}\left[H_1 \left(\sigma_1 - \sigma_2\right)^2 + H_2 \left(\sigma_2 - \sigma_3\right)^2 + H_3 \left(\sigma_3 - \sigma_1\right)^2 \right]}\,. \tag{2}$$

In this yield function, the anisotropy of the material's flow behavior is described in a Hill-like approach for orthotropic materials. If the axes of the prical stresses do not match the symmetry axes of the material, the material axes and with it the parameters H_1, H_2 and H_3 must be rotated into the coordinate system of the eigenvectors of the stress tensor.

The gradient of the yield function with respect to the prical stresses is needed for calculating the plastic strain increments in the return mapping algorithm of continuum plasticity, and can be evaluated analytically as

$$\frac{\partial f}{\partial \sigma_1} = \frac{\partial \sigma_{eq}}{\partial \sigma_1} = \frac{\left(H_1 + H_3\right)\sigma_1 - H_1\sigma_2 - H_3\sigma_3}{\sigma_{eq}} \tag{3}$$

$$\frac{\partial f}{\partial \sigma_2} = \frac{\partial \sigma_{eq}}{\partial \sigma_2} = \frac{\left(H_2 + H_1\right)\sigma_2 - H_1\sigma_1 - H_2\sigma_3}{\sigma_{eq}} \tag{4}$$

$$\frac{\partial f}{\partial \sigma_3} = \frac{\partial \sigma_{eq}}{\partial \sigma_3} = \frac{\left(H_3 + H_2\right)\sigma_3 - H_3\sigma_1 - H_2\sigma_2}{\sigma_{eq}} \tag{5}$$

Note that in the case of isotropic plasticity, i.e. $H_1 = H_2 = H_3 = 1$, the gradient takes the simple form

$$\frac{\partial f}{\partial \sigma_i} = 3\frac{\sigma_i - p}{\sigma_{eq}} \qquad (i = 1, 2, 3), \tag{6}$$

where $p = 1/3\mathrm{Tr}(\sigma)$.

1.3 Deviatoric stress space

Since plastic deformation in most metals does not depend on hydrostatic stress components, it is useful to transform the principal stresses from the representation as Cartesian (3D) vector $\sigma = (\sigma_1, \sigma_2, \sigma_3)$ in the principal stress space into a vector $s = (\sigma_{eq}, \theta, p)$ in the cylindrical coordinate system, with the equivalent stress σ_{eq} representing the norm of the deviator of σ and the polar angle θ lying in the deviatoric stress plane normal to the hydrostatic axis p. This coordinate transformation improves the efficiency of the training, because only 2D data for the equivalent stress and the polar angle θ need to be used as training features, whereas the hydrostatic component is disregarded. The coordinate transformation is performed by introducing a complex-valued deviatoric stress

$$\sigma_c' = \boldsymbol{\sigma} \cdot \boldsymbol{a} + i\boldsymbol{\sigma} \cdot \boldsymbol{b} = \sqrt{2/3}\sigma_{eq}\, e^{i\theta}, \tag{7}$$

where i is the imaginary unit, such that the polar angle is obtained as

$$\theta = \arg \sigma'_c = -i \ln \frac{\boldsymbol{\sigma} \cdot \boldsymbol{a} + i \boldsymbol{\sigma} \cdot \boldsymbol{b}}{\sqrt{2/3}\sigma_{eq}}, \tag{8}$$

where $a = (2, -1, -1)/\sqrt{6}$ (real axis) and $b = (0, 1, -1)/\sqrt{2}$ (imaginary axis) are the unit vectors that span the deviatoric stress plane normal to the hydrostatic axis $c = (1, 1, 1)/\sqrt{3}$.

An advantage of this coordinate transformation is that the gradient of the yield function w.r.t. the cylindrical coordinates has only one non-constant component. The complete gradient w.r.t. the cylindrical coordinates reads

$$\begin{aligned}
\frac{\partial f}{\partial \sigma_{eq}} &= 1 \\
\frac{\partial f}{\partial \theta} &= \frac{\partial \sigma_y}{\partial \theta} \\
\frac{\partial f}{\partial p} &= 0.
\end{aligned}$$

To transform the gradient of the flow rule from this cylindrical coordinate system back to the principle stress space, in which form it is used later on to calculate the direction of the plastic strain increments in the return mapping algorithm of the plasticity model, we introduce the Jacobian matrix for this coordinate transformation as

$$J = \frac{\partial s}{\partial \sigma} = \begin{pmatrix} \frac{\partial \sigma_{eq}}{\partial \sigma_1} & \frac{\partial \theta}{\partial \sigma_1} & \frac{\partial p}{\partial \sigma_1} \\ \frac{\partial \sigma_{eq}}{\partial \sigma_2} & \frac{\partial \theta}{\partial \sigma_2} & \frac{\partial p}{\partial \sigma_2} \\ \frac{\partial \sigma_{eq}}{\partial \sigma_3} & \frac{\partial \theta}{\partial \sigma_3} & \frac{\partial p}{\partial \sigma_3} \end{pmatrix} \tag{9}$$

with $\partial \sigma_{eq}/\partial \sigma_i$ as given above, $\partial p/\partial \sigma_i = 1/3$, and

$$\frac{\partial \theta}{\partial \sigma_j} = -i \left(\frac{\boldsymbol{a} + i \boldsymbol{b}}{\boldsymbol{\sigma} \cdot \boldsymbol{a} + i \boldsymbol{\sigma} \cdot \boldsymbol{b}} - \frac{3\boldsymbol{\sigma}'}{\sigma_{eq}^2} \right) \qquad (j = 1, 2, 3). \tag{10}$$

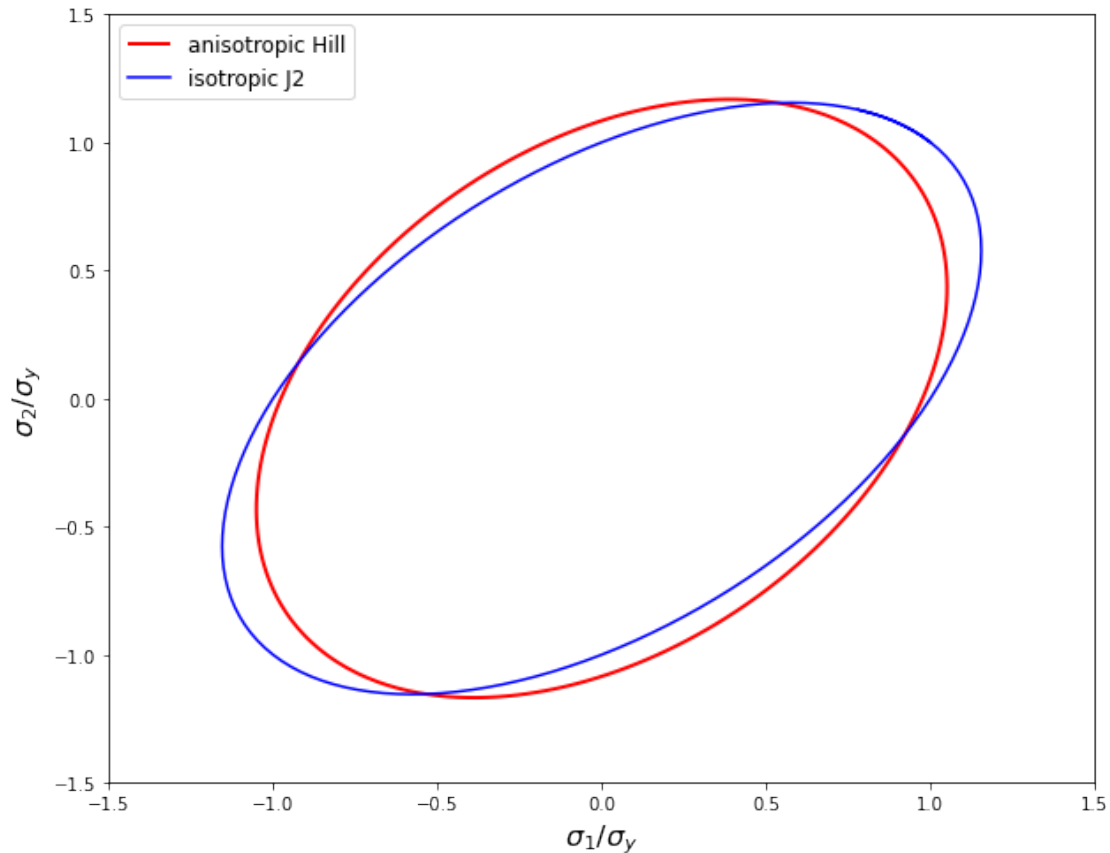
Finally, the gradient in the 3D principle stress space is obtained as

$$\frac{\partial f}{\partial \sigma_j} = \sum_{k=1}^{3} J_{jk} \frac{\partial f}{\partial s_k} \qquad (j = 1, 2, 3). \tag{11}$$

1.4 Material definition

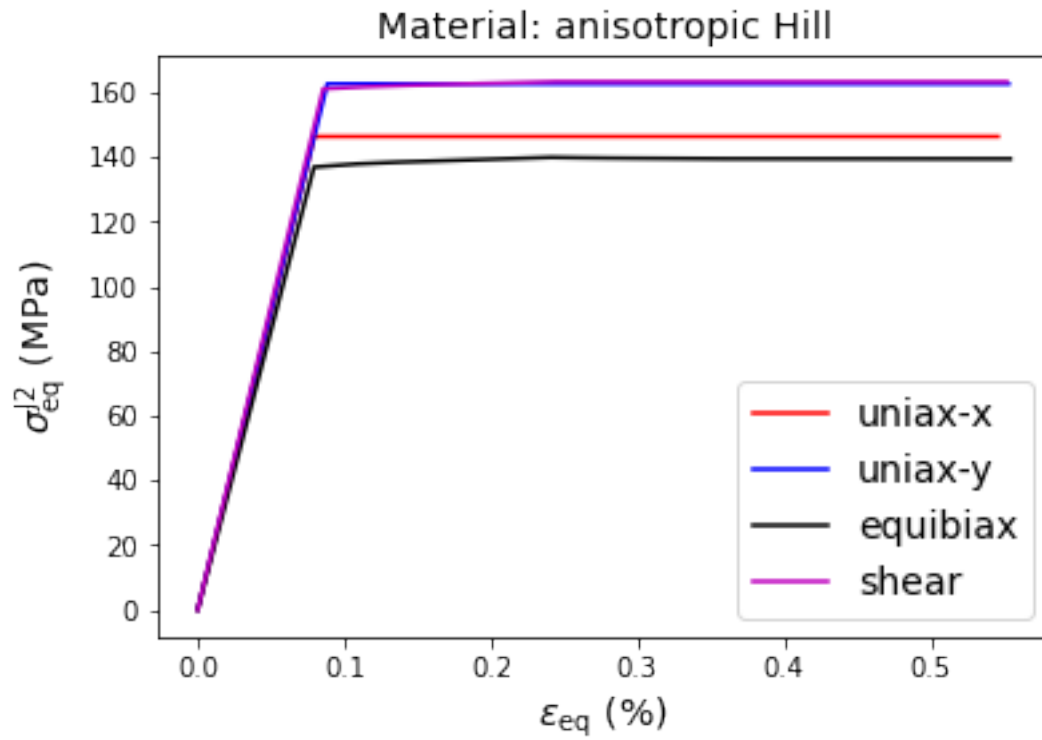
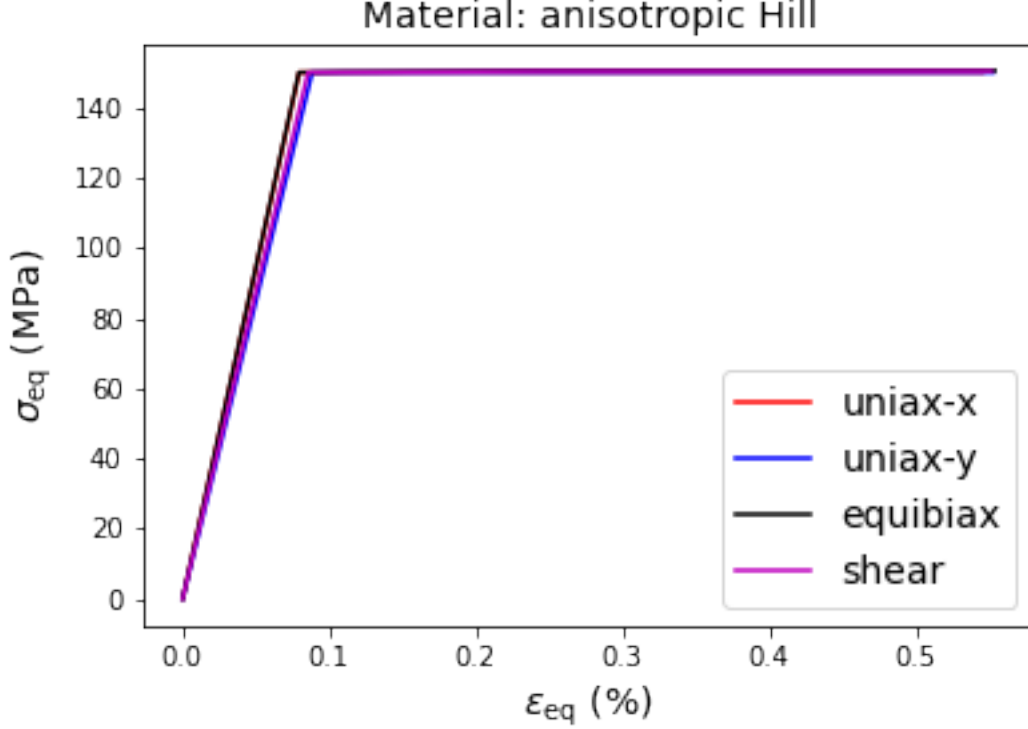
In the following, the Python class `Material` is invoked to be used as a material card in FEA, demonstrating the application of standard flow rules and machine learning (ML) flow rules. This class `Material` contains the attributes and methods to defining the elastic and plastic material behavior and to evaluate the materials constitutive behavior. Furthermore, all necessary subroutines for plotting the results are defined. Two identical materials are defined, `mat_h` will be used to apply the standard Hill-like flow rules in FEA and to generate synthetical data that is used to train a ML flow rule in `mat_ml`. Consequently, `mat_h` and `mat_ml` should reveal identical properties, which is verified in simple FEA demonstrations for 2D plane stress cases. As reference, a further material with the same yield strength, but isotropic J2 plasticity is defined as `mat_iso`. Here, we consider ideal plasticity, i.e. no work hardening, and no dependence on the hydrostatic stress.

```
[1]: import pyLabFEM as FE
     from pyLabMaterial import Material
     import numpy as np
     import matplotlib.pyplot as plt
     from scipy.optimize import fsolve
     from sklearn.metrics import r2_score

     'define two elastic-plastic materials with identical yield strength and elastic␣
      ↪properties'
     E=200.e3
     nu=0.3
     sy = 150.
     'anistropic Hill-material as reference'
     mat_h = Material(name='anisotropic Hill')
     mat_h.elasticity(E=E, nu=nu)
     mat_h.plasticity(sy=sy, hill=[0.7,1.,1.4], drucker=0., khard=0.)
     'isotropic material for ML flow rule'
     mat_ml = Material(name='ML flow rule')
     mat_ml.elasticity(E=E, nu=nu)
     mat_ml.plasticity(sy=sy, hill=[1.,1.,1.], drucker=0., khard=0.)
     print('Yield loci of anisotropic reference material and isotropic material')
     ax = mat_h.plot_yield_locus(xstart=-1.5, xend=1.5, iso=True)
```

