PROBLEM 3

Contact Solution:

The contact problem is approximated as a plane strain problem between a cylinder and an infinite half-plane. Hertzian contact is assumed allowing the normal load to be distributed parabolically over a finite span of half-width \( a \). The elastic solution to this contact problem (Fig. 1) is given in Merwin-Johnson’s paper [1] in elliptical coordinates. A more complete description of the solution is available in the work of Radzimovsky [2].

![Diagram of rolling direction](image)

**Figure 1:** Line Hertzian contact (from Jiang-Sehitoglu [3])

To compute the stress/strain path histories according to the elastic solution, python scripts were written, which essentially compute the elastic solution. The stress path is plotted in Figure 2 and the strain path is plotted in Figure 3. We follow the properties chosen by Jiang-Sehitoglu.

\[
G = 79.6 \text{GPa}
\]

\[
\nu = 0.3
\]

For the plastic behavior, we assume a linear purely kinematic hardening behavior, with a hardening modulus:

\[
h = 25.7G
\]
Figure 2: Variation of elastic stresses at a constant depth ($y = -a$)

Figure 3: Variation of elastic strains at a constant depth ($y = -a$)
APPROACH

To model the rolling fatigue problem elasto-plastically, we first make an assumption of the controlling parameter. We can either choose a strain invariant approach where each material element is subjected to a strain-controlled path as per figure 3 or we can choose a stress invariant approach where each material element is subjected to a stress-controlled approach as per figure 2.

STRESS-INvariant APPROACH (Jiang-Sehitoglu)

The stress-invariant approach is motivated by the fact that the contact stresses are not significantly affected by plastic strain accumulation. Hence, the stress-cycle for each element is equal to the elastic solution.

1. **Discretize the x-path:** Treat the stress state at each spatial x point as a point on the stress history for a material element. This means we know the complete stress history, and the goal is to keep track of stress residuals.

2. **First pass:** Away from the contact region (|x| > 10a), the stresses are elastic and we use that as the starting point for the incremental plastic solution. So the current stress state follows the elastic solution - (σ_{xx}^l, σ_{yy}^l, σ_{zz}^l, τ_{xz}^l).

3. **Stress invariant assumption:** The increments are dictated by the assumption:

   \[ \Delta \sigma_{mn}^l = \sigma_{mn}^{l+1} - \sigma_{mn}^l \]

4. **Elastic or Plastic increment:** Use the current stress-state and check if we are at the yield surface. If the stress-state is elastic, the strain increments in an elastic manner and the out-of-plane normal stress is dictated by the plane-strain condition:

   \[ \Delta \sigma_{yy}^l = \nu (\Delta \sigma_{xx}^l + \Delta \sigma_{zz}^l) \]

   For a plastic increment, the increments are computed as:

   \[ \Delta \sigma_{yy}^l = \left[ (\nu - \frac{E}{h} n_x n_y) \Delta \sigma_{xx}^l + (\nu - \frac{E}{h} n_y n_z) \Delta \sigma_{zz}^l - \frac{2E}{h} n_x n_y \Delta \tau_{xz}^l \right] \frac{1}{1 + \frac{E}{h} n_y^2} \]

   The plastic strain accumulation can be tracked by the Prandtl-Reuss equations and the Prager rule gives the increment of the back-stress (please refer [3]).

5. **Relaxation procedure:**

   Recall that we have followed a stress-controlled approach. So, after completion of the first pass, the stresses (σ_{xx}, σ_{zz}, τ_{xz}) will have almost returned to zero (depends on how big a path you choose along the rolling direction). However, there is some plastic strain that has been accumulated. The primary effect it has in the solution is via ε_{xx}.

   From a physics point of view, we expect no accumulation of normal strain along the rolling direction (else the “large cylinder”, which is approximated as the plane here, will eventually take a “wiggled/corrugated shape”. This experimental observation has been discussed, please see [1]. When experimental control was chosen to avoid that behavior, we still see surface displacement and that is the situation we seek to model). Hence, we want ε_{xx}^l = 0.

   To bring the circumferential strain back to zero we follow a relaxation procedure. This is done in small increments because it is possible for the material to yield during this process (early yield upon unload – Bauschinger effect as captured by kinematic hardening).
Further, we also ensure that global equilibrium is satisfied. This means \( \sigma_{zz}^r = \tau_{xx}^r = 0 \). While this is satisfied for the first pass, it does not hold over multiple passes and must be explicitly enforced.

The uncorrected residuals are decremented stepwise, over M steps:

\[
\Delta \varepsilon_{xx} = -\frac{\varepsilon_{xx}^b}{M}; \Delta \sigma_{zz} = -\frac{\sigma_{zz}^b}{M}; \Delta \tau_{xz} = -\frac{\tau_{xz}^b}{M}
\]

And these decrements are also done elasto-plastically.

6. **Accumulated residuals:** After relaxation, there are only two residual stresses that are allowed i.e. \( \sigma_{xx}^r, \sigma_{yy}^r \).

7. **Next pass:** For the next iteration, we add these residual stresses to the elastic solution and repeat the process.

