AdvGeomec-Poli_Renato_HW4

October 5, 2023

2.8 WP4: Solution of Navier's Equation for the Stress Field

2.8.1 Exercise 1: Stresses around a wellbore

Consider a 2D problem of a circular cavity subjected to far field effective stresses $\sigma_{xx} = 12$ MPa and $\sigma_{yy} = 3$ MPa. The diameter of the cavity is 0.2 m. Rock properties: $E = 10$ GPa, $\nu = 0.20$, unconfined compression strength $UCS = 30$ MPa, tensile strength $T_s = 2$ MPa.

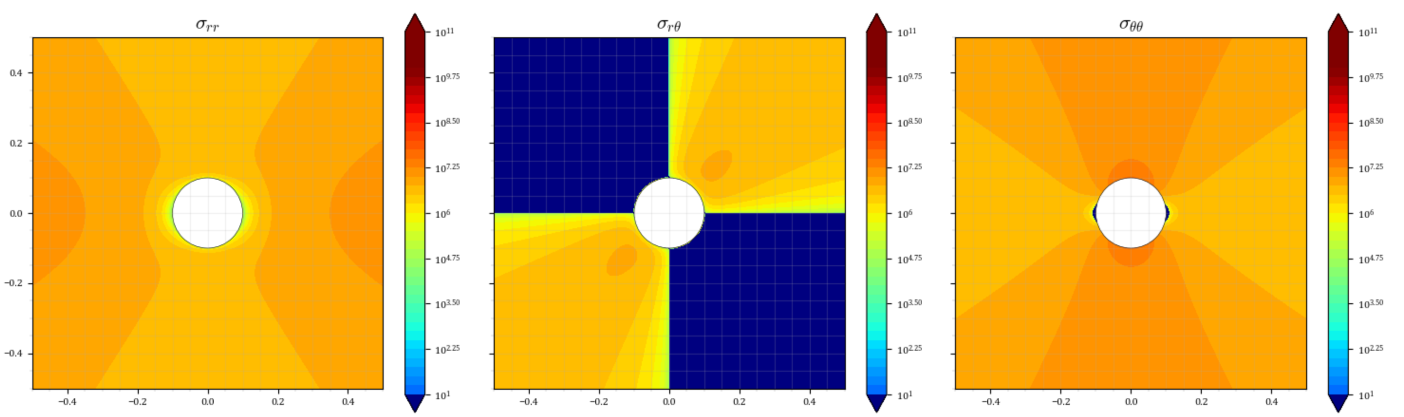
1 -

Using Kirsch equations compute (and plot) σ_{rr}, $\sigma_{\theta\theta}$ and $\sigma_{r\theta}$ for a domain x = [-1m, +1m], and y = [-1m, +1m]. You may define a polar grid for (r, θ). How far does the presence of the wellbore influence stresses?

5]:
```
# Support functions
import numpy as np
import matplotlib.pyplot as plt

def kirsch_rt( r, theta ) :
    global R, pw, s1, s2

    R_div_r = R / r
    sigtt = 1/2 * ( s1 + s2 - 2*pw ) * ( 1 + R_div_r**2 ) \
          - 1/2 * ( s1 - s2 ) * ( 1 + 3 * R_div_r**4 ) * np.cos(2*theta) \
          - pw * R_div_r**2

    sigrt = 1/2 * ( s1 - s2 ) * ( 1 + 2 * R_div_r**2 - 3*R_div_r**4 ) * np.sin(2*theta)


    sigrr = 1/2 * ( s1 + s2 - 2*pw ) * ( 1 - R_div_r**2 ) \
            + 1/2 * ( s1 - s2 ) * ( ( 1 - 4 * R_div_r**2 + 3 * R_div_r**4 )) * np.cos(2*theta) \
            + pw * R_div_r**2

    return sigtt, sigrt, sigrr

def kirsch(x,y) :
    r = np.sqrt( y**2 + x**2 )
    theta = np.arctan2( y , x )
    return kirsch_rt( r, theta )

def plot_well( X, Y, Z, ax, title ) :
    levels = np.array([])
    #levels = np.append(levels, -10**np.linspace(5,1,5))
    levels = np.append(levels, 10**np.linspace(1,11,41))

    # Background color
    import matplotlib.colors as colors
    CB = ax.contourf( X, Y, Z, levels=levels, cmap='jet', extend='both',
                    norm= colors.SymLogNorm(linthresh=0.03, linscale=0.03, vmin=0, vmax=1e10))
    import matplotlib.ticker as ticker
    fmt = ticker.LogFormatterMathtext()
    fmt.create_dummy_axis()
    ax.figure.colorbar(CB, format=ticker.FuncFormatter(fmt) )

    # Well
    from matplotlib.patches import Circle
    circle = Circle((0,0), R, facecolor='w', edgecolor='k', linewidth=.4)
    ax.add_patch(circle)
    ax.set_aspect('equal')

    # Set gadgets
    lim=.5
    ax.set_xlim(-lim,lim)
    ax.set_ylim(-lim,lim)
    ax.set_title(title, fontsize=15)
```

2]:
```python
# Overall setup

import numpy as np
import matplotlib.pyplot as plt

plt.style.use('default')   ## reset!
plt.style.use('paper.mplstyle')

# Stress setup
sig1 = 12e6
sig2 = 3e6
pw=0
s1 = sig1 + pw
s2 = sig2 + pw
# Geometry
R  = 0.2 / 2
# Mechanical parameters
E = 10e9
nu = 0.2
Ts = 2e6
UCS = 30e6
G = E / ( 2 * (1 + nu ))
lame_lambda = E*nu/((1+nu)*(1-2*nu)); # Lame constant
```

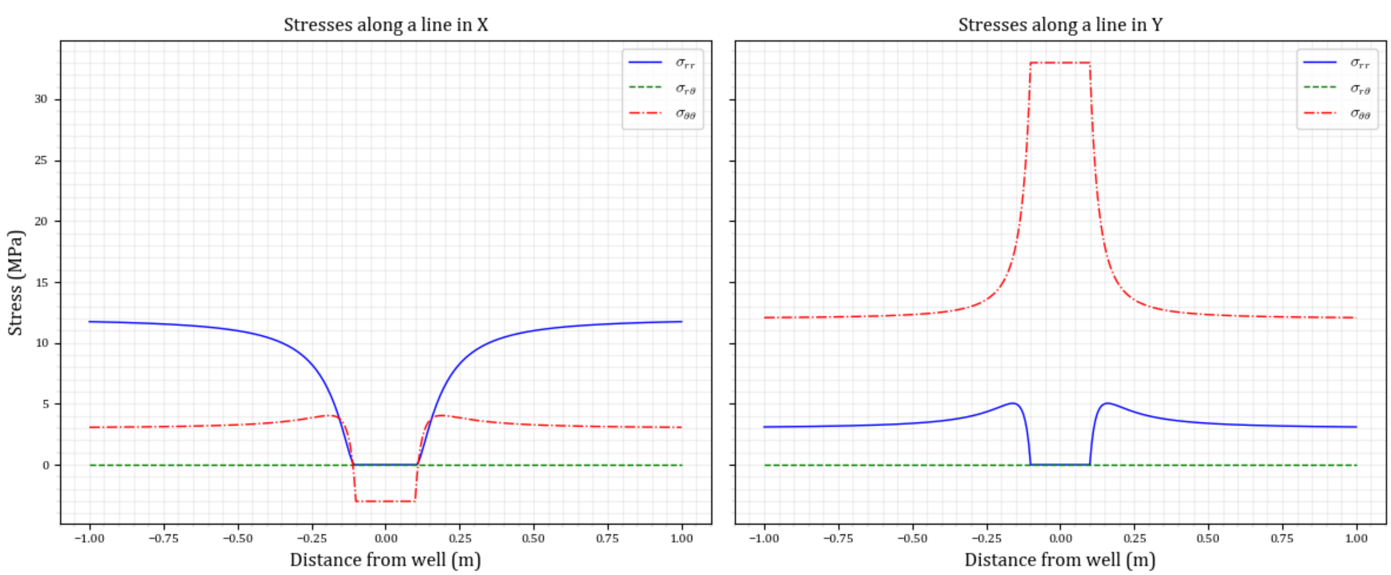
3]:
```python
# PLOT THE FULL DOMAIN

# Points in space
lim = 1
npts = 200
x = np.linspace(-lim,lim,npts)
X, Y = np.meshgrid(x, x)
SIGTT = np.zeros_like( X ) ; SIGRT = np.zeros_like( X ) ; SIGRR = np.zeros_like( X )

# Calculate the kirsch solution in space
for i in np.arange(npts) :
    for j in np.arange(npts) :
        SIGTT[i,j], SIGRT[i,j], SIGRR[i,j] = \
            kirsch(X[i,j],Y[i,j])

# Do the plotting
fig, [ax1,ax2,ax3] = plt.subplots( 1, 3, sharey=True )
fig.set_size_inches(15,4.5)

plot_well( X, Y, SIGRR, ax1, r"$\sigma_{rr}$" )
plot_well( X, Y, SIGRT, ax2, r"$\sigma_{r\theta}$" )
plot_well( X, Y, SIGTT, ax3, r"$\sigma_{\theta\theta}$" )

fig.tight_layout()
fig.savefig(f'kirsch.svg', transparent=True)
```



2 -

Using Kirsch equations compute (and plot) stresses in a line (x = [0.1m, 1m], y = 0 m) and (x = 0 m, y = [0.1 m, 1 m]). Equations in Ch. 6.2 (https://dnicolasespinoza.github.io/)

