HW9

November 29, 2023

Exercise 1

[3]:
```
E_psi = 7.3e6
E = 6894.76 * E_psi

c_ft = 150
c = 150 * 0.3048

nu = 0.2

w = 4e-3
P_net = E / ( 1 - nu**2 ) / 4 / c * w

SH_psi = 6400
SH = SH_psi * 6894.76
Sh_psi = 6300
Sh = Sh_psi * 6894.76

print(f"Pnet={P_net / 1e6} MPa")
print(f"E={E / 1e9} GPa")
print(f"c={c}m")
print(f"SH={SH/1e6} MPa")
print(f"Sh={Sh/1e6} MPa")

# Net pressure for each farcture width
w = 4e-3
P_net = E / ( 1 - nu**2 ) / 4 / c * w
print(f"w={w} => P_net={P_net/1e6:.4f}")

w = 6e-3
P_net = E / ( 1 - nu**2 ) / 4 / c * w
print(f"w={w} => P_net={P_net/1e6:.4f}")

w = 8e-3
P_net = E / ( 1 - nu**2 ) / 4 / c * w
print(f"w={w} => P_net={P_net/1e6:.4f}")

w = 4e-3
P_net = E / ( 1 - nu**2 ) / 4 / c * w
print(f"w={w} => P_net={P_net/1e6:.4f}")
```

```
Pnet=1.1467389362787985 MPa
E=50.331748 GPa
c=45.72m
SH=44.126464 MPa
Sh=43.436988 MPa
w=0.004 => P_net=1.1467
w=0.006 => P_net=1.7201
w=0.008 => P_net=2.2935
w=0.004 => P_net=1.1467
```

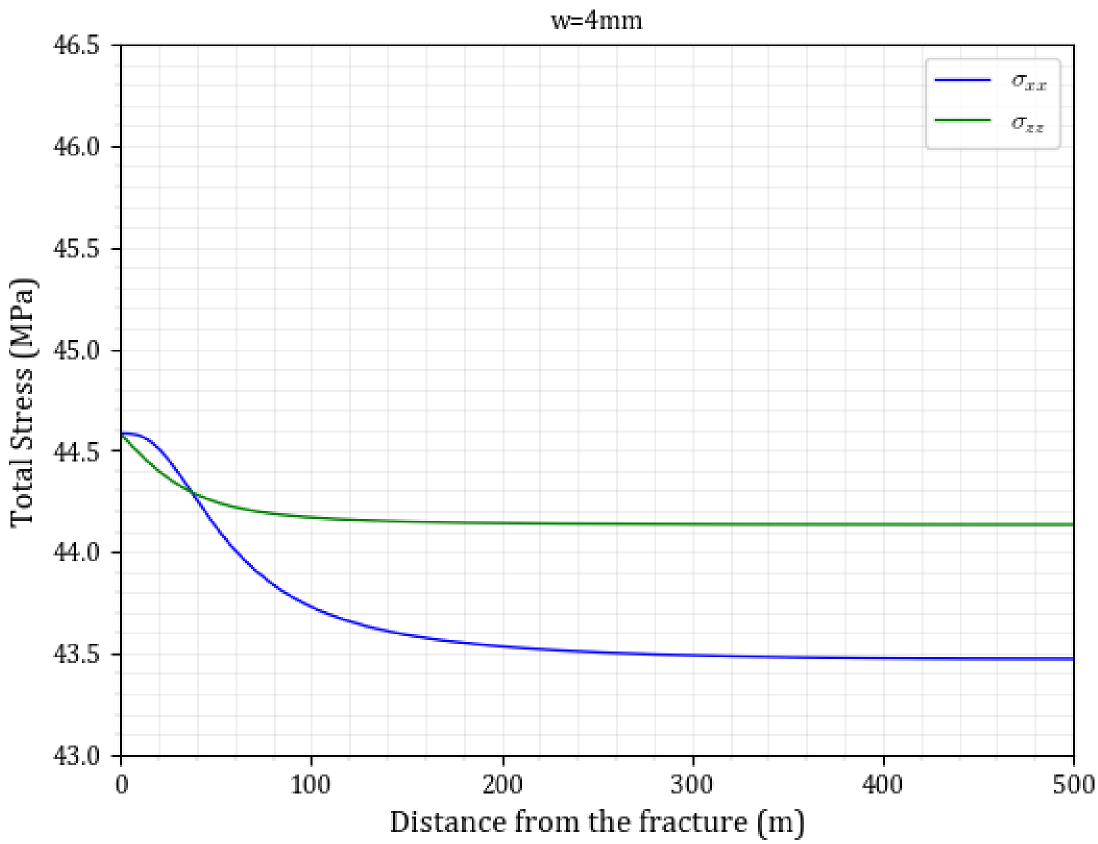
[3]:
```
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('default')    ## reset!
plt.style.use('paper.mplstyle')

#
#
#
def find_xcross (df, title) :
    df['sxx_tot'] = Sh - df.sigxx
    df['sigzz'] = nu * ( df.sigxx + df.sigyy )
    df['szz_tot'] = SH - df.sigzz

    # d_syy = nu * ( d_sxx + d_szz )
```