Yield loci of anisotropic reference material and isotropic material

1.4.1 Create reference stress strain curves

The mechanical behavior of the reference material in chracterized by FEA, invoking the pyLabFEM module. Four load cases are simulated: (i) uniaxial stress in x-direction, (ii) uniaxial stress in y-direction, (iii) equibiaxial tensile strain under plane stress, and (iv) pure x-y-shear strain.

```
[2]: 'Calculate and plot stress strain curves of reference material under various␣
      ↪load cases'
     mat_h.calc_properties(verb=False, eps=0.005, sigeps=True)
     mat_h.plot_stress_strain(Hill=True)
```

```
----------------------------------------------------------
J2 yield stress under uniax-x loading: 146.385 MPa
----------------------------------------------------------
J2 yield stress under uniax-y loading: 162.698 MPa
----------------------------------------------------------
J2 yield stress under equibiax loading: 136.931 MPa
----------------------------------------------------------
J2 yield stress under shear loading: 161.126 MPa
----------------------------------------------------------
```

```
Hill yield stress under uniax-x loading: 150.0 MPa
-----------------------------------------------------------
Hill yield stress under uniax-y loading: 150.0 MPa
-----------------------------------------------------------
Hill yield stress under equibiax loading: 150.0 MPa
-----------------------------------------------------------
Hill yield stress under shear loading: 150.0 MPa
-----------------------------------------------------------
```

1.5 Setup ML constitutive models

Here, the reference material `mat_h` with Hill-type anisotropy is used to produce training and test data for the machine learning algorithm. The yield function is represented as a step function such that Support Vector Classification (SVC) is used. In the FEA all stresses are considered in form of principal stress vectors, such that a coordinate transformation into the cylindrical coordinate system with equivalent stress and polar angle θ takes place within the training and evaluation procedure. Furthermore, to make the training process efficient, stress data for training and testing are already produced as purely deviatoric stress components, following the cylidrical stress vector definition given above.

As one cannot assume the Hill-coefficients as know for the ML material, the standard J2 equivalent stresses for isotropic materials must be used in the analysis. In consequence, the yield stress must depend on the angle θ in order to discribe the anisotropy on the flow behavior. Thus, the yield function takes the form

$$f = \sigma_{eq} - \sigma_y(\theta), \tag{12}$$

in which the dependencies on isotropic J2 equivalent stress σ_{eq} and angle θ are separated.

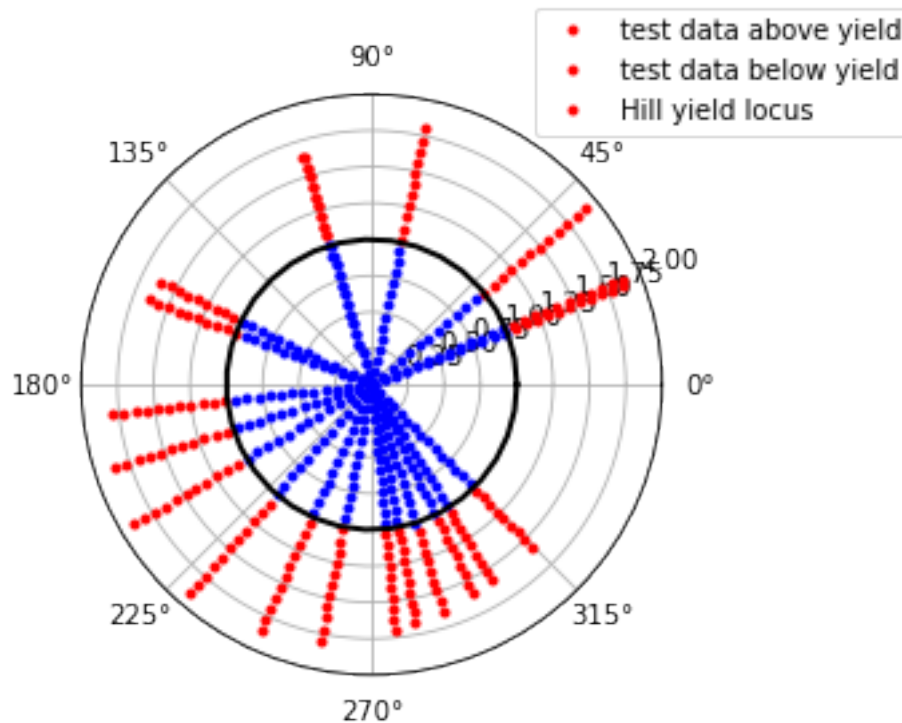
1.5.1 Train ML yield function

In the following code segment, training and test data are generated and applied for training of the SVC, and the quality test of the result. For the data and the trained ML yield function are plotted in the cylindrical stress space.

```
[3]:  'Create training data in deviatoric stress space for components seq and theta'
      def create_data(N, mat, extend=False, rand=False):
          # create stresses along unit circle normal to hydrostatic axis
          if not rand:
              theta = np.linspace(-np.pi, np.pi, N)
          else:
              theta = 2.*(np.random.rand(N)-0.5)*np.pi
          sig = FE.sp_cart(np.array([np.ones(N)*np.sqrt(3/2), theta]).T)
          offs = 0.01
          x = offs*sig
          N = 23
          for i in range(N):
              hh = offs + (1.4-offs)*(i+1)/N
              x = np.append(x, hh*sig, axis=0)
          if extend:
              # add training points in plastic regime to avoid fallback of SVC␣
      ↪decision fct. to zero
              x = np.append(x, 2.*sig, axis=0)
              x = np.append(x, 3.*sig, axis=0)
              x = np.append(x, 4.*sig, axis=0)
              x = np.append(x, 5.*sig, axis=0)
          'result data for ML yield function (only sign is considered)'
          y = np.sign(mat.calc_yf(x*mat.sy, ana=True))
          return x,y

      'Training and testing data for ML yield function, based on reference Material␣
       ↪mat_h'
      ndata = 36
      ntest = np.maximum(20, int(ndata/10))
      x_train, y_train = create_data(ndata, mat_h, extend=True)
      x_test, y_test = create_data(ntest, mat_h, rand=True)
      x_train *= mat_h.sy
      x_test *= mat_h.sy

      print('Plot theta vs. Hill eqiv. stress curves for reference material with␣
       ↪known anisotropic coefficients')
      ind1 = np.nonzero(y_test<0.)
      ind2 = np.nonzero(y_test>=0.)
      sc = FE.s_cyl(x_test, mat_h)  # convert princ. stresses into cylidrical␣
       ↪coordinates
      plt.polar(sc[ind2,1],sc[ind2,0]/mat_h.sy,'.r')
      plt.polar(sc[ind1,1],sc[ind1,0]/mat_h.sy,'.b')
      plt.polar(np.linspace(0.,2*np.pi,36), np.ones(36), '-k', linewidth=2)
      plt.legend(['test data above yield','test data below yield','Hill yield␣
       ↪locus'], loc=(0.78,0.93))
      plt.show()
```