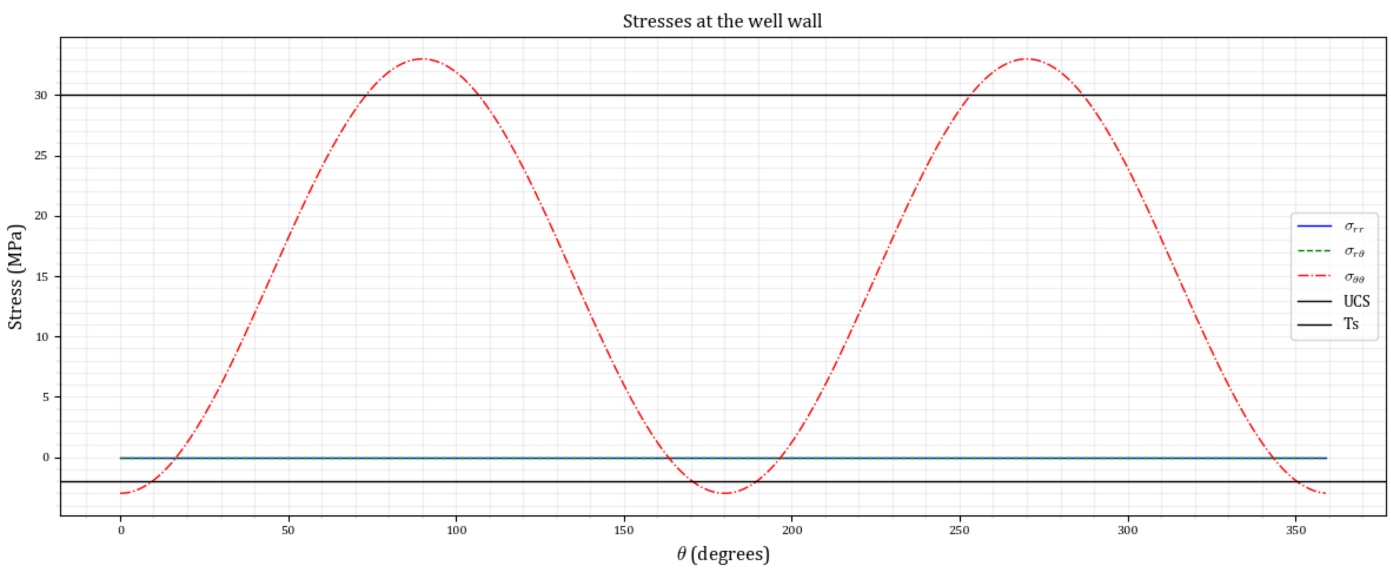
```python
# PLOT STRESS AS A FUNCTION OF DISTANCE TO THE WELL

# Points in space
lim = 1
npts = 150
max_x = 1
X = np.array([ np.linspace( -max_x, -R, npts) , np.linspace( R, max_x, npts ) ]).flatten()
SIGTT = np.zeros( 2*npts )
SIGRT = np.zeros( 2*npts )
SIGRR = np.zeros( 2*npts )

# Calculate the kirsch solution in space
for i in np.arange(len(X)) :
    SIGTT[i], SIGRT[i], SIGRR[i] = kirsch_rt(X[i], 0)

# Do the plotting
fig, [ax1, ax2] = plt.subplots( 1, 2, sharey=True )
fig.set_size_inches(12,5)

ax=ax1

ax.set_yticks( np.linspace(-60, 60, 25) )

ax.plot( X, SIGRR/1e6, '-', label=r"$\sigma_{rr}$" )
ax.plot( X, SIGRT/1e6, '--', label=r"$\sigma_{r\theta}$" )
ax.plot( X, SIGTT/1e6, '-.', label=r"$\sigma_{\theta\theta}$" )

# ax.set_ylim(-1e9, 10)
ax.set_xlabel("Distance from well (m)")
ax.set_ylabel("Stress (MPa)")
ax.set_title("Stresses along a line in X", fontsize=12)
ax.legend()

# Calculate the kirsch solution in space
for i in np.arange(len(X)) :
    SIGTT[i], SIGRT[i], SIGRR[i] = kirsch_rt(X[i], np.pi/2)

ax=ax2
ax.plot( X, SIGRR/1e6, '-', label=r"$\sigma_{rr}$" )
ax.plot( X, SIGRT/1e6, '--', label=r"$\sigma_{r\theta}$" )
ax.plot( X, SIGTT/1e6, '-.', label=r"$\sigma_{\theta\theta}$" )

# ax.set_ylim(-1e9, 10)
ax.set_xlabel("Distance from well (m)")
ax.set_title("Stresses along a line in Y", fontsize=12)
ax.legend()

fig.tight_layout()
fig.savefig(f'kirsch_wall.svg', transparent=True)
```



3 -

Using Kirsch equations compute (and plot) σ_{rr} and $\sigma_{\theta\theta}$ for $r = 0.1$ m. Is there any section* of the rock in shear or tensile failure? Where?

```python
# PLOT STRESS AROUND ON THE WALL OF THE WELL

# Points in space
lim = 1
npts = 360
THETA = np.zeros( npts )
SIGTT = np.zeros( npts )
SIGRT = np.zeros( npts )
SIGRR = np.zeros( npts )

# Calculate the kirsch solution in space
R_ = R
for i in np.arange(npts) :
    theta = 2 * np.pi * (i) / npts
    THETA[i] = theta
    SIGTT[i], SIGRT[i], SIGRR[i] = kirsch_rt(R_, theta)

# Do the plotting
fig, ax = plt.subplots( 1, 1, sharey=True )
fig.set_size_inches(12,5)

ax.plot( THETA*180/np.pi, SIGRR/1e6, '-', label=r"$\sigma_{rr}$" )
ax.plot( THETA*180/np.pi, SIGRT/1e6, '--', label=r"$\sigma_{r\theta}$" )
ax.plot( THETA*180/np.pi, SIGTT/1e6, '-.', label=r"$\sigma_{\theta\theta}$" )
ax.axhline(y=UCS/1e6, label="UCS", c='k')
ax.axhline(y=-Ts/1e6, label="Ts", c='k')

ax.set_xlabel("$\\theta$ (degrees)")
ax.set_ylabel("Stress (MPa)")
ax.set_title("Stresses at the well wall", fontsize=12)
ax.legend()

fig.tight_layout()
fig.savefig(f'kirsch_wall.svg', transparent=True)
```



4 -

Use FreeFEM++ (http://www3.freefem.org/) or FEniCS (https://fenicsproject.org/) to solve the same problem (σ_{xx}, σ_{yy} and σ_{xy}) assuming a domain size 2 m by 2 m. Compute σ_{xx} and σ_{yy} for the same lines as in point (b), and compare with Kirsch's analytical solution. Repeat the process for a domain size 0.5 m by 0.5 m. Are there any differences? Why?

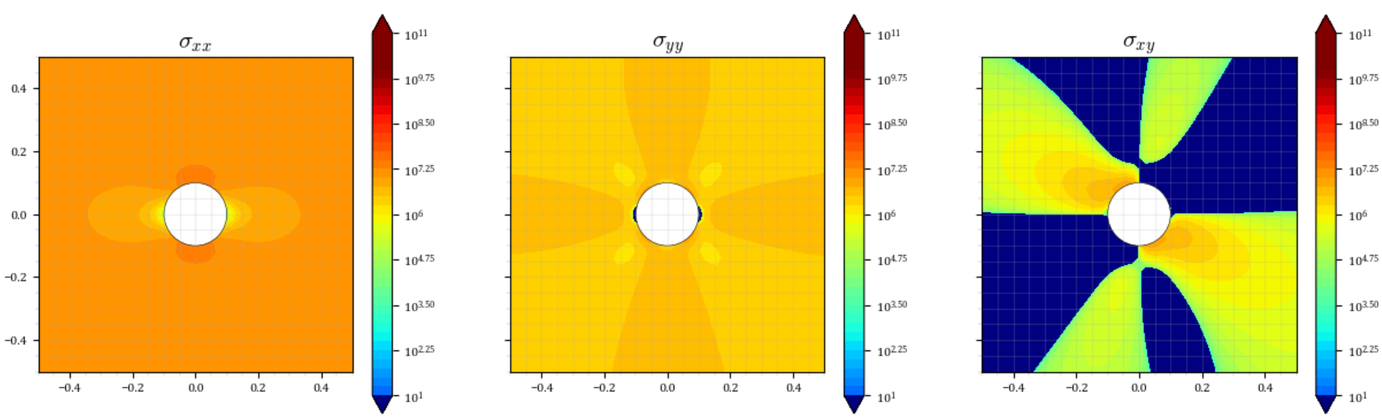
```python
import pandas as pd

df = pd.read_csv( "freefem-kirsch-1.dat" )
```

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

X = df.x
Y = df.y
# Set up a regular grid of interpolation points
xi, yi = np.linspace(X.min(), X.max(), 100), np.linspace(Y.min(), Y.max(), 100)
xi, yi = np.meshgrid(xi, yi)

# Interpolate - invert signal, as frefem++ is getting the wrong convetion
print("Interpolating SIGXX ...")
rbfxx = scipy.interpolate.Rbf(X, Y, df.sigxx, function='linear')
SIGXX = -rbfxx(xi, yi)

print("Interpolating SIGYY ...")
rbfyy = scipy.interpolate.Rbf(X, Y, df.sigyy, function='linear')
SIGYY = -rbfyy(xi, yi)

print("Interpolating SIGXY ...")
rbfxy = scipy.interpolate.Rbf(X, Y, df.sigxy, function='linear')
SIGXY = -rbfxy(xi, yi)

print("Interpolating Ux ...")
rbfux = scipy.interpolate.Rbf(X, Y, df.ux, function='linear')
UX = -rbfux(xi, yi)

print("Interpolating Uy ...")
rbfvy = scipy.interpolate.Rbf(X, Y, df.vy, function='linear')
VY = -rbfvy(xi, yi)
```

4]:

```python
print("Plotting ...")

fig, [ax1,ax2,ax3] = plt.subplots( 1, 3, sharey=True )
fig.set_size_inches(15,4.5)

plot_well( xi, yi, SIGXX, ax1, r"$\sigma_{xx}$" )
plot_well( xi, yi, SIGYY, ax2, r"$\sigma_{yy}$" )
plot_well( xi, yi, SIGXY, ax3, r"$\sigma_{xy}$" )
```