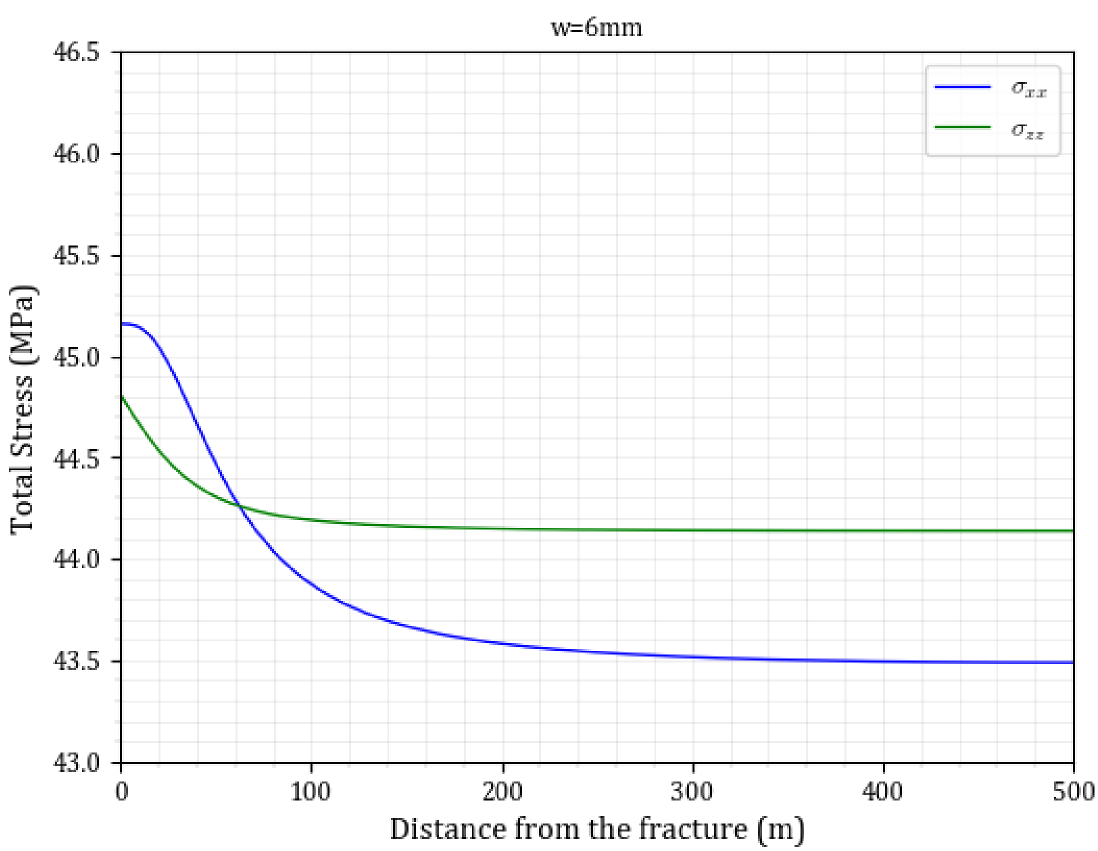
```python
    from scipy.interpolate import make_interp_spline, PPoly
    import matplotlib.pyplot as plt
    spline = make_interp_spline(df.x, df.sxx_tot - df.szz_tot, k=1) # Create a cubic spline from the data
    curve  = PPoly.from_spline(spline)     # Create a piecewise polynomial object; in essence, y = curve(x)
    width = curve.solve(y=0)
    width = width[ width > 20 ]
    df["spline"] = spline( df.x )

    fig, ax = plt.subplots()
    ax.plot( df.x, df.sxx_tot/1e6, label='$\sigma_{xx}$' )
    ax.plot( df.x, df.szz_tot/1e6, label='$\sigma_{zz}$'  )
    #ax.plot( df.x, df.szz_tot, label='$\sigma_{zz}$'  )
    ax.set_title( title )
    ax.set_xlabel("Distance from the fracture (m)")
    ax.set_ylabel("Total Stress (MPa)")
    ax.set_xlim(0,500)
    ax.set_ylim(43,46.5)
    ax.legend()

    #fig, ax = plt.subplots()
    #ax.plot( df.x, df.sxy_tot, label='sxy' )
    #ax.plot( df.x, df.szz_tot, label='szz' )
    #ax.legend()
    #fig, ax = plt.subplots()
    #ax.scatter( df.x, df.sxx_tot - df.syy_tot, label='Sxx - Syy', marker='x', alpha=0.3 )
    #ax.plot( df.x, df.spline, label='Spline', c='r' )

    print( width )
    return width[0]

w = [ 4, 6, 8, 10 ]
xc = []

df = pd.read_csv( "FractureCenter_w4.dat", sep=",")
xc.append ( find_xcross( df, "w=4mm" ) )

df = pd.read_csv( "FractureCenter_w6.dat", sep=",")
xc.append ( find_xcross( df, "w=6mm" ) )

df = pd.read_csv( "FractureCenter_w8.dat", sep=",")
xc.append ( find_xcross( df, "w=8mm" ) )

df = pd.read_csv( "FractureCenter_w10.dat", sep=",")
xc.append ( find_xcross( df, "w=10mm" ) )

fig, ax = plt.subplots()
ax.scatter( w, xc, marker='x' )
ax.set_xlabel("Fracture width $w_0$ (mm)")
ax.set_ylabel("Distance of the isotropic point (m)")
```