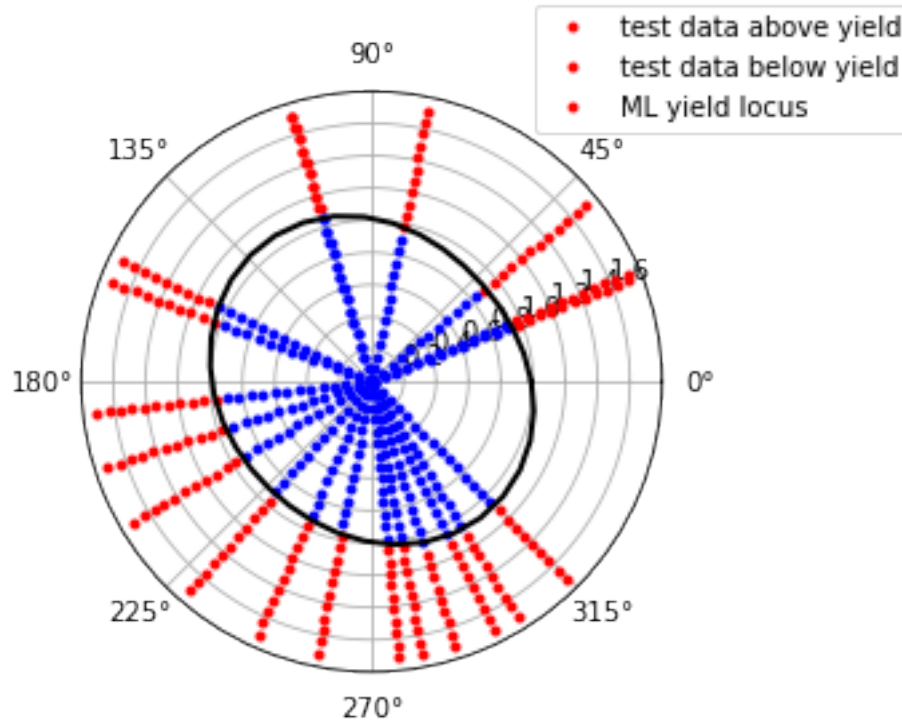
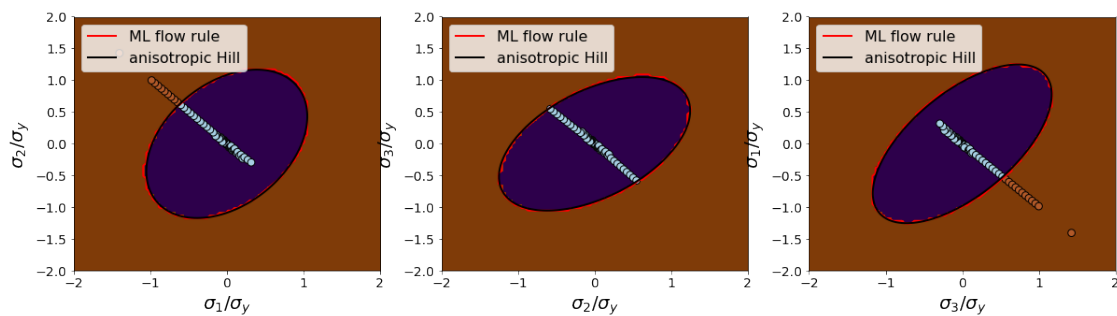
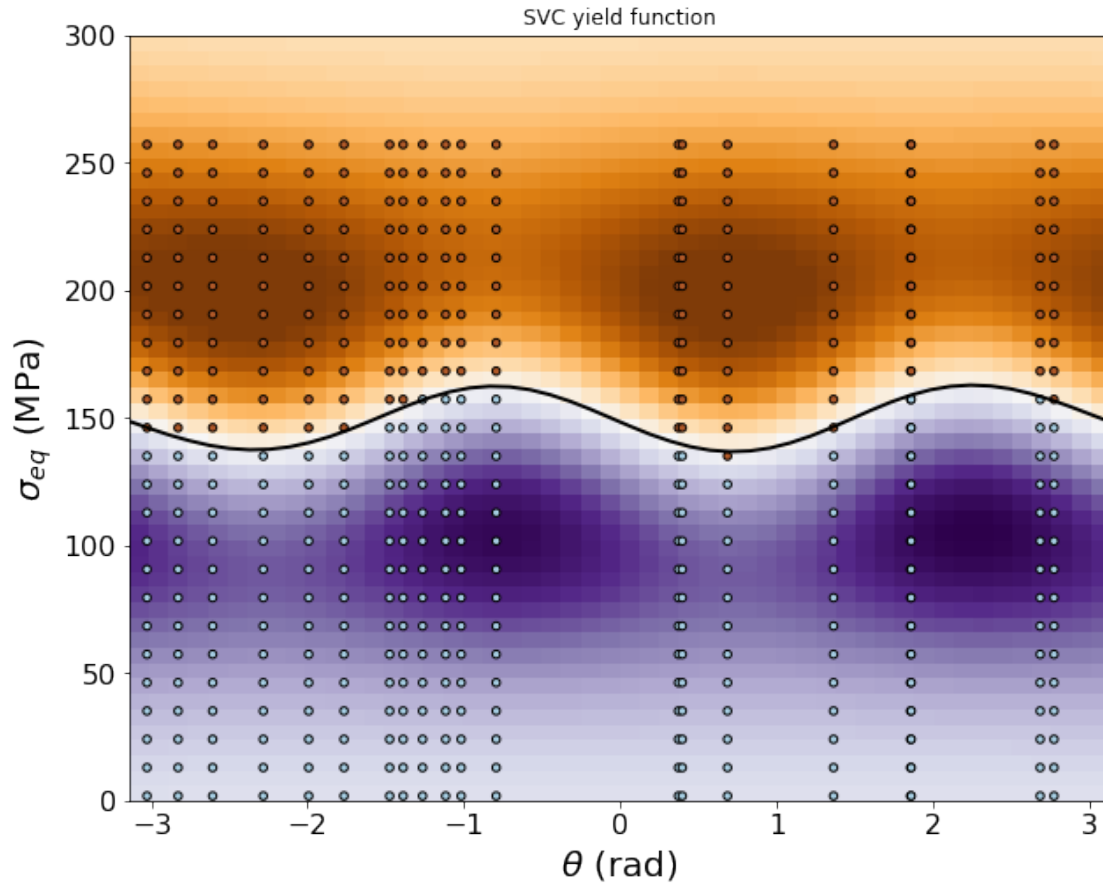
```python
'initialize and train SVC as ML yield function'
'implement ML flow rule into mat_ml'
train_sc, test_sc = mat_ml.setup_yf_SVM(x_train, y_train, x_test=x_test,
 ↪y_test=y_test,
                                         C=10, gamma=4., fs=0.4, plot=False)
y_pred_clf = mat_ml.calc_yf(x_test,pred=True)
r2_score_svm = r2_score(y_test, y_pred_clf)
print('\n------------------------\n')
print('SVM classification fitted')
print('------------------------\n')
print(mat_ml.svm_yf)
print("Training data points (only polar angle):", ndata,", Test data points:",
 ↪ntest)
print("Training set score: {} %".format(train_sc))
print("Test set score: {} %".format(test_sc))
print("r^2 on test data : %f" % r2_score_svm)
print("Number of support vectors generated: ",len(mat_ml.svm_yf.
 ↪support_vectors_))

print('Plot theta vs. J2 eqiv. stress curves for ML material')
ind1 = np.nonzero(y_test<0.)
ind2 = np.nonzero(y_test>=0.)
sc = FE.s_cyl(x_test) # convert princ. stresses into cylindrical coordinates
plt.polar(sc[ind2,1],sc[ind2,0]/mat_ml.sy,'.r')
plt.polar(sc[ind1,1],sc[ind1,0]/mat_ml.sy,'.b')
'find norm of princ. stess vector lying on yield surface'
theta = np.linspace(0.,2*np.pi,36)
snorm = FE.sp_cart(np.array([mat_ml.sy*np.ones(36)*np.sqrt(3/2), theta]).T)
x1 = fsolve(mat_ml.find_yloc, np.ones(36), args=snorm, xtol=1.e-5)
sig = snorm*np.array([x1,x1,x1]).T
s_yld = mat_ml.calc_seq(sig)
plt.polar(theta, s_yld/mat_ml.sy, '-k', linewidth=2)
plt.legend(['test data above yield','test data below yield','ML yield locus'],
 ↪loc=(0.78,0.93))
plt.show()

print('Plot of yield locus and training data in slices of 3D principle stress
 ↪space')
mat_ml.plot_yield_locus(field=True, data=x_train, ref_mat=mat_h, trange=3.e-2,
                        axis1=[0,1,2], axis2=[1,2,0])

print('Plot of trained SVM classification with test data in 2D cylindrical
 ↪stress space')
xx, yy = np.meshgrid(np.linspace(-1, 1, 50),np.linspace(-1, 1, 50))
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(10,8))
feat = np.c_[yy.ravel(),xx.ravel()]
```

```
#Z = mat_ml.svm_yf.predict(feat)
Z = mat_ml.svm_yf.decision_function(feat)
hl = mat_ml.plot_data(Z, ax, xx*np.pi, (yy+1.)*mat_ml.sy, c='black')
sc = FE.s_cyl(x_test)
h1 = ax.scatter(sc[:,1], sc[:,0], s=20, c=y_test, cmap=plt.cm.Paired,␣
 ↪edgecolors='k')
ax.set_title('SVC yield function')
ax.set_xlabel(r'$\theta$ (rad)', fontsize=20)
ax.set_ylabel(r'$\sigma_{eq}$ (MPa)', fontsize=20)
ax.tick_params(axis="x", labelsize=16)
ax.tick_params(axis="y", labelsize=16)
```

Plot theta vs. Hill eqiv. stress curves for reference material with known
anisotropic coefficients

Using priciples stresses for training

SVM classification fitted

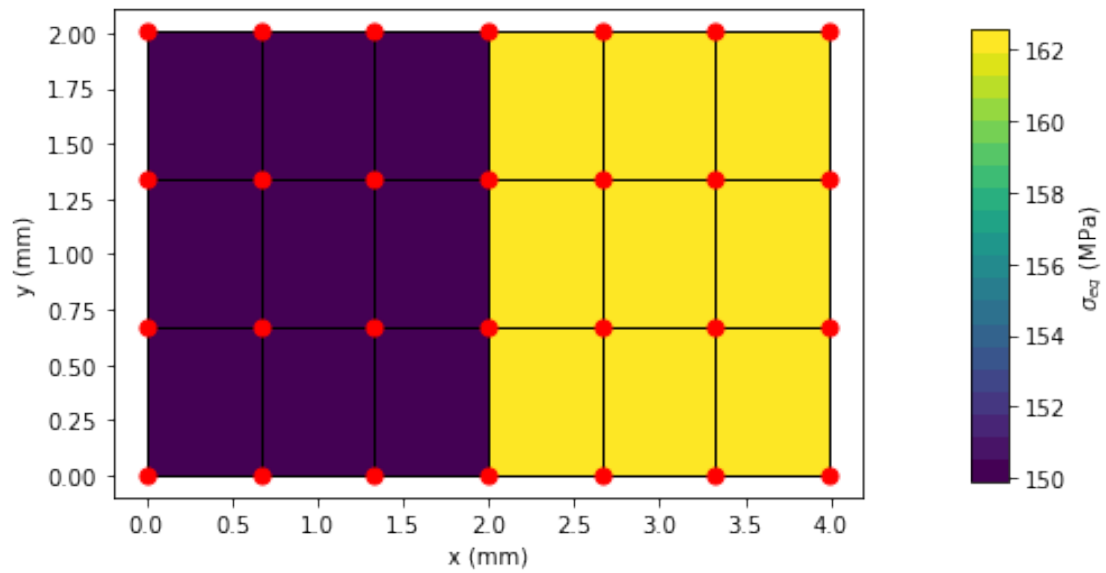
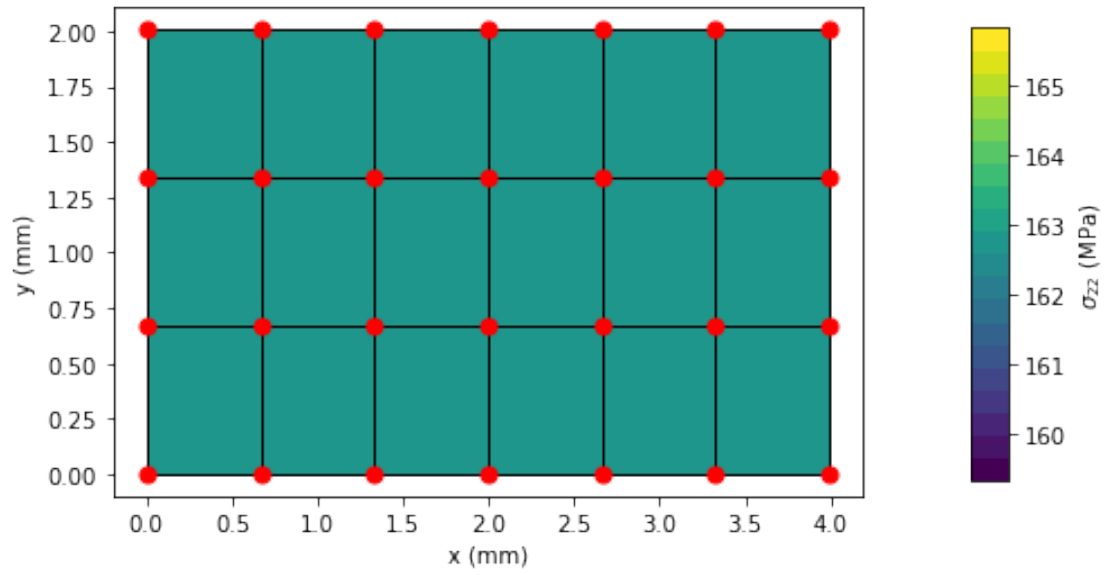
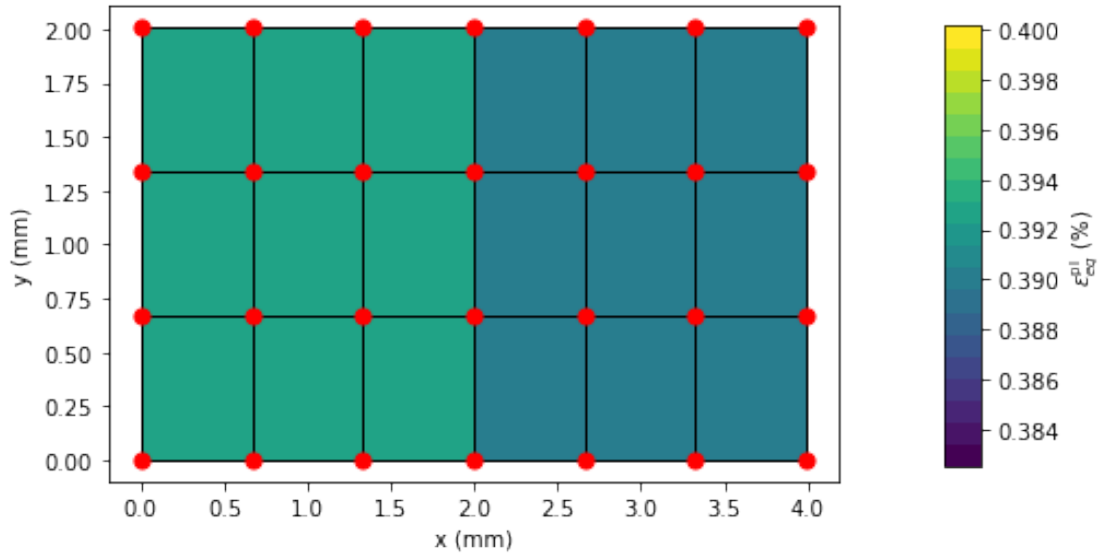
SVC(C=10, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,

```
          decision_function_shape='ovr', degree=3, gamma=4.0, kernel='rbf',
          max_iter=-1, probability=False, random_state=None, shrinking=True,
          tol=0.001, verbose=False)
Training data points (only polar angle): 36 , Test data points: 20
Training set score: 99.78976874562018 %
Test set score: 99.16666666666667 %
r^2 on test data : 0.965899
Number of support vectors generated:  145
Plot theta vs. J2 eqiv. stress curves for ML material
```