```
Plotting ...
```



5 -

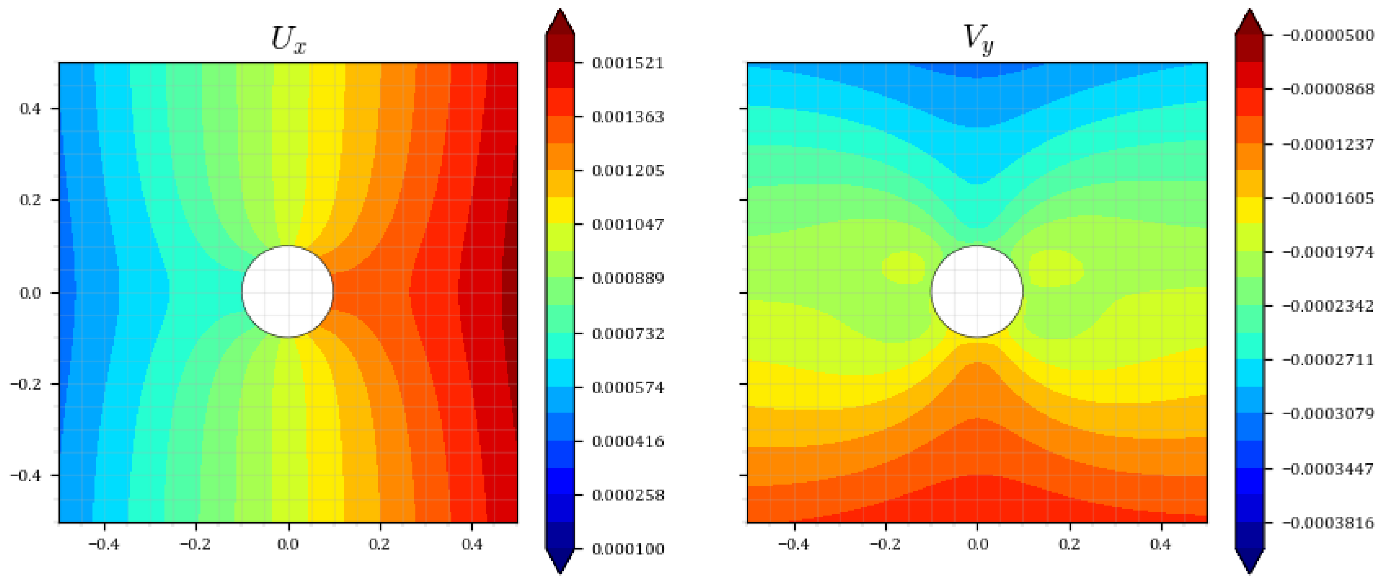
Plot the displacement field.

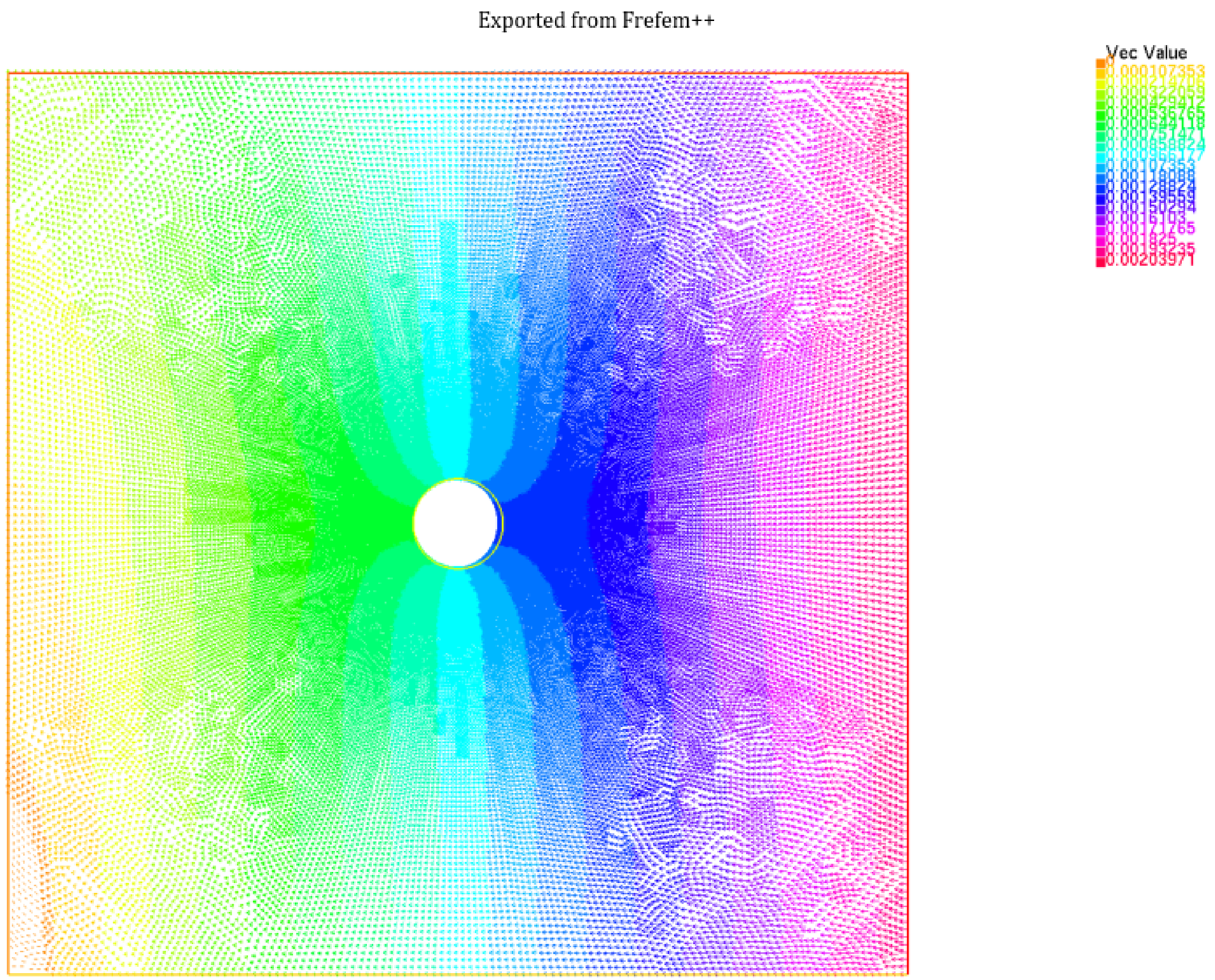
3]:

```python
def plot_displacement( X, Y, Z, ax, title, min, max ) :
    levels = np.array([])
    #levels = np.append(levels, -10**np.linspace(5,1,5))
    levels = np.append(levels, 10**np.linspace(1,11,41))

    # Background color
    import matplotlib.colors as colors
```

```python
    CB = ax.contourf( X, Y, Z, levels=np.linspace(min,max,20), cmap='jet', extend='both')
    import matplotlib.ticker as ticker
    ax.figure.colorbar(CB )

    # Well
    from matplotlib.patches import Circle
    circle = Circle((0,0), R, facecolor='w', edgecolor='k', linewidth=.4)
    ax.add_patch(circle)
    ax.set_aspect('equal')

    # Set gadgets
    lim=.5
    ax.set_xlim(-lim,lim)
    ax.set_ylim(-lim,lim)
    ax.set_title(title, fontsize=15)

fig, [ax4,ax5] = plt.subplots( 1, 2, sharey=True )
fig.set_size_inches(10,4.5)
plot_displacement( xi, yi, UX, ax4, r"$U_x$", 1e-4, 16e-4 )
plot_displacement( xi, yi, VY, ax5, r"$V_y$", -4e-4, -.5e-4 )

from matplotlib import image as mpimg

fig, ax = plt.subplots( 1, 1, sharey=True )
fig.set_size_inches(10,10)

ax.set_title("Exported from Frefem++")
ax.axis('off')
image = mpimg.imread("displacement_field.png")
ax.imshow(image)
```

```
3]: <matplotlib.image.AxesImage at 0x7f58cbf2c220>
```



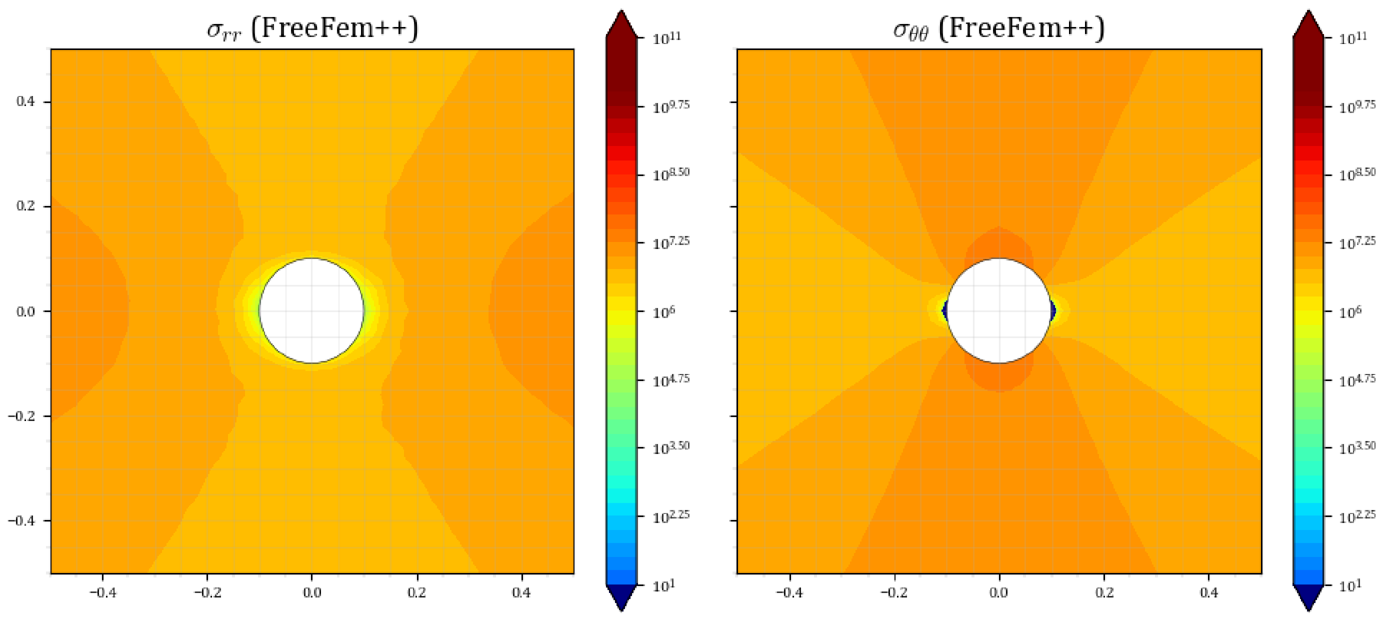


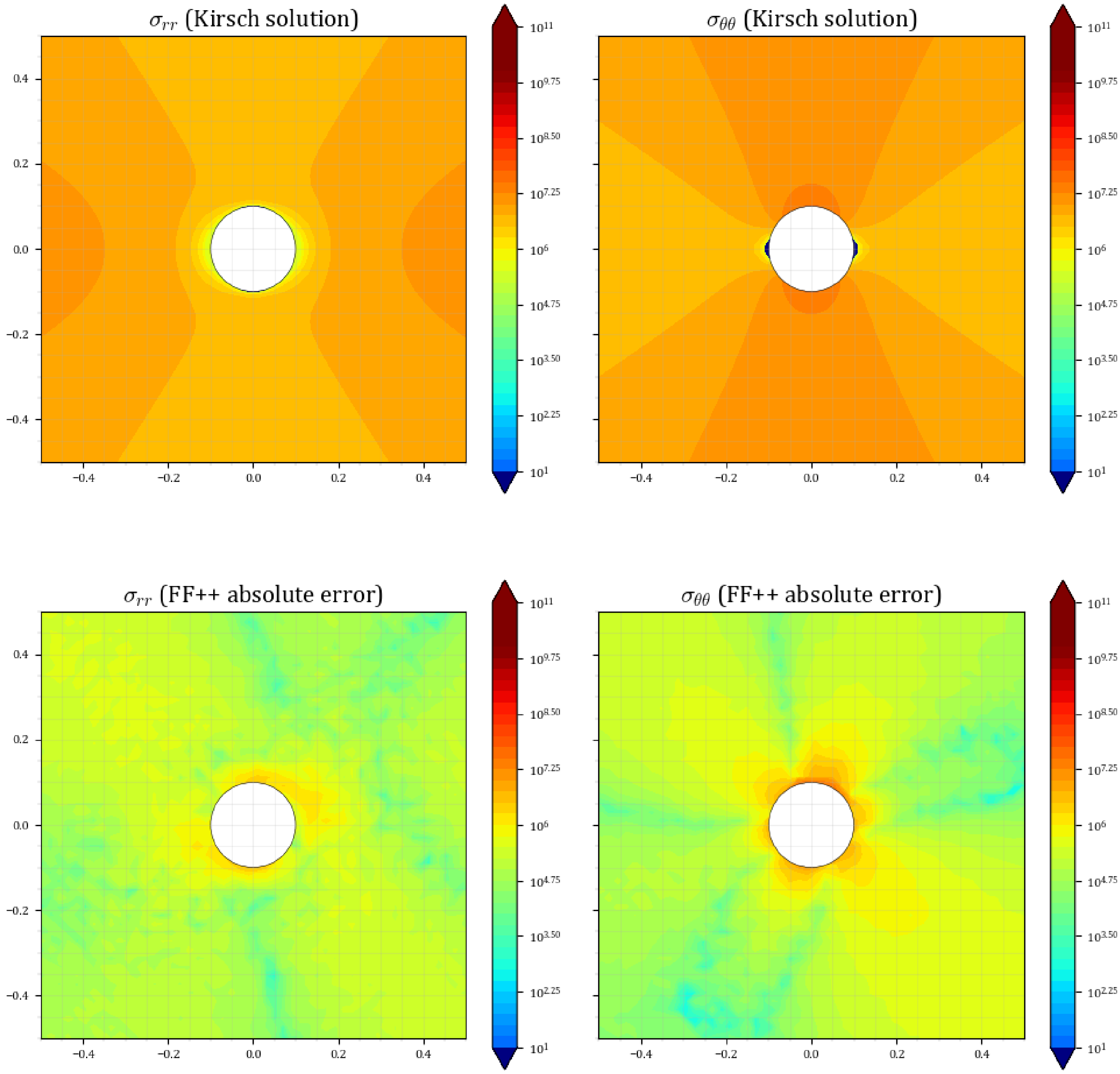
6 -

EXTRA: compute principal stresses within FreeFEM++ and plot σ_{rr} and $\sigma_{\theta\theta}$.