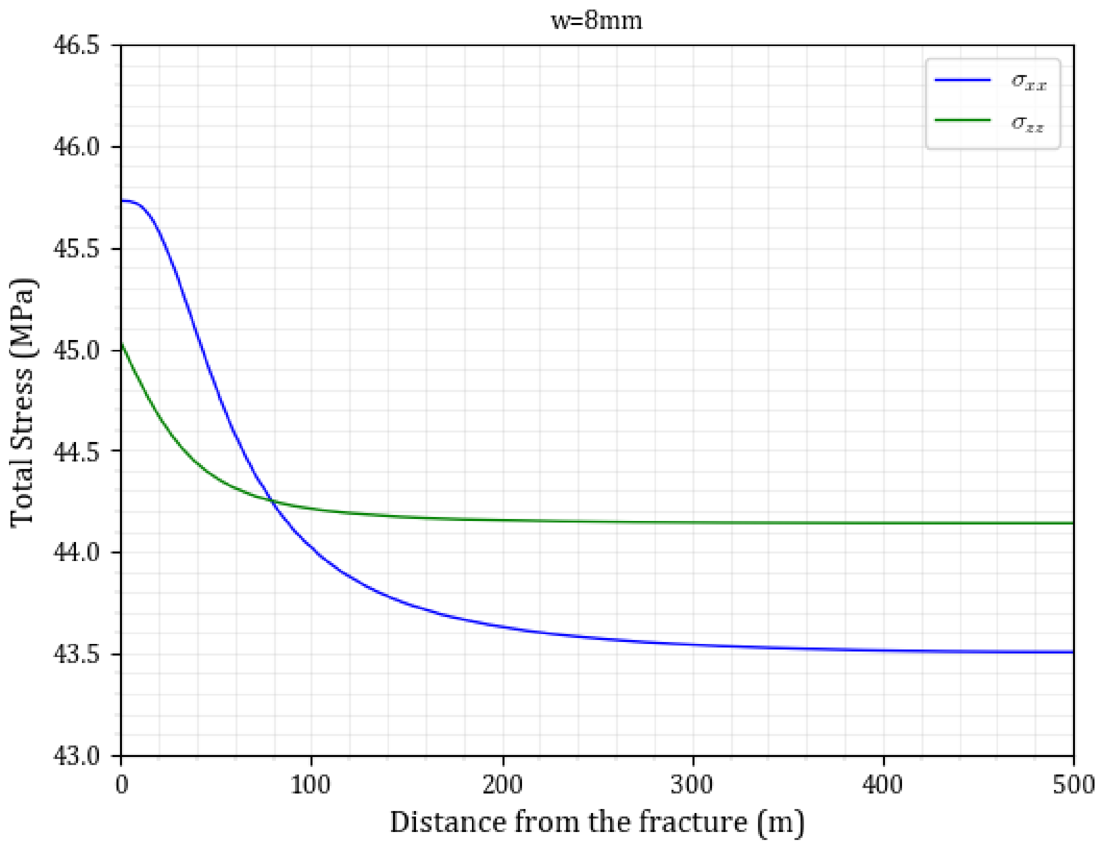
```
[37.28153309]
[62.35566458]
[79.30553238]
[93.25465621]
```