Plot of yield locus and training data in slices of 3D principle stress space
Plot of trained SVM classification with test data in 2D cylindrical stress space

1.5.2 Apply trained ML flow rule in FEA

After the successful training of the ML yield function, it can be applied directly in FEA. Its gradients of can be calculated directly from the coefficients and the support vectors resulting from the training process. The following code segment, demonstrates the application of the ML yield function in the pyLabFEA module. First, a model with two sections is generates, where section 1 is assigned to the reference material `mat_h` and section 2 is assigned to `mat_ml` with the ML yield function.

```
[4]:  fem=FE.Model(dim=2,planestress=True)
      fem.geom([2, 2], LY=2.)   # define sections in absolute lengths
      print('==========================================')
      print('=== FEA with reference and ML material ===')
      print('==========================================')
      fem.assign([mat_h, mat_ml])   # create a model with trained ML-flow rule (mat_h)
      ↪and reference material (mat3)
      fem.bcleft(0.)
      fem.bcbot(0.)
      fem.bcright(0., 'force')    # free right edge
```

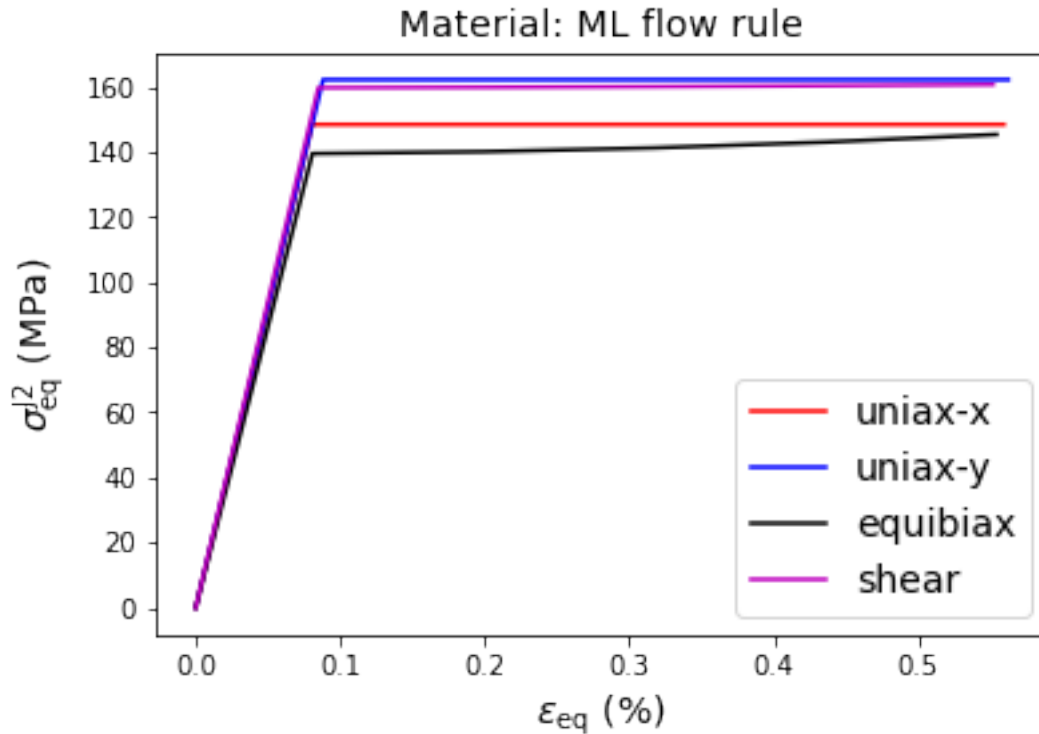
```
fem.bctop(0.004*fem.leny, 'disp')  # apply displacement at top nodes (uniax␣
 ↪y-stress)
fem.mesh(NX=6, NY=3)
fem.solve()
fem.calc_global()
print('Global strain: ', np.round(fem.glob['eps'],decimals=4))
print('Element strain (ref): ', np.round(fem.element[0].eps,decimals=4))
print('Element strain (ML): ', np.round(fem.element[10].eps,decimals=4))
print('Global stress: ', np.round(fem.glob['sig'],decimals=4))
print('Element stress (ref): ', np.round(fem.element[0].sig,decimals=4))
print('Element stress (ML): ', np.round(fem.element[10].sig,decimals=4))
print('Global plastic strain: ', np.round(fem.glob['epl'],decimals=5))
print('Plastic strain (ref): ', np.round(fem.element[0].epl,decimals=5))
print('Plastic strain (ML): ', np.round(fem.element[10].epl,decimals=5))
print('--------------------------')
print('Material 1 (left block):  ',mat_h.msg)
print('Material 2 (right block): ',mat_ml.msg)

fem.plot('stress2',mag=1)
fem.plot('seq',mag=1)
print('Note: Different definitions of SEQ for Hill and J2.')
fem.plot('peeq',mag=1)
```

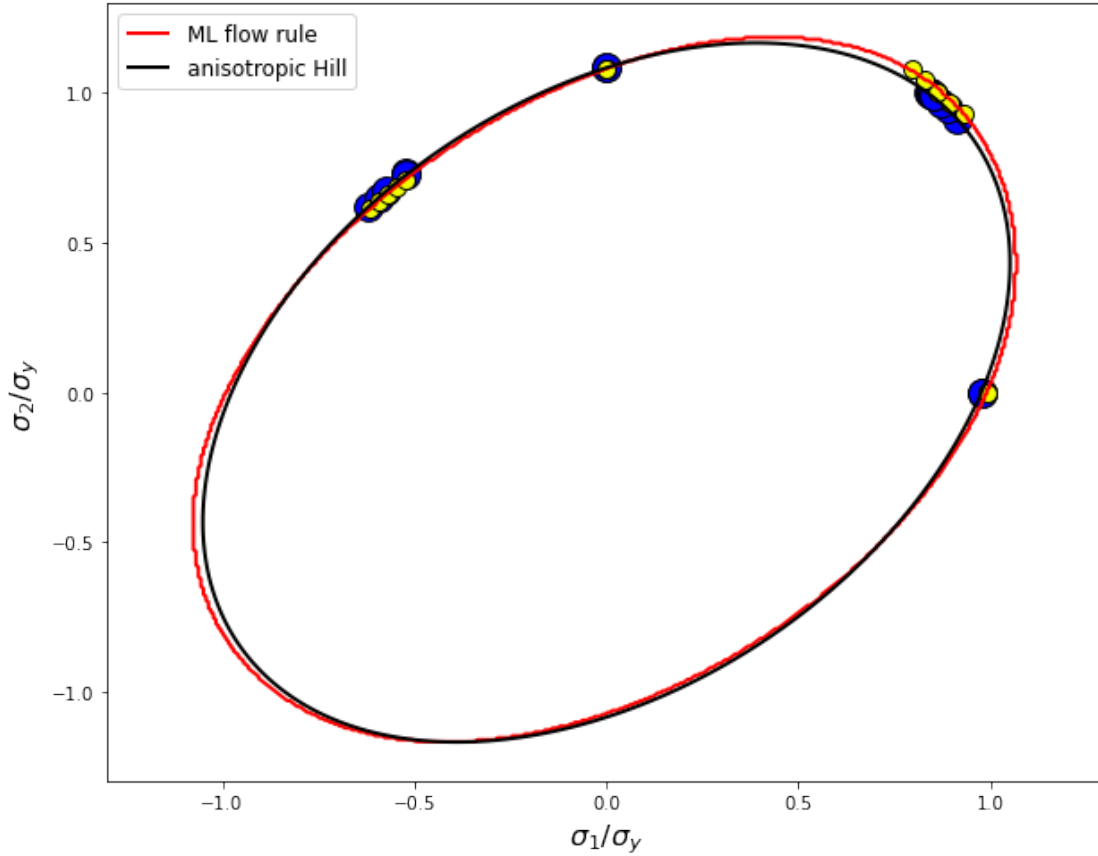
```
========================================
=== FEA with reference and ML material ===
========================================
Global strain:  [-0.0017  0.004  -0.001   0.      0.     -0.    ]
Element strain (ref):  [-0.0016  0.004  -0.001   0.      0.     -0.    ]
Element strain (ML):  [-0.0018  0.004  -0.0009  0.      0.     -0.    ]
Global stress:  [  0.     162.5974  0.      0.      0.     -0.    ]
Element stress (ref):  [  0.     162.5974  0.      0.      0.     -0.    ]
Element stress (ML):  [  0.     162.5974  0.      0.      0.     -0.    ]
Global plastic strain:  [-0.00144  0.00319 -0.00175  0.      0.      0.     ]
Plastic strain (ref):  [-0.00131  0.00319 -0.00187  0.      0.      0.     ]
Plastic strain (ML):  [-0.00156  0.00319 -0.00163  0.      0.      0.     ]
----------------------------
Material 1 (left block):   {'yield_fct': 'analytical', 'gradient': 'analytical'}
Material 2 (right block):  {'yield_fct': 'ML_yf-decision-fct', 'gradient':
'gradient to ML_yf'}
```

Note: Different definitions of SEQ for Hill and J2.

Note that the element stresses in both materials are very similar (top figure). However, the different definitions of the Hill equivalent stress and the J2 equivalent stress produce a numerical difference in the plot of the equivalent stresses (middle figure). The resulting plastic strains in both sections are quite similar (bottom figure).

The second example applies the same four load cases under which the reference material has been characterized and compares the results.

```
[5]: print('\n\n====== Stress-Strain-Curves ======')
     mat_ml.calc_properties(verb=False, eps=0.005, sigeps=True, min_step=5)
     mat_ml.plot_stress_strain()
     for sel in mat_h.prop:
         hh = mat_ml.propJ2[sel]['ys']/mat_h.propJ2[sel]['ys'] - 1.
         print('*** load case: ',mat_h.prop[sel]['name'])
         print('            Rel. error in yield strength : ', np.round(100*hh,␣
     ↪decimals=3),'%')
         hh = np.amax(mat_ml.propJ2[sel]['peeq'])/np.amax(mat_h.propJ2[sel]['peeq'])␣
     ↪- 1.
         print('Rel. error in plastic strain at max. load : ', np.
     ↪round(100*hh,decimals=3),'%')

     'plot yield locus'
     ax = mat_ml.plot_yield_locus(xstart=-1.3, xend=1.3, ref_mat=mat_h, field=False,␣
     ↪Nmesh=400)
     print('\nPlot evolution of stresses during plastic deformation for both␣
     ↪materials')
     print('Hill-material: blue color')
     print('ML-material: yellow color')
     s=100
```

```
'Hill material'
stx = mat_h.sigeps['stx']['sig'][:,0:2]/mat_h.sy
sty = mat_h.sigeps['sty']['sig'][:,0:2]/mat_h.sy
et2 = mat_h.sigeps['et2']['sig'][:,0:2]/mat_h.sy
ect = mat_h.sigeps['ect']['sig'][:,0:2]/mat_h.sy
ax.scatter(stx[1:,0],stx[1:,1],s=s*2.5, c='#0000ff', edgecolors='k')
ax.scatter(sty[1:,0],sty[1:,1],s=s*2.5, c='#0000ff', edgecolors='k')
ax.scatter(et2[1:,0],et2[1:,1],s=s*2.5, c='#0000ff', edgecolors='k')
ax.scatter(ect[1:,0],ect[1:,1],s=s*2.5, c='#0000ff', edgecolors='k')
'ML material'
stx = mat_ml.sigeps['stx']['sig'][:,0:2]/mat_h.sy
sty = mat_ml.sigeps['sty']['sig'][:,0:2]/mat_h.sy
et2 = mat_ml.sigeps['et2']['sig'][:,0:2]/mat_h.sy
ect = mat_ml.sigeps['ect']['sig'][:,0:2]/mat_h.sy
ax.scatter(stx[1:,0],stx[1:,1],s=s, c='#f0ff00', edgecolors='k')
ax.scatter(sty[1:,0],sty[1:,1],s=s, c='#f0ff00', edgecolors='k')
ax.scatter(et2[1:,0],et2[1:,1],s=s, c='#f0ff00', edgecolors='k')
ax.scatter(ect[1:,0],ect[1:,1],s=s, c='#f0ff00', edgecolors='k')
```

```
====== Stress-Strain-Curves ======
-----------------------------------------------------------
J2 yield stress under uniax-x loading: 148.447 MPa
-----------------------------------------------------------
J2 yield stress under uniax-y loading: 162.203 MPa
-----------------------------------------------------------
J2 yield stress under equibiax loading: 139.509 MPa
-----------------------------------------------------------
J2 yield stress under shear loading: 159.728 MPa
-----------------------------------------------------------
```

```
*** load case:  uniax-x
            Rel. error in yield strength :   1.408 %
Rel. error in plastic strain at max. load :  -1.948 %
*** load case:  uniax-y
            Rel. error in yield strength :  -0.304 %
Rel. error in plastic strain at max. load :  -0.448 %
*** load case:  equibiax
            Rel. error in yield strength :   1.883 %
Rel. error in plastic strain at max. load :  -1.356 %
*** load case:  shear
            Rel. error in yield strength :  -0.867 %
Rel. error in plastic strain at max. load :   0.242 %

Plot evolution of stresses during plastic deformation for both materials
Hill-material: blue color
ML-material: yellow color
```