```python
# Rotate stresses from FREEFEM to get sigrr and sigtt

SIGRR_ff = np.zeros_like( SIGXX )
SIGTT_ff = np.zeros_like( SIGXX )

for i in np.arange(len(SIGXX)) :
    for j in np.arange(len(SIGXX[i])) :
        x,y = xi[i,j], yi[i,j]
        sxx, syy, sxy = SIGXX[i,j], SIGYY[i,j], SIGXY[i,j]

        r = np.sqrt( y**2 + x**2 )
        theta = np.arctan2( y , x )
        ROT = np.array( [ [ np.cos(theta), -np.sin(theta) ], [ np.sin(theta), np.cos(theta) ] ] )
        SIGij = np.array( [ [ sxx, sxy ] , [ sxy, syy ] ] )
        SIGmn = ROT.T @ SIGij @ ROT
        SIGRR_ff[i,j] = SIGmn[0,0]
        SIGTT_ff[i,j] = SIGmn[1,1]

fig, [ax1,ax2] = plt.subplots( 1, 2, sharey=True )
fig.set_size_inches(10,4.5)
plot_well( xi, yi, SIGRR_ff, ax1, r"$\sigma_{rr}$ (FreeFem++)" )
plot_well( xi, yi, SIGTT_ff, ax2, r"$\sigma_{\theta\theta}$ (FreeFem++)" )
fig.tight_layout()
```

```python
#
# Compare to the analyticallim = 1
#

npts = 100
x = np.linspace(-lim,lim,npts)
X, Y = np.meshgrid(x, x)
SIGTT = np.zeros_like( X ) ; SIGRT = np.zeros_like( X ) ; SIGRR = np.zeros_like( X )

# Calculate the kirsch solution in space
for i in np.arange(npts) :
    for j in np.arange(npts) :
        SIGTT[i,j], SIGRT[i,j], SIGRR[i,j] = \
            kirsch(X[i,j],Y[i,j])

# Do the plotting
fig, [ax1,ax2] = plt.subplots( 1, 2, sharey=True )
fig.set_size_inches(10,4.5)
plot_well( X, Y, SIGRR, ax1, r"$\sigma_{rr}$ (Kirsch solution)" )
plot_well( X, Y, SIGTT, ax2, r"$\sigma_{\theta\theta}$ (Kirsch solution)" )
fig.tight_layout()

#
# Calculate the errors
#
TT_err = np.abs( SIGTT_ff - SIGTT )
RR_err = np.abs( SIGRR_ff - SIGRR )
fig, [ax1,ax2] = plt.subplots( 1, 2, sharey=True )
fig.set_size_inches(10,4.5)
plot_well( X, Y, RR_err, ax1, r"$\sigma_{rr}$ (FF++ absolute error)" )
plot_well( X, Y, TT_err, ax2, r"$\sigma_{\theta\theta}$ (FF++ absolute error)" )
fig.tight_layout()
```

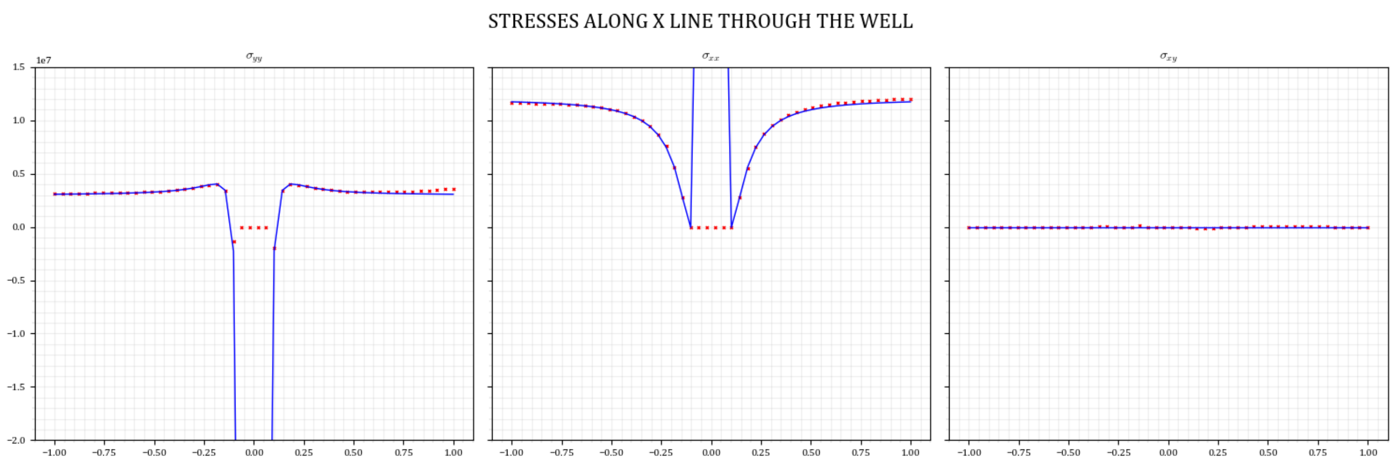





```
x = np.linspace(-1,1,50)
y = np.zeros_like(x)

# Invert signal as freefem is delivering the wrong convention
syy = -rbfyy(x,y)
sxx = -rbfxx(x,y)
sxy = -rbfxy(x,y)

# Analytical
a_stt = np.zeros_like(x)
a_srr = np.zeros_like(x)
a_srt = np.zeros_like(x)
for i in np.arange(len(x)) :
    a_stt[i], a_srt[i], a_srr[i] = kirsch(x[i], y[i])