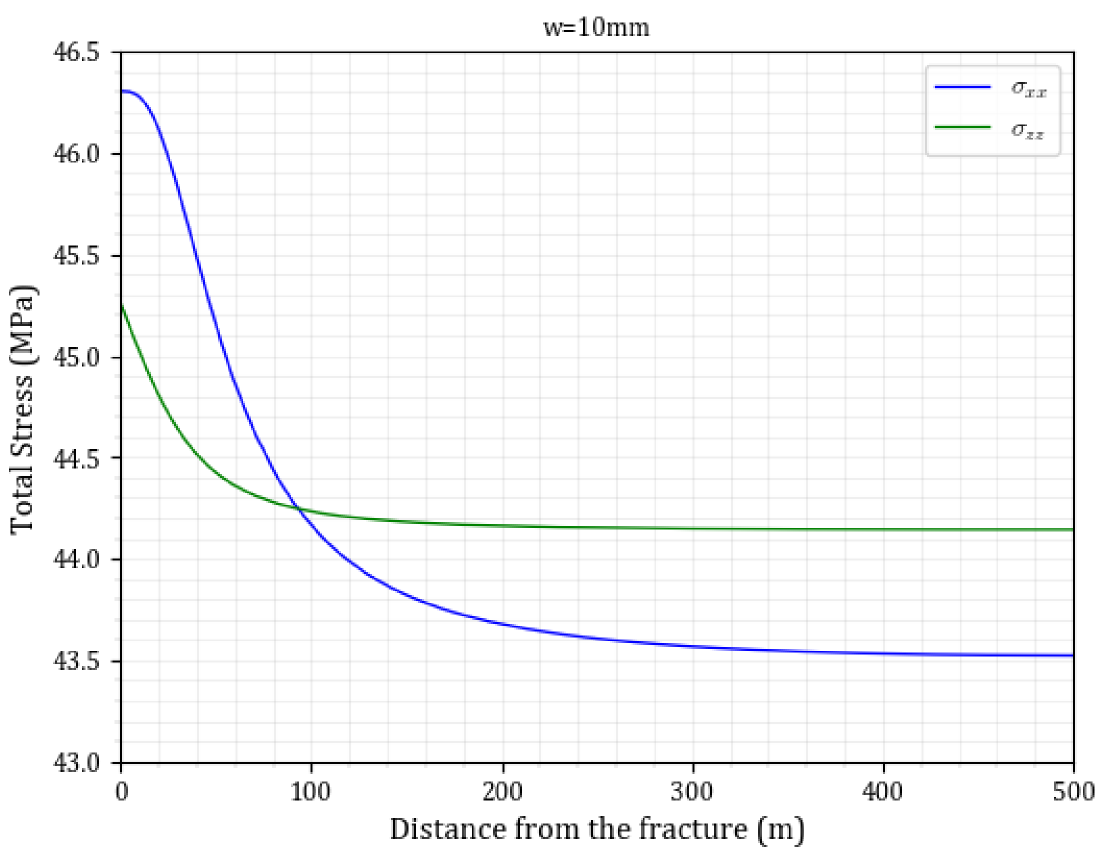
3]:

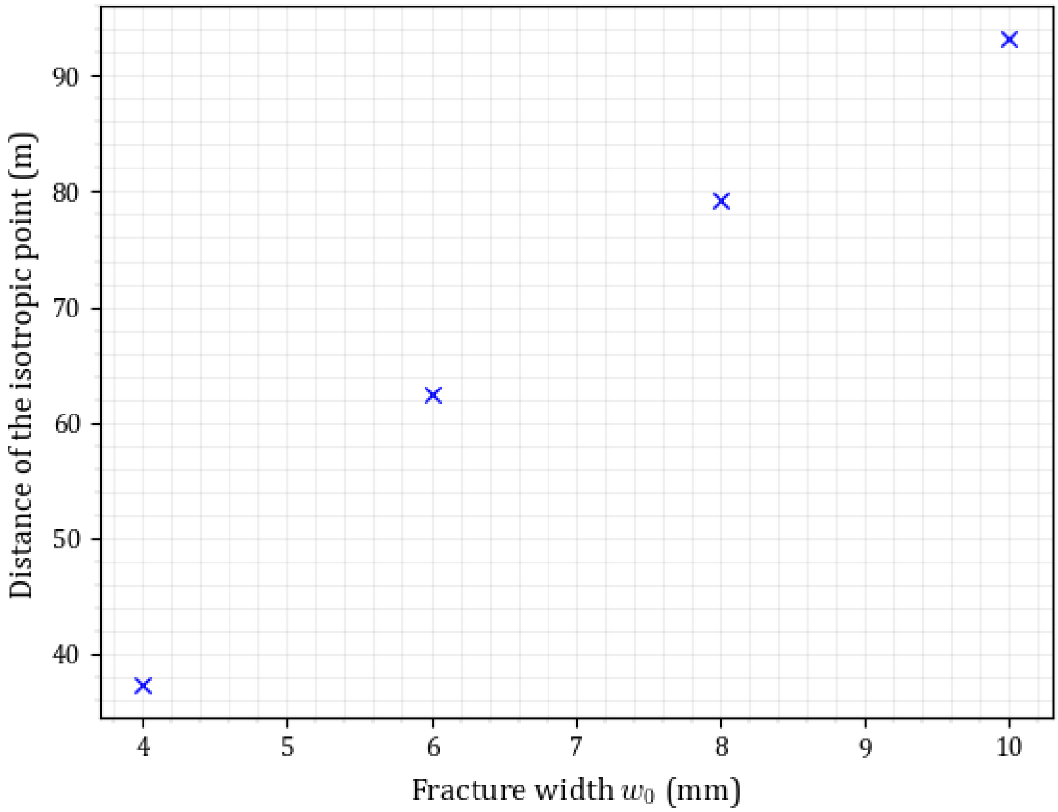
```
Text(0, 0.5, 'Distance of the isotropic point (m)')
```


w=4mm
46.5
46.0
45.5
45.0
44.5
44.0
43.5
43.0
Total Stress (MPa)
0
100
200
300
400
500
Distance from the fracture (m)
σ_{xx}
σ_{zz}


w=6mm
46.5
46.0
45.5
45.0
44.5
44.0
43.5
43.0
Total Stress (MPa)
0
100
200
300
400
500
Distance from the fracture (m)
σ_{xx}
σ_{zz}


w=8mm
46.5
46.0
45.5
45.0
44.5
44.0
43.5
43.0
Total Stress (MPa)
0
100
200
300
400
500
Distance from the fracture (m)
σ_{xx}
σ_{zz}


w=10mm
46.5
46.0
45.5
45.0
44.5
44.0
43.5
43.0
Total Stress (MPa)
0
100
200
300
400
500
Distance from the fracture (m)
σ_{xx}
σ_{zz}