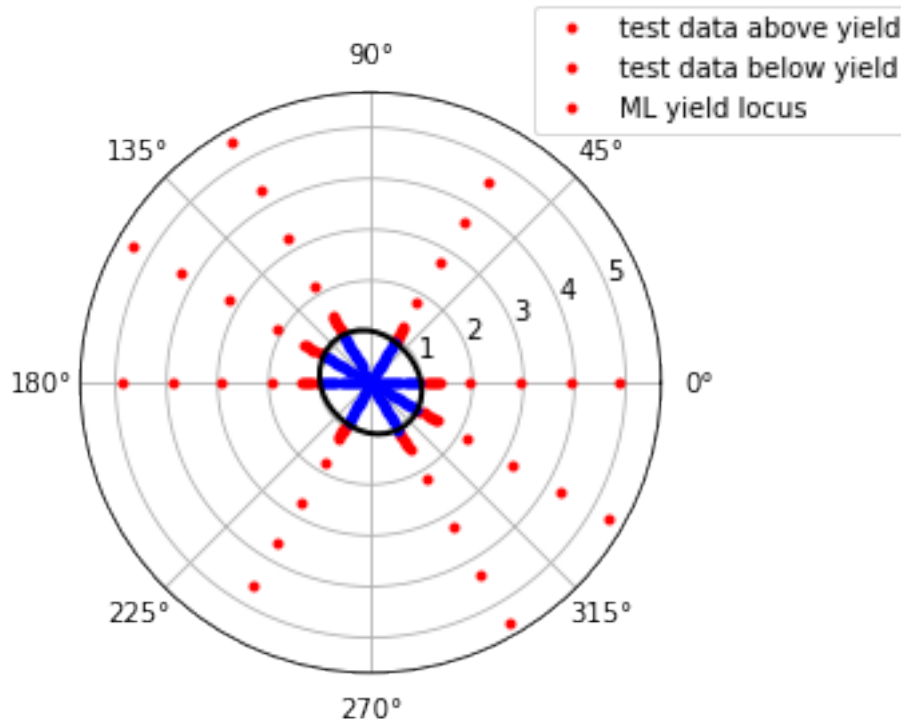
[5]: <matplotlib.collections.PathCollection at 0x1a24d32a50>

1.6 Train ML yield function with results of 4 load cases

In the following, the training data is created only from the four load cases of the reference material. For all load cases $\sigma_3 = 0$, yet due to the data-oriented formulation of the yield function on the deviatoric plane, a full yield function can be achieved. A new material `mat3` is created on which the ML training is performed.

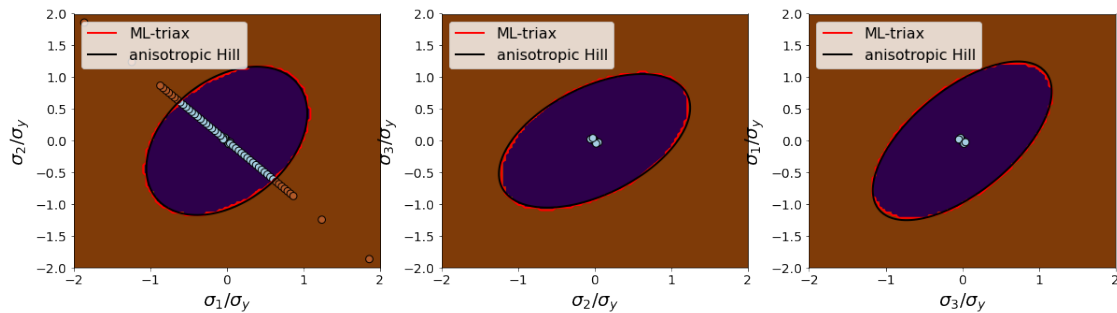
```
[6]: 'get principle yield stresses of 4 load cases from reference material'
     sig=np.zeros((4,3))
     i=0
     for sel in mat_h.sigeps:
         peeq = FE.eps_eq(mat_h.sigeps[sel]['epl'])
         iys = np.nonzero(peeq<1.e-6)   # take stress at last index of elastic regime
         ys = mat_h.sigeps[sel]['sig']
         sp_yld = FE.Stress(ys[iys[0][-1]]).p
         sig[i,:] = sp_yld
         i += 1

     'mirror data in theta space: tension-compression symmetry'
     sc1 = FE.s_cyl(sig)
```

```python
sc2 = np.zeros((4,3))
sc2[:,0] = sc1[:,0]
sc2[:,1] = sc1[:,1]-np.pi
sc2[:,2] = sc1[:,2]
sc = np.append(sc1,sc2,axis=0)
sig = FE.sp_cart(sc)

'expand stresses into elastic and plastic regimes'
offs = 0.01
x = offs*sig
N = 23
for i in range(N):
    hh = offs + (1.4-offs)*(i+1)/N
    x = np.append(x, hh*sig, axis=0)
# add training points in plastic regime to avoid fallback of SVC decision fct.␣
 ↪to zero
x = np.append(x, 2.*sig, axis=0)
x = np.append(x, 3.*sig, axis=0)
x = np.append(x, 4.*sig, axis=0)
x = np.append(x, 5.*sig, axis=0)

'result data for training of ML yield function'
y = mat_ml.calc_yf(x, pred=True)

print('Plot theta vs. J2 eqiv. stress curves for ML material')
ind1 = np.nonzero(y<0.)
ind2 = np.nonzero(y>=0.)
sc = FE.s_cyl(x) # convert princ. stresses into cylindrical coordinates
plt.polar(sc[ind2,1],sc[ind2,0]/mat_ml.sy,'.r')
plt.polar(sc[ind1,1],sc[ind1,0]/mat_ml.sy,'.b')
'find norm of princ. stess vector lying on yield surface'
theta = np.linspace(0.,2*np.pi,36)
snorm = FE.sp_cart(np.array([mat_ml.sy*np.ones(36)*np.sqrt(3/2), theta]).T)
x1 = fsolve(mat_ml.find_yloc, np.ones(36), args=snorm, xtol=1.e-5)
hs = snorm*np.array([x1,x1,x1]).T
s_yld = mat_ml.calc_seq(hs)
plt.polar(theta, s_yld/mat_ml.sy, '-k', linewidth=2)
plt.legend(['test data above yield','test data below yield','ML yield locus'],␣
 ↪loc=(0.78,0.93))
plt.show()

'define material for training with new data'
mat3 = Material(name='ML-triax')
mat3.elasticity(E=E, nu=nu)
mat3.plasticity(sy=mat_h.sy, hill=[1., 1., 1.], drucker=0., khard=0.)

'initialize and train SVC as yield function'
```

```python
train_sc, test_sc = mat3.setup_yf_SVM(x, y, x_test=x_test, y_test=y_test,
                                      C=10, gamma=4., fs=0.3, plot=False)
y_pred_clf = mat3.calc_yf(x_test,pred=True)
r2_score_svm = r2_score(y_test, y_pred_clf)
print('\n------------------------\n')
print('SVM classification fitted')
print('------------------------\n')
print(mat3.svm_yf)
print("Training data points (only polar angle):", ndata,", Test data points:",
 →ntest)
print("Training set score: {} %".format(train_sc))
print("Test set score: {} %".format(test_sc))
print("r^2 on test data : %f" % r2_score_svm)
print("Number of support vectors generated: ",len(mat3.svm_yf.support_vectors_))

print('Plot of yield locus and training data in slices of 3D principle stress
 →space')
mat3.plot_yield_locus(ref_mat=mat_h, data=x, field=True,
                      trange=3.e-2, axis1=[0,1,2], axis2=[1,2,0])
```

Plot theta vs. J2 eqiv. stress curves for ML material

Using priciples stresses for training

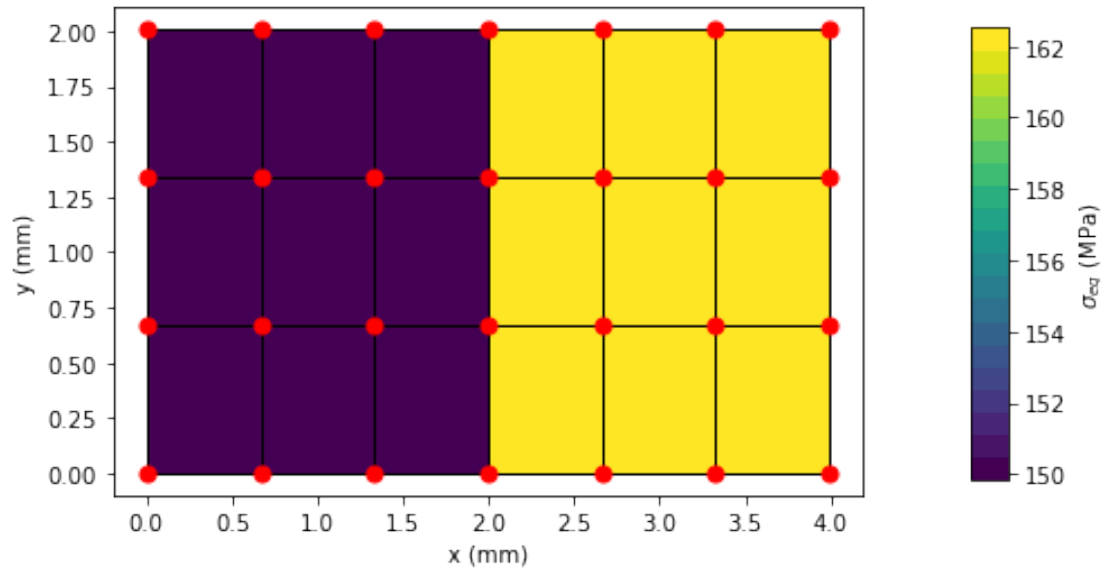
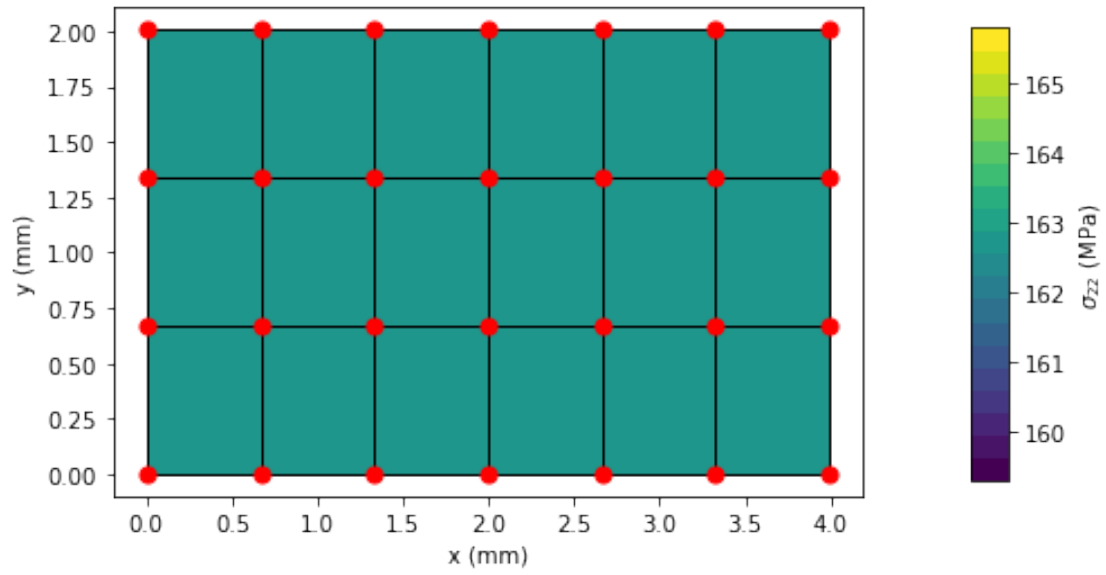
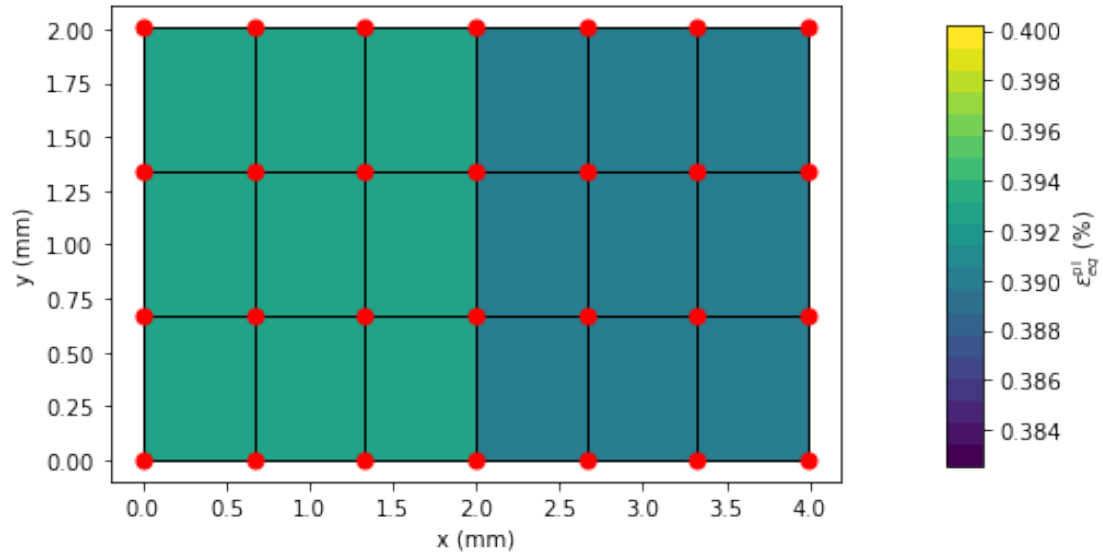
```
--------------------------

SVM classification fitted
--------------------------

SVC(C=10, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=4.0, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
Training data points (only polar angle): 36 , Test data points: 20
Training set score: 99.28571428571429 %
Test set score: 99.16666666666667 %
r^2 on test data : 0.965899
Number of support vectors generated:  67
Plot of yield locus and training data in slices of 3D principle stress space
```