fig, [ax1,ax2,ax3] = plt.subplots(1,3, sharey=True)
fig.set_size_inches(15,5)
ax1.scatter(x,syy, marker='x', s=5, c='r')
ax1.plot(x,a_stt)
```

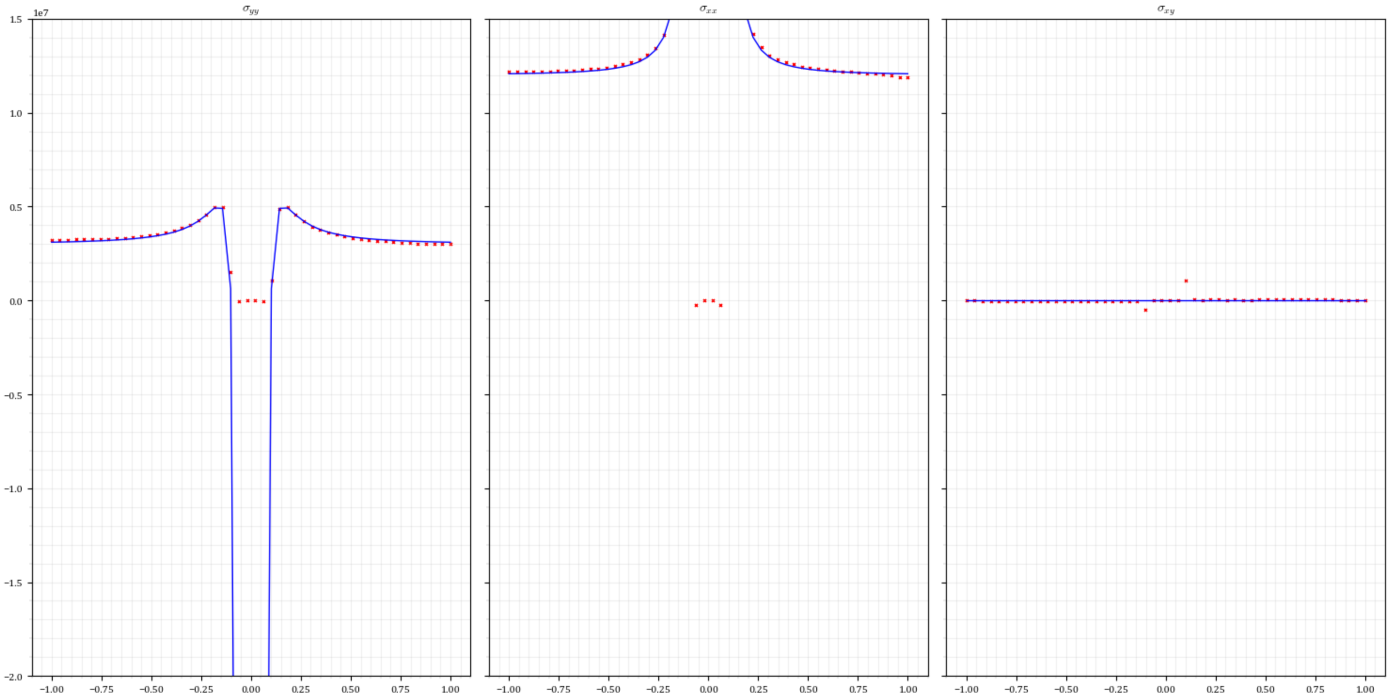
```
ax1.set_title("$\sigma_{yy}$")

ax2.scatter(x,sxx, marker='x', s=5, c='r')
ax2.plot(x,a_srr)
ax2.set_title("$\sigma_{xx}$")

ax3.scatter(x,sxy, marker='x', s=5, c='r')
ax3.plot(x,a_srt)
ax3.set_title("$\sigma_{xy}$")

fig.suptitle('STRESSES ALONG X LINE THROUGH THE WELL', fontsize=16)

fig.tight_layout()

ax1.set_ylim(-2e7, 1.5e7)
```

4]: (-20000000.0, 15000000.0)



9]:
```
y = np.linspace(-1,1,50)
x = np.zeros_like(x)

syy = rbfyy(x,y)
sxx = rbfxx(x,y)
sxy = rbfxy(x,y)

# Analytical
a_stt = np.zeros_like(x)
a_srr = np.zeros_like(x)
a_srt = np.zeros_like(x)
for i in np.arange(len(x)) :
    a_stt[i], a_srt[i], a_srr[i] = kirsch(x[i], y[i])

fig, [ax1,ax2,ax3] = plt.subplots(1,3, sharey=True)
fig.set_size_inches(15,8)
ax1.scatter(y,-syy, marker='x', s=5, c='r')
ax1.plot(y,a_srr)
ax1.set_title("$\sigma_{yy}$")

ax2.scatter(y,-sxx, marker='x', s=5, c='r')
ax2.plot(y,a_stt)
ax2.set_title("$\sigma_{xx}$")

ax3.scatter(y,sxy, marker='x', s=5, c='r')
ax3.plot(y,a_srt)
ax3.set_title("$\sigma_{xy}$")

ax1.set_ylim(-2e7, 1.5e7)
fig.suptitle('STRESSES ALONG Y LINE THROUGH THE WELL', fontsize=16)
fig.tight_layout()
```

STRESSES ALONG Y LINE THROUGH THE WELL



2.8.2 Exercise 2: Stresses around a planar fracture

Consider a 2D problem of an elliptical fracture (half-length $c = 10$ m). Solve the problem using just half of the domain. Set the fracture along the left boundary of a domain: $x =$ [0 m, 100 m] and $y =$ [-50 m, 50 m], with fracture center at $(x, y) =$ (0,0) m. This boundary will have a pressure boundary condition. All other boundaries will have zero displacement. Rock properties: $E = 30$ GPa, $\nu = 0.20$.

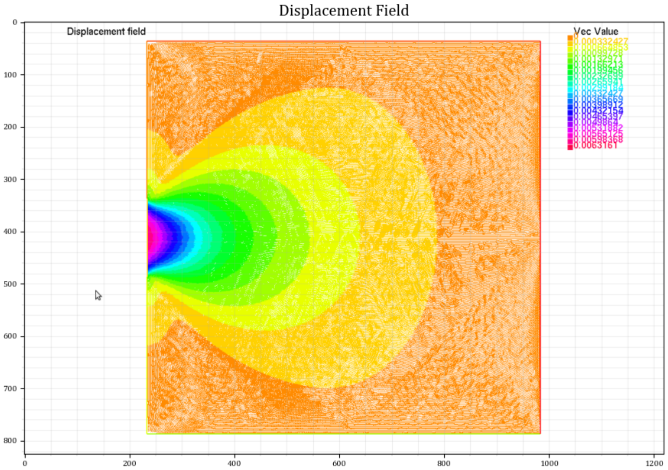
1 -

Use FreeFEM++ (http://www3.freefem.org/) or FEniCS (https://fenicsproject.org/) to solve for σ_{xx}, σ_{yy} and σ_{xy} imposing a fracture pressure $p = 10$ MPa. Plot results.

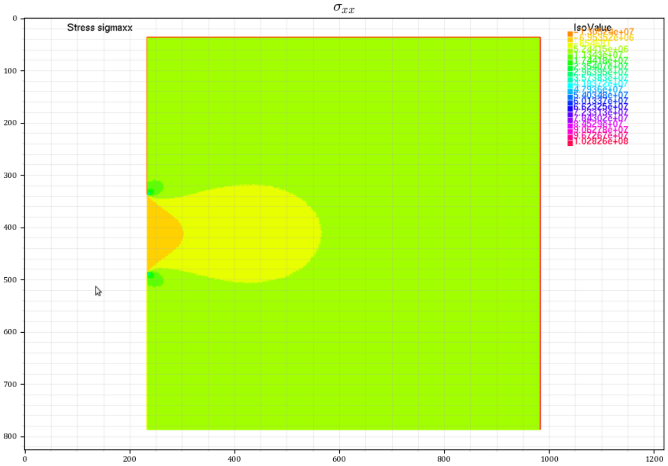
0]:

```
imgs = [
    { 't': 'Displacement Field', 'f': "Displacement" },
    { 't': '$\sigma_{xx}$',      'f': "SigXX" },
    { 't': '$\sigma_{xy}$',      'f': "SigXY" },
    { 't': '$\sigma_{yy}$',      'f': "SigYY" },
]

fig, _axs = plt.subplots(4, 1, figsize=[15,25])
axs = _axs.flatten()

from matplotlib.image import imread
for e, ax in zip( imgs, axs ) :
    ax.imshow( imread( f"png/{e['f']}.png" ) )
    ax.set_title( e['t'], fontsize=15 )

fig.tight_layout()
```

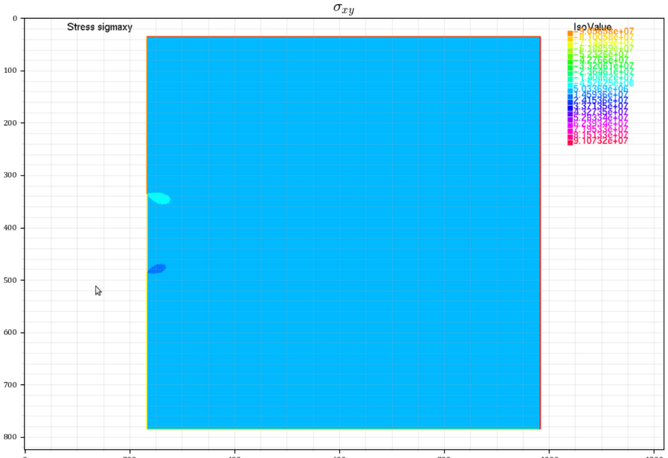
Displacement Field



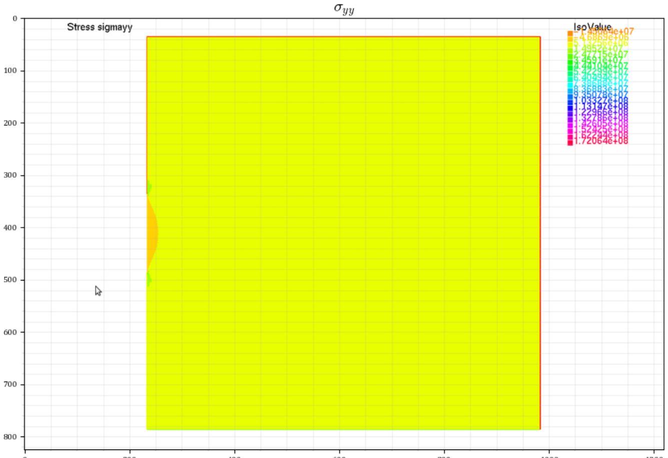
σ_{xx}



σ_{xy}



σ_{yy}

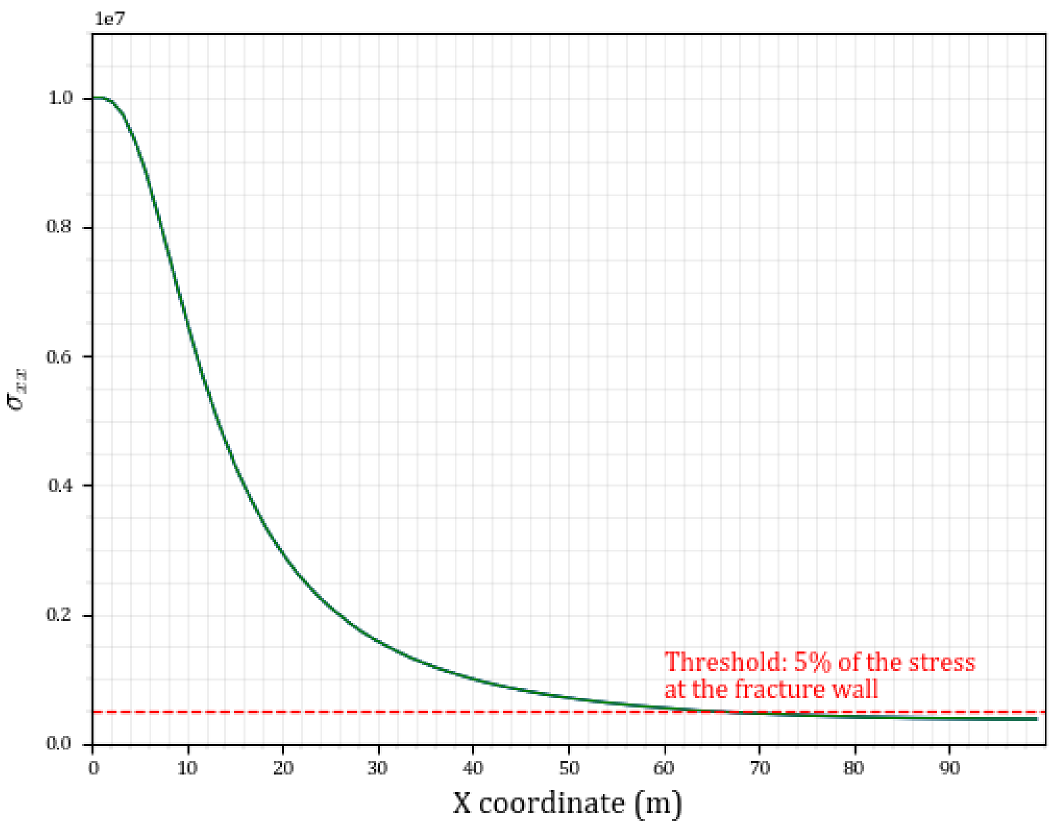


2 -

Export and plot stress perpendicular to the fracture direction σ_{xx} at the middle of the fracture (L1 = (x = [0, 100 m], y = 0 m), Figure 2.32). How far does the influence of the fracture extend?

1]:
```
import pandas as pd