Exercise 2

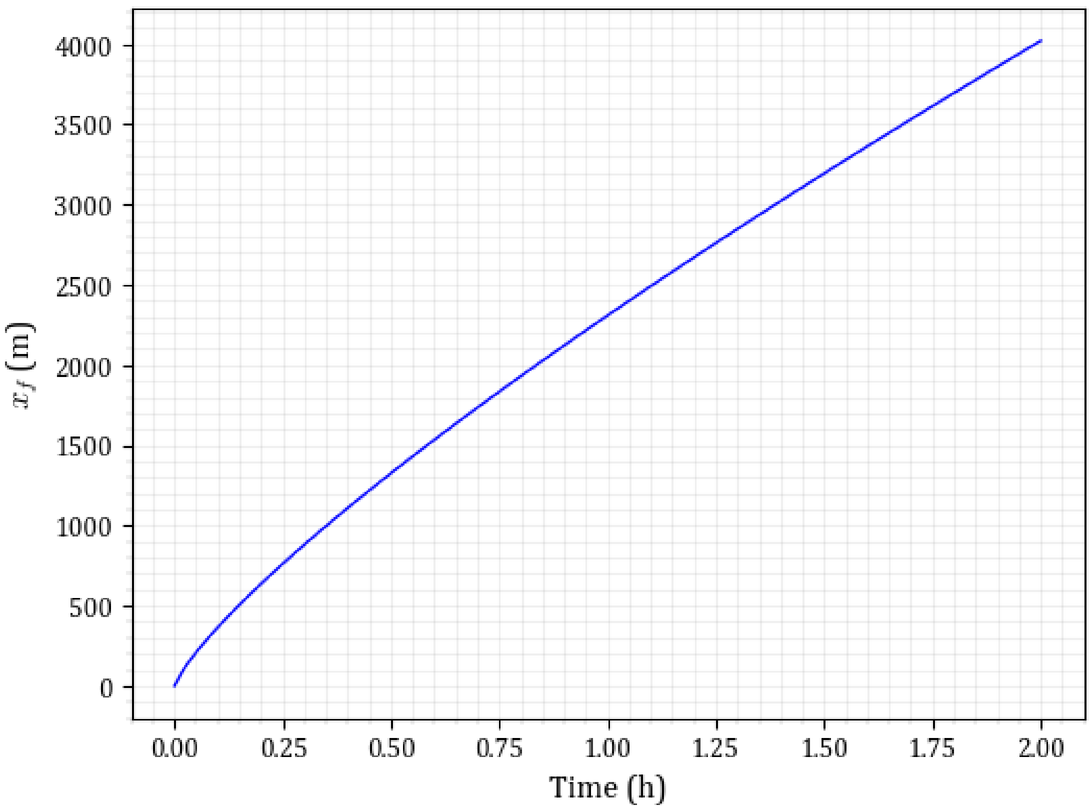
```
[6]: import numpy as np

hf_ft = 170
hf = hf_ft * 0.3048   # m

E__psi = 8.9e6
E_ = E__psi * 6894.76    # Pa

Q_bbl_min = 50   # bbl/min
Q = 50 / 377.3884 # m3/s

mu_cp = 2    #cp
mu = 2 * 1e-3 # Pa.S

def PKN( t ) :
    global Q, E_, mu, hf
    xf = 0.524 * ( Q**3 * E_ / mu / hf**4 ) ** 0.2 * t**(4/5)
    w0 = 3.04 * ( Q**2 * mu / E_ / hf )**0.2*t**0.2
    Pnet = 1.52 * ( E_**4 * Q**2 * mu / hf**6 ) ** 0.2 * t**0.2

    return xf, w0, Pnet
```