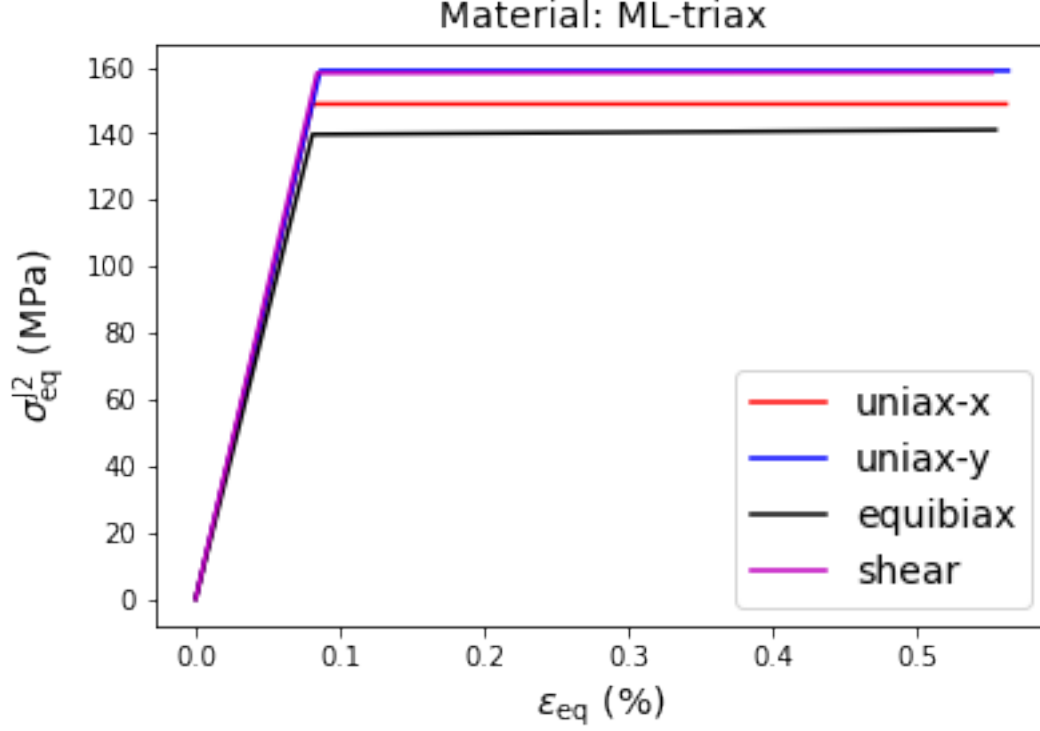
[6]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x1a2476bc90>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1a24769650>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1a24b92c90>],
      dtype=object)
```

1.7 Comparison of different training methods

The following code segment shows the same set of FE simulations as in the case of the yield function
fitted to regular training data

[7]:
```
fem=FE.Model(dim=2,planestress=True)
fem.geom([2, 2], LY=2.)  # define sections in absolute lengths

print('========================================')
print('=== FEA with reference and ML material ===')
print('========================================')
fem.assign([mat_h, mat3])  # create a model with trained ML-flow rule (mat_h)
 ↪and reference material (mat3)
fem.bcleft(0.)
```

```
fem.bcbot(0.)
fem.bcright(0., 'force')    # free right edge
fem.bctop(0.004*fem.leny, 'disp')   # apply displacement at top nodes (uniax␣
 ↪y-stress)
fem.mesh(NX=6, NY=3)
fem.solve()
fem.calc_global()
print('Global strain: ', np.round(fem.glob['eps'],decimals=4))
print('Element strain (ref): ', np.round(fem.element[0].eps,decimals=4))
print('Element strain (ML): ', np.round(fem.element[10].eps,decimals=4))
print('Global stress: ', np.round(fem.glob['sig'],decimals=4))
print('Element stress (ref): ', np.round(fem.element[0].sig,decimals=4))
print('Element stress (ML): ', np.round(fem.element[10].sig,decimals=4))
print('Global plastic strain: ', np.round(fem.glob['epl'],decimals=5))
print('Plastic strain (ref): ', np.round(fem.element[0].epl,decimals=5))
print('Plastic strain (ML): ', np.round(fem.element[10].epl,decimals=5))
print('--------------------------')
print('Material 1 (left section):  ',mat_h.msg)
print('Material 3 (right section): ',mat3.msg)

fem.plot('stress2',mag=1)
fem.plot('seq',mag=1)
print('Note: Different definitions of SEQ for Hill and J2.')
fem.plot('peeq',mag=1)
```

```
==========================================
=== FEA with reference and ML material ===
==========================================
Global strain:  [-0.0017  0.004  -0.001   0.      0.     -0.    ]
Element strain (ref):  [-0.0016  0.004  -0.001   0.      0.      0.    ]
Element strain (ML):  [-0.0018  0.004  -0.0009  0.      0.     -0.    ]
Global stress:  [  0.    162.545   0.      0.      0.     -0.    ]
Element stress (ref):  [  0.    162.545   0.      0.      0.     -0.    ]
Element stress (ML):  [  0.    162.545   0.      0.      0.     -0.    ]
Global plastic strain:  [-0.00144  0.00319 -0.00174  0.       0.       0.      ]
Plastic strain (ref):  [-0.00131  0.00319 -0.00187  0.       0.       0.      ]
Plastic strain (ML):  [-0.00157  0.00319 -0.00161  0.       0.       0.      ]
----------------------------
Material 1 (left section):  {'yield_fct': 'analytical', 'gradient':
'analytical'}
Material 3 (right section): {'yield_fct': 'ML_yf-decision-fct', 'gradient':
'gradient to ML_yf'}
```

Note: Different definitions of SEQ for Hill and J2.

```
[8]: print('\n\n====== Stress-Strain-Curves ======')
     mat3.calc_properties(verb=False, eps=0.005)
     mat3.plot_stress_strain()
     for sel in mat_h.prop:
         hh = mat_ml.propJ2[sel]['ys']/mat_h.propJ2[sel]['ys'] - 1.
         print('*** load case: ',mat_h.prop[sel]['name'])
         print('              Rel. error in yield strength : ', np.round(100*hh,
     ↪decimals=3),'%')
         hh = np.amax(mat_ml.propJ2[sel]['peeq'])/np.amax(mat_h.propJ2[sel]['peeq'])
     ↪- 1.
         print('Rel. error in plastic strain at max. load : ', np.
     ↪round(100*hh,decimals=3),'%')
```

```
====== Stress-Strain-Curves ======
----------------------------------------------------------
J2 yield stress under uniax-x loading: 148.865 MPa
----------------------------------------------------------
J2 yield stress under uniax-y loading: 158.896 MPa
----------------------------------------------------------
J2 yield stress under equibiax loading: 139.707 MPa
----------------------------------------------------------
J2 yield stress under shear loading: 158.19 MPa
----------------------------------------------------------
```

```
*** load case:  uniax-x
            Rel. error in yield strength :   1.408 %
Rel. error in plastic strain at max. load :   -1.948 %
*** load case:  uniax-y
            Rel. error in yield strength :   -0.304 %
Rel. error in plastic strain at max. load :   -0.448 %
*** load case:  equibiax
            Rel. error in yield strength :   1.883 %
Rel. error in plastic strain at max. load :   -1.356 %
*** load case:  shear
            Rel. error in yield strength :   -0.867 %
Rel. error in plastic strain at max. load :   0.242 %
```

1.8   Tresca yield function

In the next example a ML flow rule will be trained to a reference material with Tresca yield function

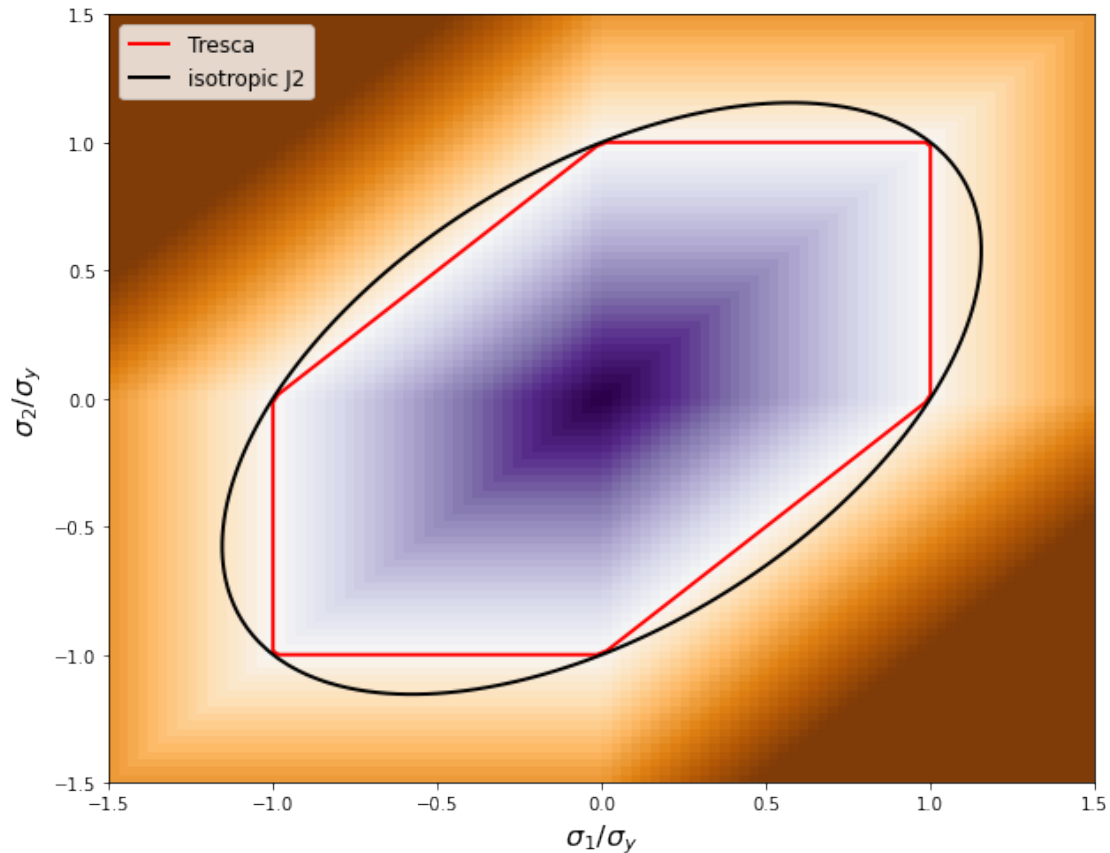
$$f = \sigma_{eq}^{\text{Tresca}} - \sigma_y \,, \tag{13}$$

with the Tresca equivalent stress

$$\sigma_{eq}^{\text{Tresca}} = \frac{\sigma_{\text{I}} - \sigma_{\text{III}}}{2}. \tag{14}$$

based on the principal stresses in the sequence $\sigma_I \geq \sigma_{II} \geq \sigma_{III}$. In the following cell, the material definitions are performed.