df = pd.read_csv("FractureCenter.dat");
df["perc_inf"] = -df.sigxx / 1e7 * 100
fig,ax = plt.subplots()
ax.plot(df.x,-df.sigxx)
ax.plot(df.x,-df.sigxx)

th = 5e5
ax.text( 60, th*1.4, "Threshold: 5% of the stress\nat the fracture wall" , c='r')
ax.axhline(y = th, color = 'r', linestyle = '--')

ax.set_xticks( np.arange( 0, 100, 10 ))
ax.set_xlim(0,100)
ax.set_ylim(0,1.1e7)
ax.set_xlabel("X coordinate (m)")
ax.set_ylabel("$\sigma_{xx}$")
```

1]:
```
Text(0, 0.5, '$\\sigma_{xx}$')
```



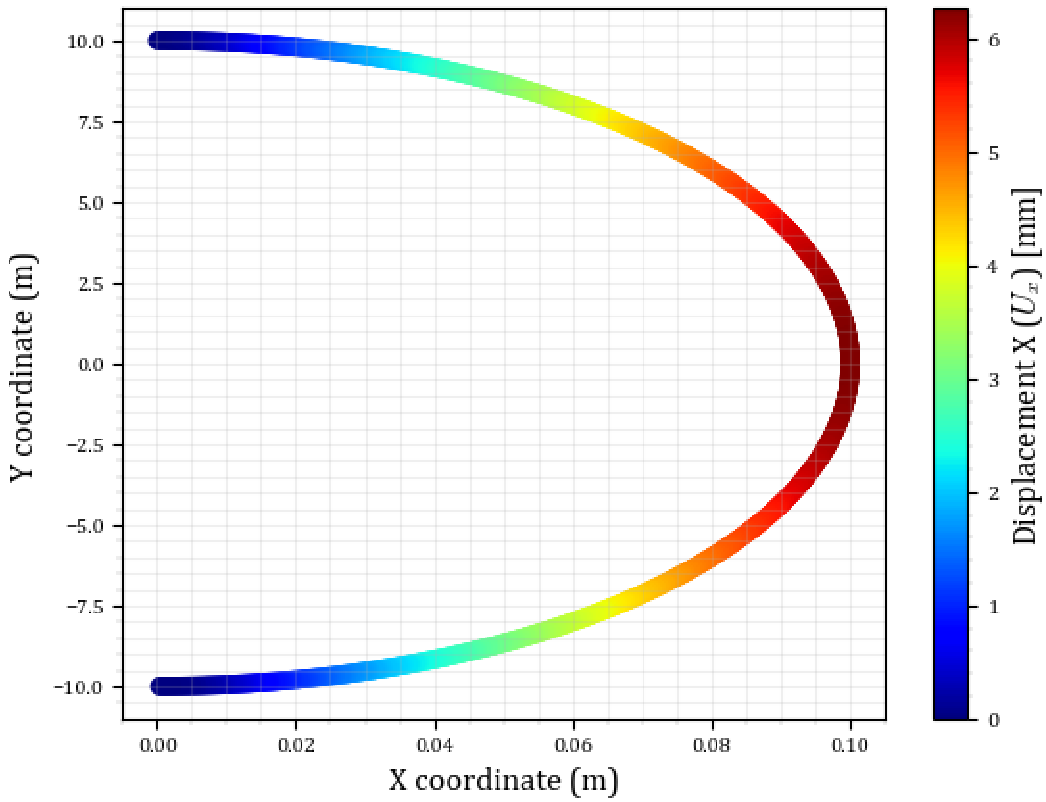
3 -

Plot x-displacements at the face of the fracture. Compare with analytical equation. Equations in Ch. 7.3.2 (https://dnicolasespinoza.github.io/).

1]:
```
import pandas as pd

df = pd.read_csv("FractureWall.dat");
fig,ax = plt.subplots()
cb = ax.scatter(df.x, df.y, c=df.u1 * 1e3, cmap='jet')
```

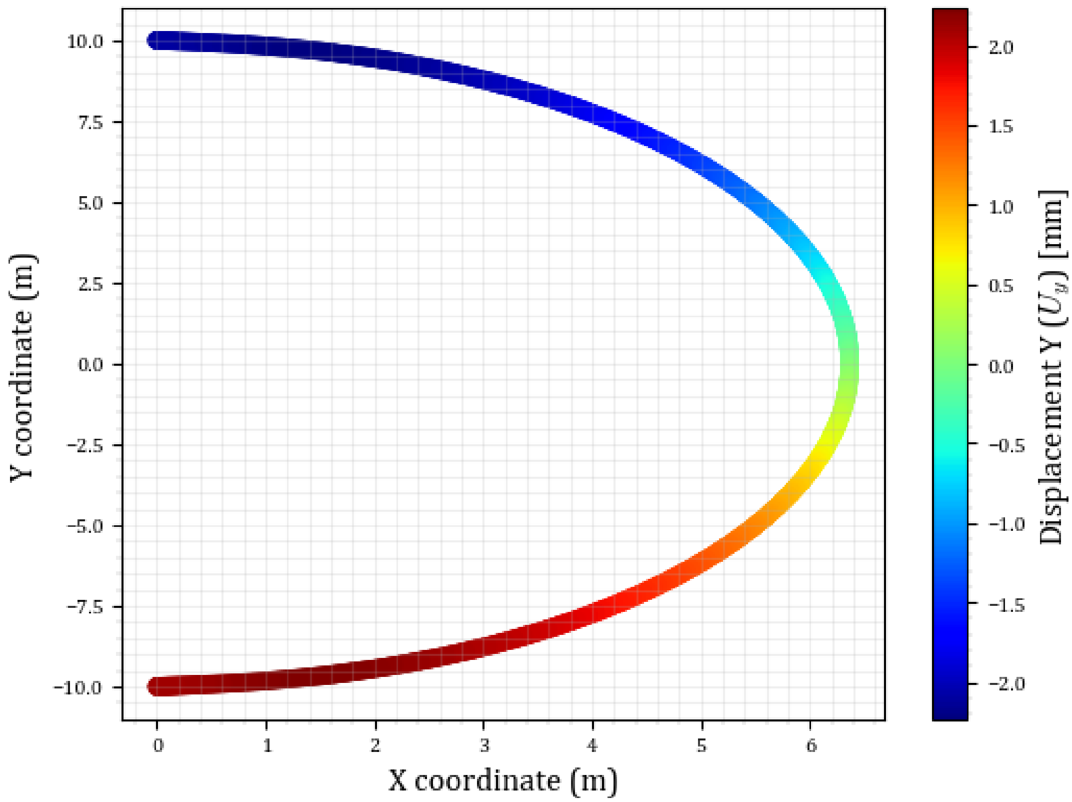
```
cb = fig.colorbar(cb)
cb.set_label("Displacement X ($U_x$) [mm]")

ax.set_xlabel("X coordinate (m)")
ax.set_ylabel("Y coordinate (m)")
```

1]: Text(0, 0.5, 'Y coordinate (m)')



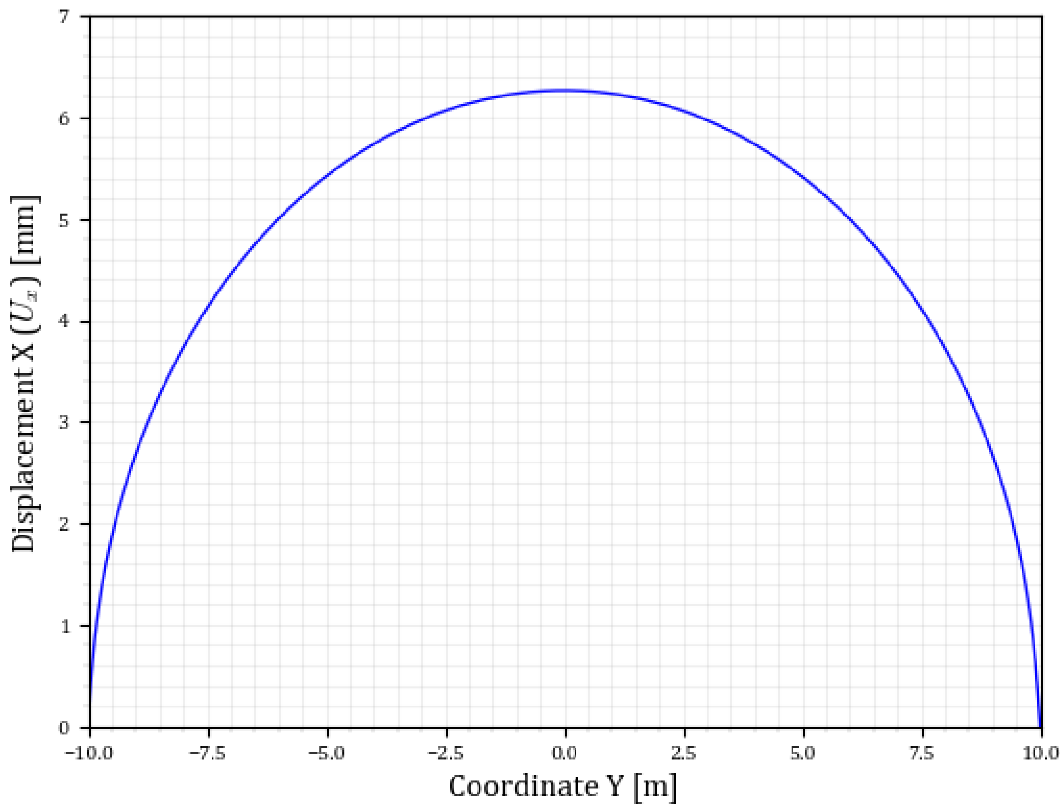
7]:
```
import pandas as pd