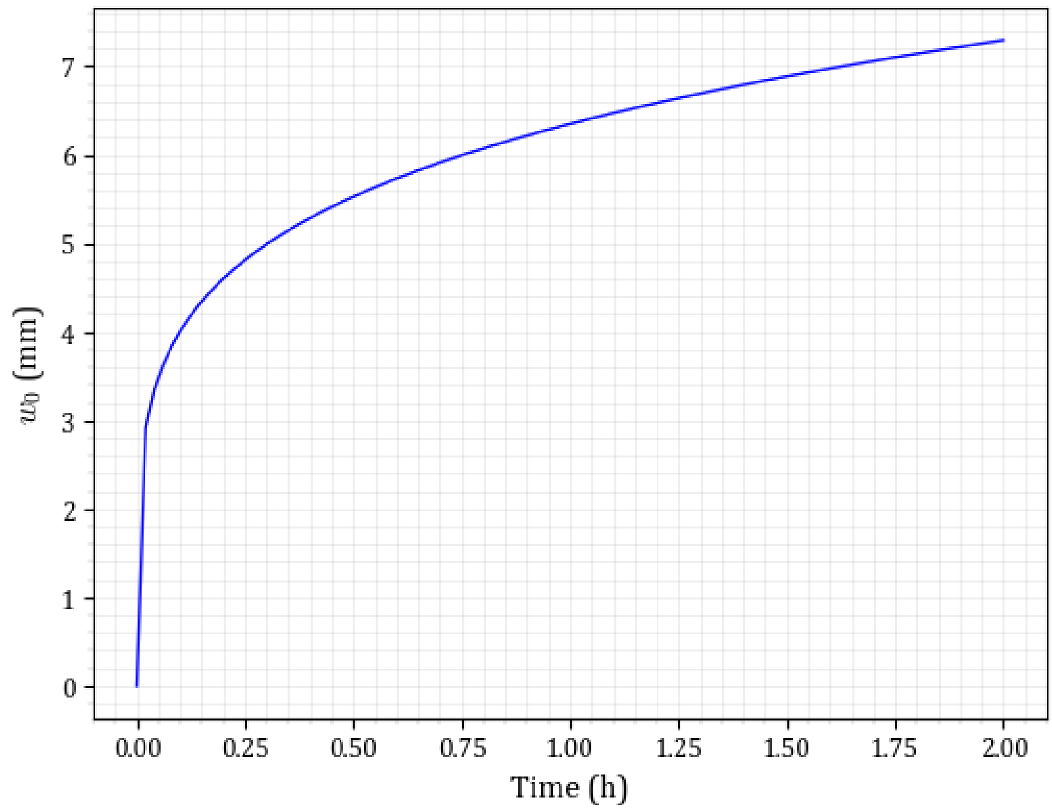
```
[7]: t_hour = 1
t = 1 * 60 * 60 # sec

xf, w0, Pnet = PKN(t)

print( f"xf={xf:.2f} m" )
print( f"w0={w0*1e3:.2f} mm" )
print( f"Pnet={Pnet/1e6:.2f} MPa" )
```

```
xf=2309.33 m
w0=6.35 mm
Pnet=3.76 MPa
```