```
[9]: 'define two elastic-plastic materials with identical yield strength and elastic␣
      ↪properties'
     E=200.e3
     nu=0.3
     sy = 150.
     'Tresca-material as training model'
     mat_t = Material(name='Tresca')
     mat_t.elasticity(E=E, nu=nu)
     mat_t.plasticity(sy=sy, Tresca=True)
     'Isotropic J2 material as basis for comparisons'
     mat_iso = Material(name='isotropic J2')
     mat_iso.elasticity(E=E, nu=nu)
     mat_iso.plasticity(sy=sy)
     'isotropic material as basis for ML flow rule'
     mat_mlt = Material(name='ML flow rule')
     mat_mlt.elasticity(E=E, nu=nu)
     mat_mlt.plasticity(sy=sy, hill=[1.,1.,1.], drucker=0., khard=0.)
     print('Yield loci of anisotropic reference material and isotropic material')
     ax = mat_t.plot_yield_locus(xstart=-1.5, xend=1.5, ref_mat=mat_iso, field=True)
```

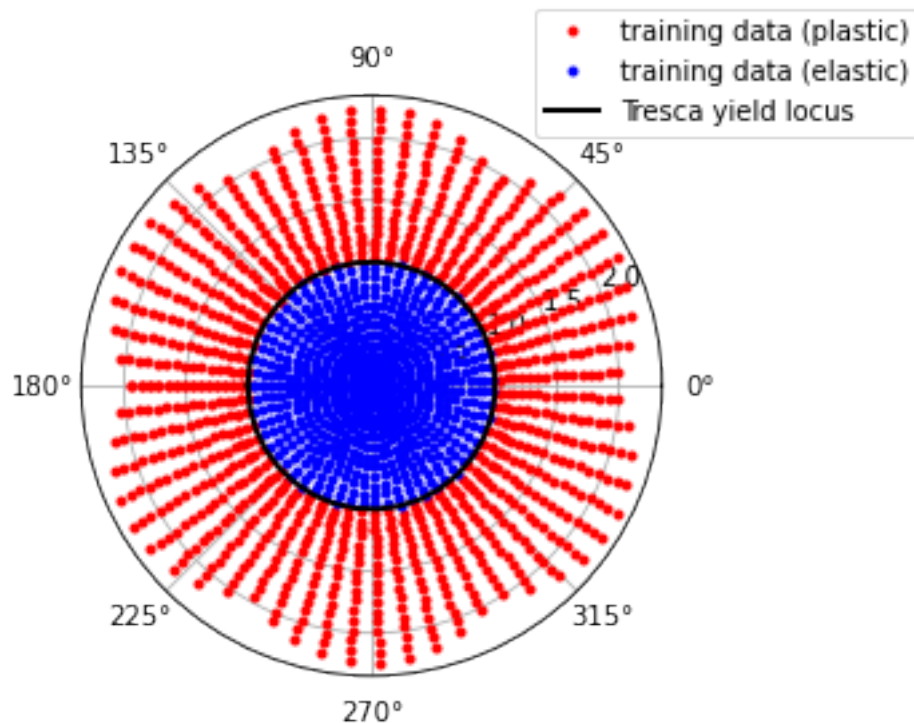
Yield loci of anisotropic reference material and isotropic material

In the nect step, the training data is produced based on the material with Tresca yield function, and the training of the ML yield function is performed.

```
[10]: 'Create training data in deviatoric stress space for components seq and theta'
      def create_data(N, mat, extend=False, rand=False):
          # create stresses along unit circle normal to hydrostatic axis
          if not rand:
              theta = np.linspace(-np.pi, np.pi, N)
          else:
              theta = 2.*(np.random.rand(N)-0.5)*np.pi
          sig = np.array([np.ones(N), theta]).T
          'expand stresses into elastic and plastic regimes'
          'construct elastic data points'
          xt = sig*np.array([0.01,1.])
          offs = 0.05
          nl = 15
          for i in range(nl):
              hh = np.array([1 - (i/nl)**1.1, 1.])  # down-scaling of equiv. stresses␣
      ↪in elastic regime
              xt = np.append(xt, hh*sig, axis=0)
```

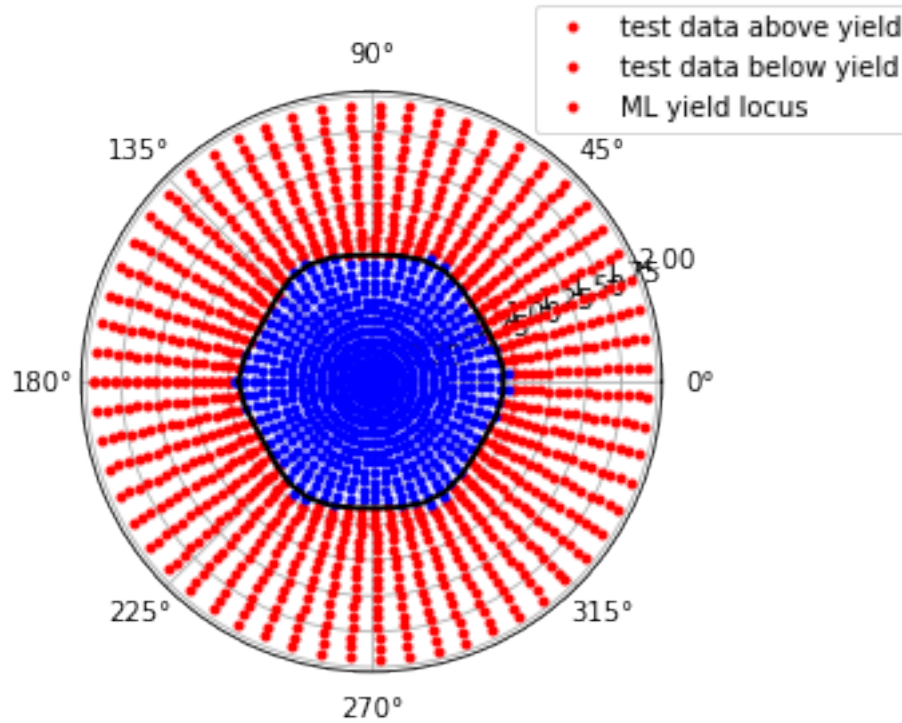
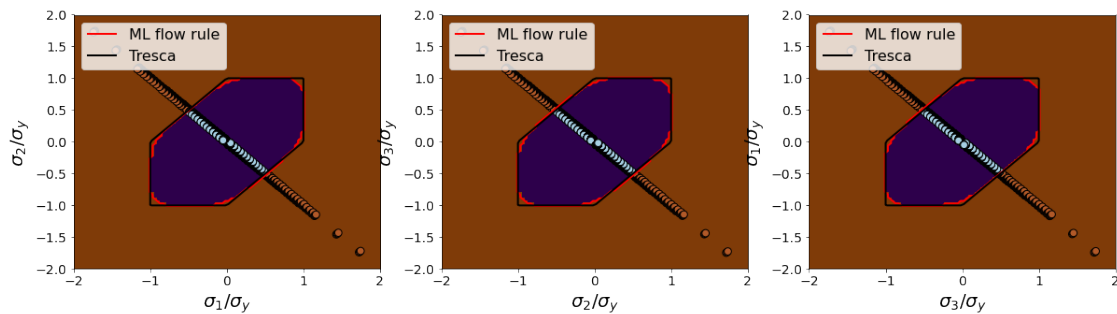
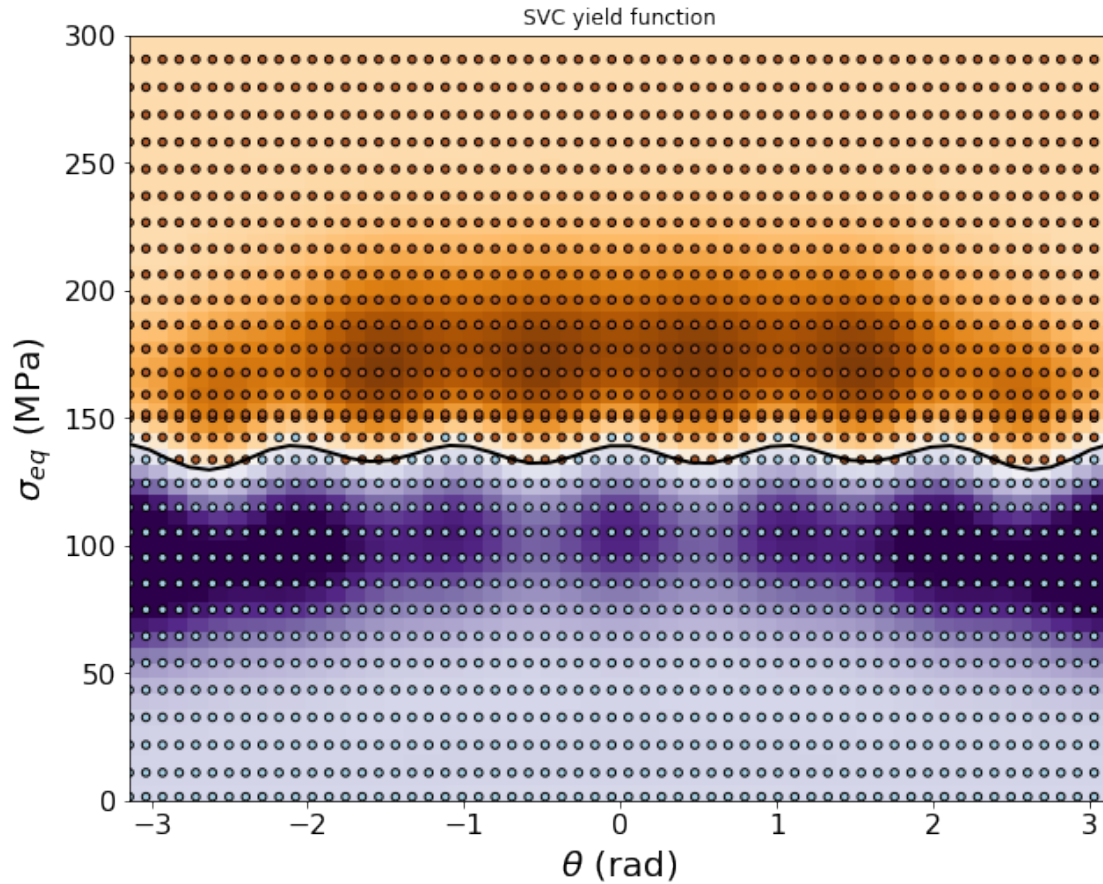
```python
    'construct plastic data points'
    offs = 1.01
    for i in range(nl):
        hh = np.array([offs + (i/nl)**1.1, 1.])
        xt = np.append(xt, hh*sig, axis=0)

    'add further training points far in plastic regime to avoid fallback of SVC␣
 ↪decision fct. to zero'
    if extend:
        for i in range(9):
            hh = np.array([i*0.5+2., 1.])
            xt = np.append(xt, hh*sig, axis=0)
    'result data for ML yield function (only sign is considered)'
    x = FE.sp_cart(xt)
    y = np.sign(mat.calc_yf(x*mat.sy))
    return x,y

'Training and testing data for ML yield function, based on reference Material␣
 ↪mat_t'
ndata = 600
ntest = np.maximum(20, int(ndata/10))
x_train, y_train = create_data(ndata, mat_t, extend=True)
x_test, y_test = create_data(ntest, mat_t, rand=False)
x_train *= mat_t.sy
x_test *= mat_t.sy

print('Plot theta vs. Tresca eqiv. stress curves for reference material')
ind_e = np.nonzero(y_test<0.)[0]
ind_pl = np.nonzero(y_test>=0.)[0]
sc = FE.s_cyl(x_test, mat_t)  # convert princ. stresses into cylidrical␣
 ↪coordinates
plt.polar(sc[ind_pl,1],sc[ind_pl,0]/mat_t.sy,'.r')
plt.polar(sc[ind_e,1],sc[ind_e,0] /mat_t.sy,'.b')
'find norm of princ. stess vector lying on yield surface'
theta = np.linspace(-np.pi,np.pi,36)
snorm = FE.sp_cart(np.array([mat_t.sy*np.ones(36)*np.sqrt(3/2), theta]).T)
x1 = fsolve(mat_t.find_yloc, np.ones(36), args=snorm, xtol=1.e-5)
sig = snorm*np.array([x1,x1,x1]).T
s_yld = mat_t.calc_seq(sig)
plt.polar(theta, s_yld/mat_t.sy, '-k', linewidth=2)
plt.legend(['training data (plastic)','training data (elastic)','Tresca yield␣
 ↪locus'], loc=(0.78,0.93))
plt.show()

'initialize and train SVC as ML yield function'
'implement ML flow rule into mat_mlt'
```