df = pd.read_csv("FractureWall.dat");
fig,ax = plt.subplots()
cb = ax.scatter(df.x + 1e3*df.u1, df.y, c=df.u2 * 1e3, cmap='jet')
cb = fig.colorbar(cb)
cb.set_label("Displacement Y ($U_y$) [mm]")

ax.set_xlabel("X coordinate (m)")
ax.set_ylabel("Y coordinate (m)")
```

7]: Text(0, 0.5, 'Y coordinate (m)')



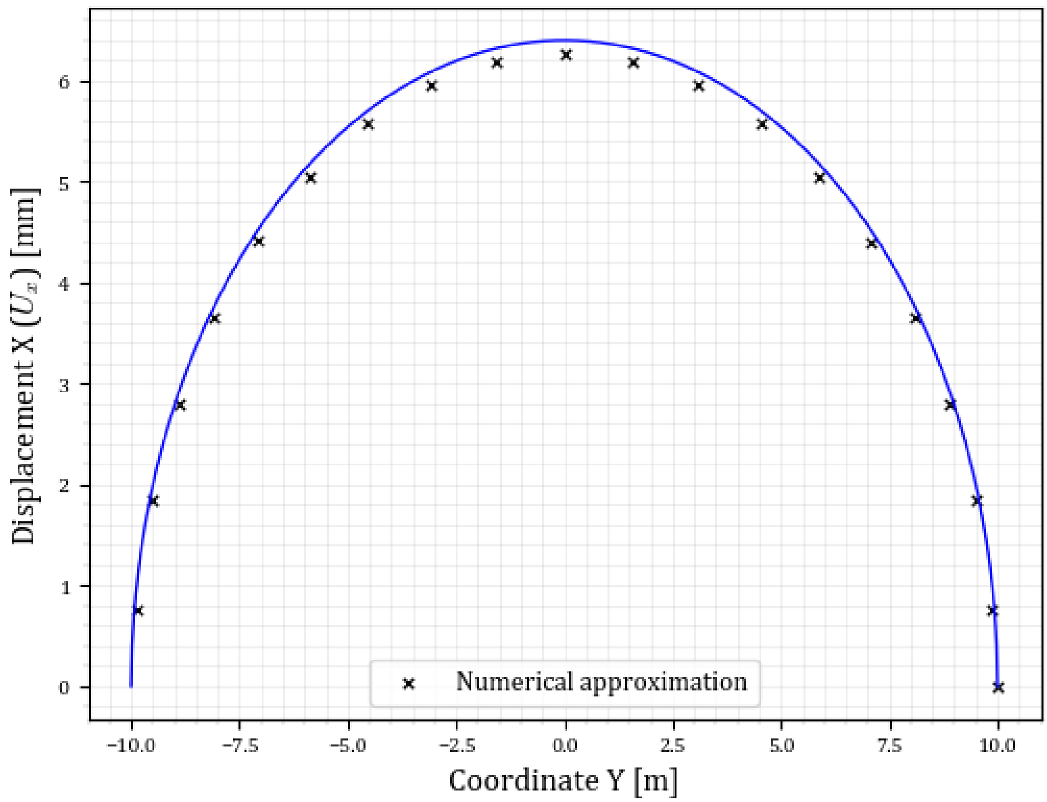
8]:
```
import pandas as pd

df = pd.read_csv("FractureWall.dat");
fig,ax = plt.subplots()
ax.plot(df.y, df.u1 * 1e3 )

ax.set_xlabel("Coordinate Y [m]")
ax.set_ylabel("Displacement X ($U_x$) [mm]")

ax.set_xlim(-10,10)
ax.set_ylim(0,7)
```

8]: (0.0, 7.0)



9]:
```
# Analytical solution
import numpy as np
Pfrat = 1e7

E=30E9
nu=0.2
c = 10

X = np.linspace( -c, c, 1000 )
A_U1 = np.zeros_like(X)
for i in np.arange(len(X)) :
    A_U1[i] = 2 * Pfrat / E * ( 1 - nu**2 ) * np.sqrt( c**2 - X[i]**2 )

fig,ax = plt.subplots()
ax.plot(X, A_U1 * 1e3 )

# Numerical solution
ax.scatter(df[::50].y, df[::50].u1 * 1e3, marker='x' , c='k' , s=15, label='Numerical approximation')

ax.legend()

ax.set_xlabel("Coordinate Y [m]")
ax.set_ylabel("Displacement X ($U_x$) [mm]")
```

9]:
```
Text(0, 0.5, 'Displacement X ($U_x$) [mm]')
```



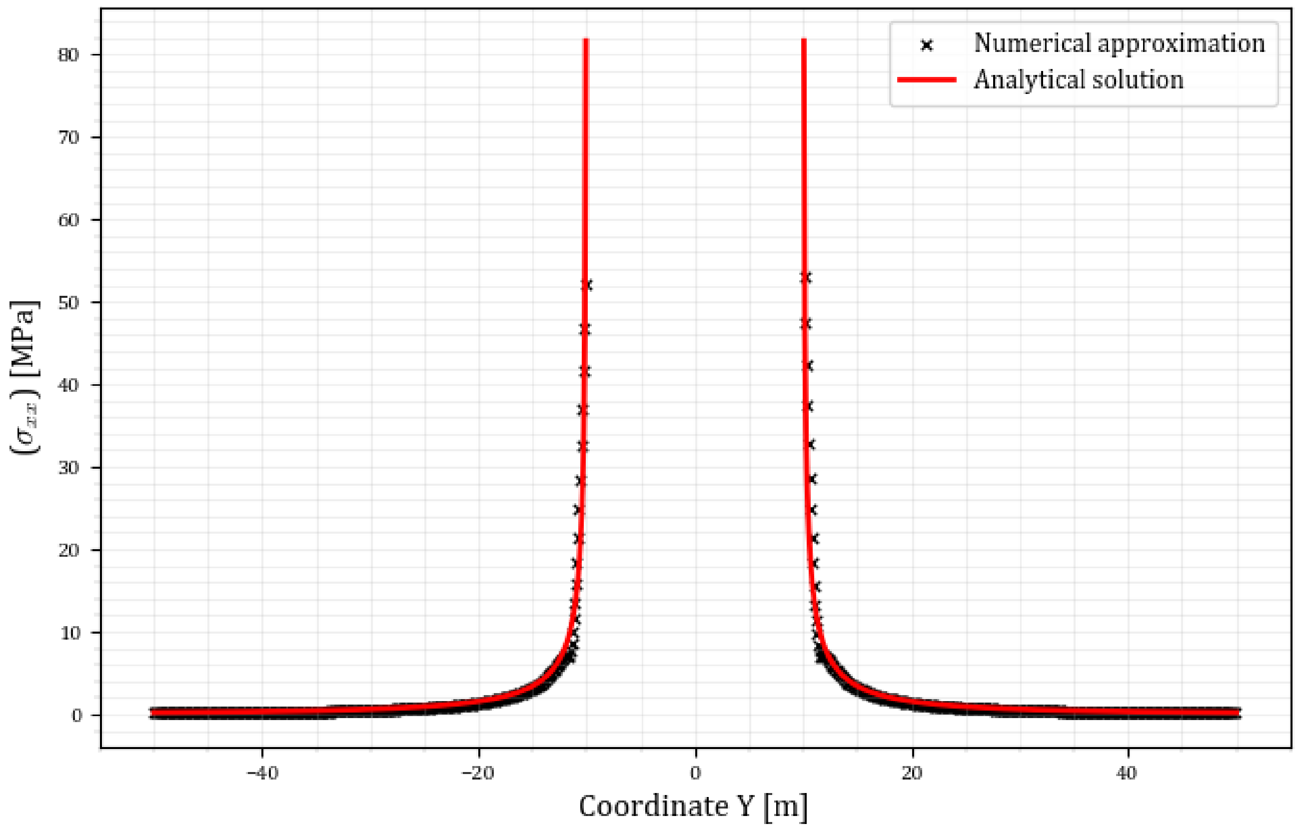
4 -

Plot σ_{xx} along fracture length and beyond fracture tips (line L2 = (x = 0 m, y = [-50, 50]) m, Figure 2.32) and compare with analytical Griffith solution.

4]:
```python
# Load Numerical

import pandas as pd

fig,ax = plt.subplots()
fig.set_size_inches(8,5)
df = pd.read_csv("Along_X_equals_0.dat");

# Analytical
ylim = 50
X = np.linspace( -ylim, ylim, 1000 )
A_SXX = np.zeros_like(X)
A_U1 = np.zeros_like(X)

for i in np.arange(len(X)) :
    A_SXX[i] = np.nan
    A_U1[i] = np.nan

    if ( X[i]**2 < c**2 ) :
        A_U1[i] = 2 * Pfrat / E * ( 1 - nu**2 ) * np.sqrt( c**2 - X[i]**2 )

    if ( X[i]**2 > c**2 ) :
        A_SXX[i] = Pfrat * ( np.abs(X[i]) / np.sqrt(X[i]**2 - c**2) - 1 )

# Numerical solution
dff = df
dff = dff[ dff.y**2 > c**2 ]
ax.scatter(dff.y,dff.sxx/1e6, marker='x' , c='k' , s=15, label='Numerical approximation')
# Analytical solution
ax.plot(X, A_SXX/1e6, c='r', linewidth=2, label='Analytical solution')

ax.legend()

ax.set_xlabel("Coordinate Y [m]")
```

```
ax.set_ylabel("($\sigma_{xx}$) [MPa]")
```

4]: Text(0, 0.5, '($\\sigma_{xx}$) [MPa]')



Appendix A

WELL code for FreeFem++