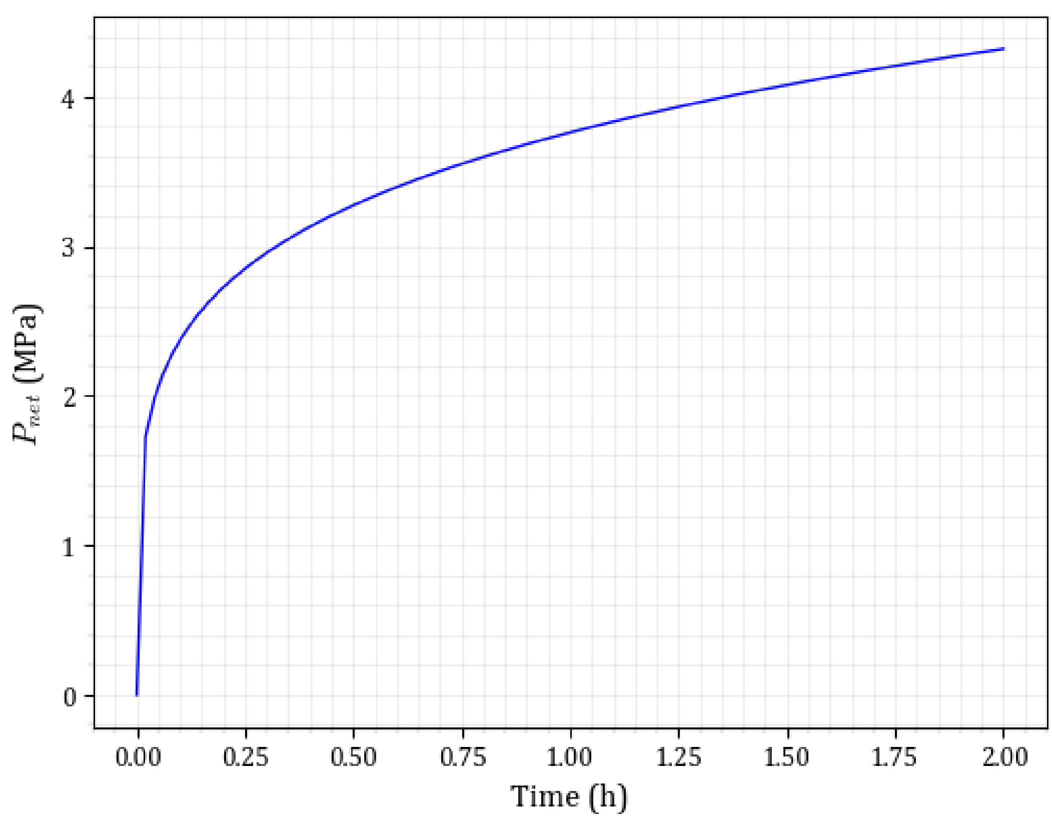
2]:
```
T = np.linspace( 0, 2*60*60, 100 ) #seconds
T_h = T/60/60

xf, w0, Pnet = PKN(T)

import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('default')   ## reset!
plt.style.use('paper.mplstyle')

fig, ax = plt.subplots()
ax.plot(T_h,xf)
ax.set_ylabel("$x_f$ (m)")
ax.set_xlabel("Time (h)")

fig, ax = plt.subplots()
ax.plot(T_h,w0*1e3)
ax.set_ylabel("$w_0$ (mm)")
ax.set_xlabel("Time (h)")

fig, ax = plt.subplots()
ax.plot(T_h,Pnet/1e6)
ax.set_ylabel("$P_{net}$ (MPa)")
ax.set_xlabel("Time (h)")
```

2]:
```
Text(0.5, 0, 'Time (h)')
```





w_0 (mm)
Time (h)
0
1
2
3
4
5
6
7
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00



P_{net} (MPa)
Time (h)
0
1
2
3
4
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

3. What should you do to your solution in order to consider leak-off? Justify and explain briefly the algorithm to calculate x_f, $w_{w,0}$, and p_n. The leakoff would reduce the effective rate of fluid inside the fracture (namely, the i parameter in the equation), and the energy aviailable to open the fracture.

The algorithm would probably be iterative, as the leakoff model would depend tightly on the x_f, P_{net} and w_0 parameters, and vice-versa. So I believe we would need to calculate the solution by stepping, to deal with such tight coupling.

[3]:
```python
with open('Fracture.edp', 'r') as file:
    data = file.read().rstrip()

print(">> --------------------------------------------------")
print(">> FreeFem++ input - Fracture.edp")
print(">> --------------------------------------------------\n\n")
print(data)
```

```
>> --------------------------------------------------
>> FreeFem++ input - Fracture.edp
>> --------------------------------------------------


load "iovtk"
//----------------------------------------------------------------------
// Dimensions
real xa = 0. ;
real xb = 500. ;  // x-size of the domain

real ya = -300. ;
real yb = 300. ;  // y-size of the domain

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

// Elastic constants
real E = 50.33e9 ;     // 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 Pfrac = 2.8668e6;

// w=0.004 => P_net=1.1467e6
// w=0.006 => P_net=1.7201e6
// w=0.008 => P_net=2.2935e6
// w=0.01 => P_net=2.8668e6

// FRACTURE
real xf = 45.72; // fracture half-length
real fw = .000; // 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_w10.dat");
fc << "x,y,sigxx,sigyy,sigxy" << endl;
for(int i=0;i<1000;i++) {
  real xline = xa + xSize*i/1000.;
  real yline = 0;

  fc<< xline <<", "<< yline
    <<", "<< sigmaxx(xline,yline)
```

```
    <<", "<< sigmayy(xline,yline)
    <<", "<< sigmaxy(xline,yline)
    <<endl;
}
```