```python
train_sc, test_sc = mat_mlt.setup_yf_SVM(x_train, y_train, x_test=x_test,␣
 ↪y_test=y_test,
                                         C=50, gamma=9., fs=0.4, plot=False)
y_pred_clf = mat_mlt.calc_yf(x_test,pred=True)
r2_score_svm = r2_score(y_test, y_pred_clf)
print('\n------------------------\n')
print('SVM classification fitted')
print('------------------------\n')
print(mat_mlt.svm_yf)
print("Training data points (only polar angle):", ndata,", Test data points:",␣
 ↪ntest)
print("Training set score: {} %".format(train_sc))
print("Test set score: {} %".format(test_sc))
print("r^2 on test data : %f" % r2_score_svm)
print("Number of support vectors generated: ",len(mat_mlt.svm_yf.
 ↪support_vectors_))

print('Plot theta vs. J2 eqiv. stress curves for ML material')
ind1 = np.nonzero(y_test<0.)
ind2 = np.nonzero(y_test>=0.)
sc = FE.s_cyl(x_test) # convert princ. stresses into cylindrical coordinates
plt.polar(sc[ind2,1],sc[ind2,0]/mat_mlt.sy,'.r')
plt.polar(sc[ind1,1],sc[ind1,0]/mat_mlt.sy,'.b')
'find norm of princ. stess vector lying on yield surface'
theta = np.linspace(-np.pi,np.pi,36)
snorm = FE.sp_cart(np.array([mat_mlt.sy*np.ones(36)*np.sqrt(3/2), theta]).T)
x1 = fsolve(mat_mlt.find_yloc, 0.8*np.ones(36), args=snorm, xtol=1.e-5)
sig = snorm*np.array([x1,x1,x1]).T
s_yld = mat_mlt.calc_seq(sig)
plt.polar(theta, s_yld/mat_mlt.sy, '-k', linewidth=2)
plt.legend(['test data above yield','test data below yield','ML yield locus'],␣
 ↪loc=(0.78,0.93))
plt.show()

print('Plot of yield locus and training data in slices of 3D principle stress␣
 ↪space')
mat_mlt.plot_yield_locus(field=True, data=x_train, ref_mat=mat_t, trange=3.e-2,
                  axis1=[0,1,2], axis2=[1,2,0])

print('Plot of trained SVM classification with test data in 2D cylindrical␣
 ↪stress space')
xx, yy = np.meshgrid(np.linspace(-1, 1, 50),np.linspace(-1, 1, 50))
fig, ax  = plt.subplots(nrows=1, ncols=1, figsize=(10,8))
feat = np.c_[yy.ravel(),xx.ravel()]
#Z = mat_mlt.svm_yf.predict(feat)
Z = mat_mlt.svm_yf.decision_function(feat)
```

```
hl = mat_mlt.plot_data(Z, ax, xx*np.pi, (yy+1.)*mat_mlt.sy, c='black')
sc = FE.s_cyl(x_test)
h1 = ax.scatter(sc[:,1], sc[:,0], s=20, c=y_test, cmap=plt.cm.Paired,␣
 ↪edgecolors='k')
ax.set_title('SVC yield function')
ax.set_xlabel(r'$\theta$ (rad)', fontsize=20)
ax.set_ylabel(r'$\sigma_{eq}$ (MPa)', fontsize=20)
ax.tick_params(axis="x", labelsize=16)
ax.tick_params(axis="y", labelsize=16)
```

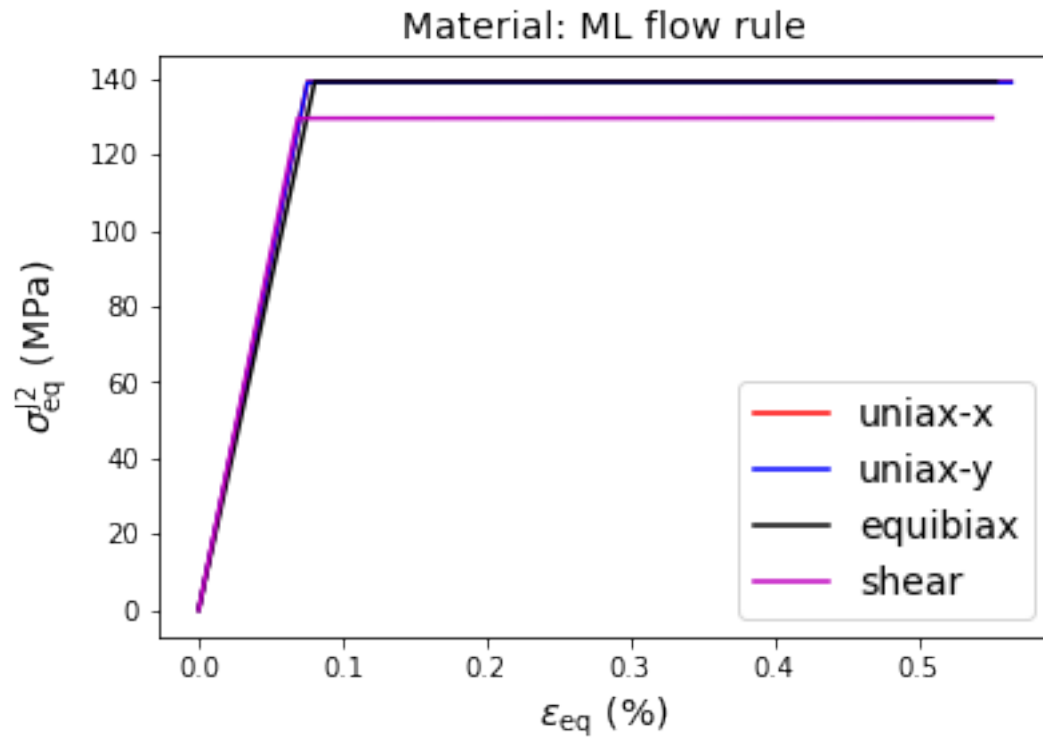
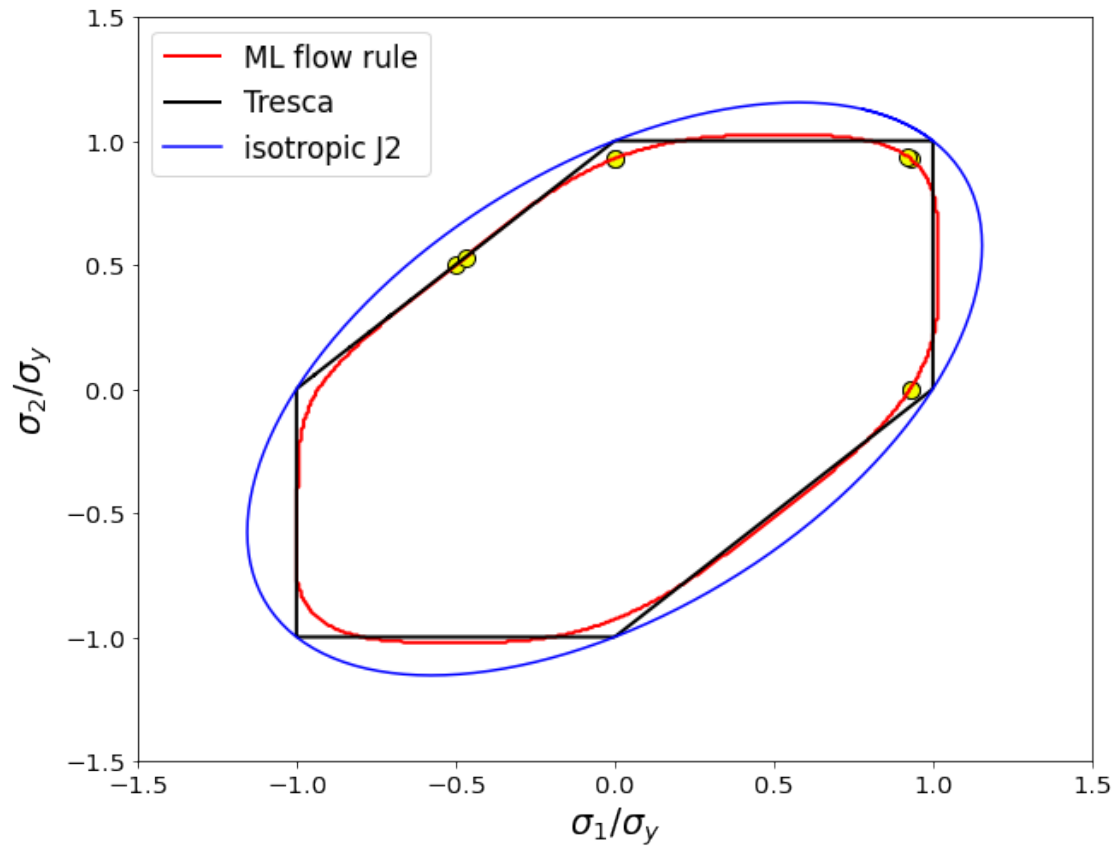
Plot theta vs. Tresca eqiv. stress curves for reference material

Using priciples stresses for training

```
-------------------------

SVM classification fitted
-------------------------

SVC(C=50, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=9.0, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

Training data points (only polar angle): 600 , Test data points: 60
Training set score: 99.19345238095238 %
Test set score: 99.03225806451613 %
r^2 on test data : 0.960800
Number of support vectors generated: 1689
Plot theta vs. J2 eqiv. stress curves for ML material

Plot of yield locus and training data in slices of 3D principle stress space
Plot of trained SVM classification with test data in 2D cylindrical stress space

The properties of the material with the ML yield function are assesed with FEA, and the stress-strain curves and the flow stresses are graphically displayed.

```
[11]: print('\n\n====== Stress-Strain-Curves ======')
      mat_mlt.calc_properties(verb=False, eps=0.005, sigeps=True)
      mat_mlt.plot_stress_strain()

      print('Yield loci and flow stresses in sig_1-sig_2 principle stress space')
      ax = mat_mlt.plot_yield_locus(xstart=-1.5, xend=1.5, ref_mat=mat_t, iso=True,␣
       ↪field=False, Nmesh=400, fontsize=26)
      s=100
      'ML material'
      stx = mat_mlt.sigeps['stx']['sig'][:,0:2]/mat_mlt.sy
      sty = mat_mlt.sigeps['sty']['sig'][:,0:2]/mat_mlt.sy
      et2 = mat_mlt.sigeps['et2']['sig'][:,0:2]/mat_mlt.sy
      ect = mat_mlt.sigeps['ect']['sig'][:,0:2]/mat_mlt.sy
      ax.scatter(stx[1:,0],stx[1:,1],s=s, c='#f0ff00', edgecolors='k')
      ax.scatter(sty[1:,0],sty[1:,1],s=s, c='#f0ff00', edgecolors='k')
      ax.scatter(et2[1:,0],et2[1:,1],s=s, c='#f0ff00', edgecolors='k')
```

```
ax.scatter(ect[1:,0],ect[1:,1],s=s, c='#f0ff00', edgecolors='k')
plt.show()

sect = mat_t.calc_seq(mat_mlt.sigeps['ect']['sig'][1,0:3])
set2 = mat_t.calc_seq(mat_mlt.sigeps['et2']['sig'][1,0:3])
stx  = mat_t.calc_seq(mat_mlt.sigeps['stx']['sig'][1,0:3])
sty  = mat_t.calc_seq(mat_mlt.sigeps['sty']['sig'][1,0:3])
print('\n====== Tresca equivalent stress at yielding ======')
print('Nominal yield strength: ',mat_mlt.sy,'MPa')
print('load case: ',mat_mlt.prop['stx']['name'], '; Tresca equivalent stress:
 ↪',stx.round(decimals=2),'MPa')
print('load case: ',mat_mlt.prop['sty']['name'], '; Tresca equivalent stress:
 ↪',sty.round(decimals=2),'MPa')
print('load case: ',mat_mlt.prop['et2']['name'], '; Tresca equivalent stress:
 ↪',set2.round(decimals=2),'MPa')
print('load case: ',mat_mlt.prop['ect']['name'], '; Tresca equivalent stress:
 ↪',sect.round(decimals=2),'MPa')
```

```
====== Stress-Strain-Curves ======
----------------------------------------------------------
J2 yield stress under uniax-x loading: 139.277 MPa
----------------------------------------------------------
J2 yield stress under uniax-y loading: 139.136 MPa
----------------------------------------------------------
J2 yield stress under equibiax loading: 139.306 MPa
----------------------------------------------------------
J2 yield stress under shear loading: 129.657 MPa
----------------------------------------------------------
```

Yield loci and flow stresses in sig_1-sig_2 principle stress space

```
====== Tresca equivalent stress at yielding ======
Nominal yield strength:  150.0 MPa
load case:  uniax-x ; Tresca equivalent stress:  139.28 MPa
load case:  uniax-y ; Tresca equivalent stress:  139.14 MPa
load case:  equibiax ; Tresca equivalent stress:  139.31 MPa
load case:  shear ; Tresca equivalent stress:  149.72 MPa
```