```
//----------------------------------------------------------------------
// Dimensions
real ySize = 2. ;  // y-size of the domain
real xSize = 2. ;  // x-size of the domain
real R = 0.1 ;      // wellbore radius

// Elastic constants
real E = 1e10 ;     // Young's modulus
real nu = 0.3 ;     // Poisson's ratio

real G = E/(2*(1+nu )) ; // shear modulus
real lambda = E*nu/((1+nu)*(1-2*nu)) ; // Lame constant

//Stresses
real Sx = 12e6 ;
real Sy = 3e6 ;
real Pwell = 0.0e6;


//----------------------------------------------------------------------
// First define boundaries
border Right(t=-ySize/2,ySize/2){x=xSize/2;y=t;}
border Top(t=xSize/2,-xSize/2){x=t;y=ySize/2;}
border Left(t=ySize/2,-ySize/2){x=-xSize/2;y=t;}
border Bottom(t=-xSize/2,xSize/2){x=t;y=-ySize/2;}
border Well(t=0,-2*pi){x=R*cos(t);y=R*sin(t);}

//SHOW DOMAIN
plot( Right(10)+Top(10)+Left(10)+Bottom(10) + Well(40), wait=true);


//----------------------------------------------------------------------
// Create mesh
int n = 40; // number of mesh nodes on the outer borders
int nwell = 100; // number of mesh nodes on wellbore
mesh Omega = buildmesh (Right(n)+Top(n)+Left(n)+Bottom(n)+Well(nwell));

plot(Omega, wait=true);

// FE spaces
fespace Displacement(Omega, P2); // linear shape functions
fespace Stress(Omega, P1); // piecewise constants

Displacement u1, u2, v1, v2;
Stress sigmaxx, sigmayy, sigmaxy;


//----------------------------------------------------------------------
// definition of 2 macros :
// macro for strain
macro e(u1,u2)
    [
        dx(u1),
        (dy(u1)+dx(u2))/2 ,
        (dx(u2)+dy(u1))/2 ,
        dy(u2)
    ]//eps_xx, eps_xy , eps_yx , eps_yy
// macro for stress
macro sigma(u1,u2)
    [
        (lambda+2.*G)*e(u1,u2)[0]+lambda*e(u1,u2)[3],
        2.*G*e(u1,u2)[1],
        2.*G*e(u1,u2)[2],
        lambda*e(u1,u2)[0]+(lambda+2.*G)*e(u1,u2)[3]
    ] //stress s_xx, s_xy, s_yx, s_yy


    //  Define  system  of  equations
problem Elasticity([u1,u2],[v1,v2]) =
    int2d(Omega)(sigma(u1,u2)'*e(v1,v2))
    // Boundary conditions
    + on(Left,u1=0)              // Dirichlet boundary conditions
```

```
    + on(Bottom,u2=0)
    + int1d(Omega,Right)(Sx*v1)  // Neumann boundary conditions
        //- int1d(Omega,Left)(Sx*v1)
    + int1d(Omega,Top)(Sy*v2)
        //- int1d(Omega,Bottom)(Sy*v2)
    + int1d(Omega,Well)(Pwell*(N.x*v1+N.y*v2))
    ;


//----------------------------------------------------------------------
//  Solve system
Elasticity;


// Stresses
sigmaxx = sigma(u1,u2)[0];
sigmayy = sigma(u1,u2)[3];
sigmaxy = sigma(u1,u2)[1];  // we could use [2] as  well


//----------------------------------------------------------------------
// plot on the deformed surface
mesh Th=movemesh(Omega,[x+10*u1,y+10*u2]);
plot(Th,cmm="Deformed configuration",wait=1);


// plot the deformation field and stress
plot([u1,u2],coef=10,cmm="Displacement field",wait=1,value=true);
plot(sigmaxx,fill=1, cmm="Stress sigmaxx",wait=1,value=true);


//write files
ofstream ff("output.dat");
ff << "x,y,sigxx,sigyy,sigxy,ux,vy" << endl;
for(int i=0;i<100;i++) {
for(int j=0;j<100;j++) {
    // x, y, Sxx, Syy, Sxy
        real xline = -xSize/2 + xSize*i/100.;
        real yline = -ySize/2 + ySize*j/100.;
    // Analytical solution
    //write file numerical and analytical solution
    ff<< xline <<", "<< yline
        <<", "<< sigmaxx(xline,yline)
        <<", "<< sigmayy(xline,yline)
        <<", "<< sigmaxy(xline,yline)
        <<", "<< u1(xline,yline)
        <<", "<< u2(xline,yline)
        <<endl;
}
}
```

Appendix B

FreeFem++ Code for the fracture

```
//-------------------------------------------------------------------
// Dimensions
real xa = 0. ;
real xb = 100. ;  // x-size of the domain

real ya = -50. ;
real yb = 50. ;  // y-size of the domain

real xSize = xb - xa ;
real ySize = yb - ya ;

// Elastic constants
real E = 30e9 ;     // Young's modulus
real nu = 0.2 ;     // Poisson's ratio

real G = E/(2*(1+nu )) ; // shear modulus
real lambda = E*nu/((1+nu)*(1-2*nu)) ; // Lame constant

//Stresses
real Sx = 12e6 ;
real Sy = 3e6 ;
real Pfrac = 10.0e6;

// FRACTURE
real xf = 10; // fracture half-length
real fw = .1; // fracture half-width


//-------------------------------------------------------------------
// First define boundaries
border Right(t=ya, yb){x=xb;y=t;}
border Top(t=xb, xa){x=t;y=yb;}
border Left1(t=yb,(ya+ySize/2+xf)){x=xa;y=t;}
border Frac(t = -pi/2, pi/2) {x = fw*cos(t); y = xf*sin(-t);}
border Left2(t=(ya+ySize/2-xf), ya){x=xa;y=t;}
border Bottom(t=xa,xb){x=t;y=ya;}


//SHOW DOMAIN
plot( Right(10)+Top(10)+Left1(10), Left2(10) +Bottom(10) +Frac(40), wait=true);


//-------------------------------------------------------------------
// Create mesh
int n = 30; // number of mesh nodes on the outer borders
int nfrac = 30; // number of mesh nodes on wellbore
mesh Omega = buildmesh (Right(n)+Top(n)+Left1(n/2)+Left2(n/2)+Bottom(n)+Frac(nfrac));

plot(Omega, wait=true);

// FE spaces
fespace Displacement(Omega, P2); // linear shape functions
fespace Stress(Omega, P2); // piecewise constants

Displacement u1, u2, v1, v2;
Stress sigmaxx, sigmayy, sigmaxy;


//-------------------------------------------------------------------
// definition of 2 macros :
// macro for strain
macro e(u1,u2)
    [
        dx(u1),
        (dy(u1)+dx(u2))/2 ,
        (dx(u2)+dy(u1))/2 ,
        dy(u2)
    ]//eps_xx, eps_xy , eps_yx , eps_yy
// macro for stress
macro sigma(u1,u2)
    [
        (lambda+2.*G)*e(u1,u2)[0]+lambda*e(u1,u2)[3],
```

```
        2.*G*e(u1,u2)[1],
        2.*G*e(u1,u2)[2],
        lambda*e(u1,u2)[0]+(lambda+2.*G)*e(u1,u2)[3]
    ] //stress s_xx, s_xy, s_yx, s_yy


    // Define system of equations
problem Elasticity([u1,u2], [v1,v2]) =
    int2d(Omega) (  sigma(u1,u2)'*e(v1,v2)  )
    + on(Left1,u1=0)              // Dirichlet boundary conditions
    + on(Left2,u1=0)              // Dirichlet boundary conditions
    + on(Bottom,u2=0)
    + on(Right,u1=0)
    + on(Top,u2=0)
  // Boundary conditions
  // condition only on one component
  + int1d(Omega, Frac) (Pfrac*(N.x*v1))
  ;



//---------------------------------------------------------------------
//  Solve system
Elasticity;


// Stresses
sigmaxx = sigma(u1,u2)[0];
sigmayy = sigma(u1,u2)[3];
sigmaxy = sigma(u1,u2)[1];  // we could use [2] as  well


//---------------------------------------------------------------------
// plot on the deformed surface
mesh Th=movemesh(Omega,[x+10*u1,y+10*u2]);
plot(Th,cmm="Deformed configuration",wait=1);


// plot the deformation field and stress
plot([u1,u2],coef=10,cmm="Displacement field",wait=1,value=true);
plot(sigmaxx,fill=1, cmm="Stress sigmaxx",wait=1,value=true);
plot(sigmayy,fill=1, cmm="Stress sigmayy",wait=1,value=true);
plot(sigmaxy,fill=1, cmm="Stress sigmaxy",wait=1,value=true);


// Write stress field
ofstream ff("FractureShadow.dat");
for(int i=0;i<100;i++) {
for(int j=0;j<100;j++) {
    // x, y, Sxx, Syy, Sxy
        real xline = xa + xSize*i/100.;
        real yline = ya + ySize*j/100.;
    // Analytical solution
    //write file numerical and analytical solution
    ff<< xline <<", "<< yline
        <<", "<< sigmaxx(xline,yline)
        <<", "<< sigmayy(xline,yline)
        <<", "<< sigmaxy(xline,yline)
        <<endl;
}
}

// Write horizontal line from the fracture center
ofstream fc("FractureCenter.dat");
fc << "x,y,sigxx" << endl;
for(int i=0;i<1000;i++) {
  real xline = xa + xSize*i/1000.;
  real yline = 0;

  fc<< xline <<", "<< yline
    <<", "<< sigmaxx(xline,yline)
    <<endl;
}


// Write displacement along the fracture wall
```

```
ofstream ffw("FractureWall.dat");
ffw << "x,y,u1,u2" << endl;
for(int i=0;i<1000;i++) {
  real t = -pi/2 + pi * i / 1000. ;
  real x = fw * cos(t) ;
  real y = xf * sin(-t) ;

  ffw<< x <<","<< y <<","<< u1(x,y) << "," << u2(x,y) << endl;
}

// Along X=0
ofstream ffx0("Along_X_equals_0.dat");
ffx0 << "x,y,u1,u2,sxx,syy,sxy" << endl;
for(int i=0;i<1000;i++) {
  real x = 0;
  real y = ya + ySize/1000*i;

  ffx0 << x << "," << y
       << "," << u1(x,y)
       << "," << u2(x,y)
       << "," << sigmaxx(x,y)
       << "," << sigmayy(x,y)
       << "," << sigmaxy(x,y)
       << endl;
}
```

]:

```

```