It is worthwhile to note the following:

a. **Stress partitioning:** Throughout the process, there are two parts in the stress state. One adheres to the spatially varying elastic solution, and the other part is the residual stress (only two components). The elastic solution follows equilibrium (derived that way) and the residual stresses were explicitly modified to follow equilibrium.

\[
\sigma_{mn} = \sigma_{mn}^e (\text{contact solution}) + \sigma_{mn}^p (\text{residual})
\]

b. **Strain partitioning:** The strain can be partitioned into three parts. One is the elastic strain associated with the elastic contact solution, the second is the plastic strain which we track along the process, and the third is the elastic strain associated with the residual stress.

\[
\varepsilon_{mn} = \varepsilon_{mn}^e (\text{contact solution}) + \varepsilon_{mn}^p (\text{elastic residual}) + \varepsilon_{mn}^p (\text{plastic})
\]

The residual stresses and the elastic strains caused by the elastic residual stresses are **not plastic** and must not be confused with those terms.

Two python codes were written (attached):

i. GetStressPaths.py – To compute the stress paths
ii. JiangSehitoglu.py - To get the data for accumulated residuals at each depth \( z/a \).

**STRAIN-IN Variant APPROACH (Merwin-Johnson):**

The strain-invariant approach is motivated by the fact that around the contact zone, the material must meet some boundary displacement conditions. Hence, it is assumed that the strain cycle is equal to the elastic solution. After discretizing the x-path to get a strain-history, we proceed as follows:

1. **First pass:** Away from the contact region (\(|x| > 10a\)), the stresses are elastic and we use that as the starting point for the incremental plastic solution. So the current strain state follows the elastic solution - \( (\varepsilon_{xx}^l, \varepsilon_{zz}^l, \varepsilon_{yy}^l, \gamma_{xz}^l) \).
2. **Stress invariant assumption:** The increments are dictated by the assumption:

\[
\Delta \varepsilon_{mn}^l = \varepsilon_{mn}^{l+1} - \varepsilon_{mn}^l
\]

3. **Elastic or Plastic increment:** Use the current stress-state and check if we are at the yield surface. If the stress-state is elastic, the stress increments in an elastic manner. Hence, Hooke’s law gives the stress-increments.

For a plastic increment, the stress increments need to be solved for. The plane-strain condition is the same.
\[
\Delta \sigma_{yy}^i = \left[ \left( \frac{v - \frac{E}{h} n_x n_y}{1 + \frac{E}{h} n_y} \right) \Delta \sigma_{xx}^i + \left( \frac{v - \frac{E}{h} n_x n_y}{1 + \frac{E}{h} n_y} \right) \Delta \sigma_{zz}^i - \frac{2E}{h} n_x n_y \Delta \tau_{xz}^i \right] \]

The strain-increment equations are the additional:

\[ \Delta \epsilon_{ij} = \frac{1}{h} < \Delta S_{kl} n_{kl} > n_{ij} + S_{ijkl} \Delta \sigma_{kl} \]

where, Hooke’s law for isotropic linear elastic materials gives the S tensor (compliance tensor).

Together, we have 4 equations for 4 stress increments (\( \Delta \sigma_{xx}, \Delta \sigma_{zz}, \Delta \sigma_{yy}, \Delta \tau_{xz} \)). The stress increments are solved for.

4. Relaxation procedure:

Having followed a strain-controlled approach, So after completion of the first pass, the strains \((\sigma_{xx}, \sigma_{zz}, \tau_{xz})\) will have almost returned to zero (depends on how big a path you choose along the rolling direction). In this approach, while plastic strain has been accumulated, \(\epsilon_{xx}^b \approx 0\) after the first pass. However, we have the stress-residuals which have to be relaxed to zero. Over subsequent passes, all 3 will be non-zero and should be relaxed.

The uncorrected residuals are decremented stepwise, over M steps:

\[ \Delta \epsilon_{xx} = -\frac{\epsilon_{xx}^b}{M} ; \Delta \sigma_{zz} = -\frac{\sigma_{zz}^b}{M} ; \Delta \tau_{xz} = -\frac{\tau_{xz}^b}{M} \]

5. Accumulated residuals: After relaxation, there are only two residual stresses that are allowed i.e. \(\sigma_{xx}, \sigma_{yy}\).

6. Next pass: This is the second interesting distinction from the stress-invariant approach. As mentioned in the previous step, we have two accumulated residual stress components to carry over. But recall that we are working with a strain-control approach. So, in addition to carrying over the stress residual for the next cycle (affects the stress-state), we need to compute the elastic strain residuals \(\epsilon_{yy}, \epsilon_{zz}\) from these increments and add it to the next iteration:

\[ \epsilon_{yy}^r = \frac{1}{E} (\sigma_{yy}^r - v \sigma_{xx}^r) \]
\[ \epsilon_{zz}^r = \frac{1}{E} (\sigma_{zz}^r + \sigma_{yy}^r) \]

(Remember \(\epsilon_{xx}^r = \sigma_{xx}^r = \tau_{xz}^r = 0\))

Two python scripts were written:

i. GetStrainPaths.py – To compute the strain paths
ii. MerwinJohnson.py – To get the data for accumulated stress residuals at each depth z/a

RESULTS:

The results of both approaches are shown in figures 4 and 5.
Figure 4: Plot of residual tangential normal stress ($\sigma_x^r/k$)

- Dashed line: Jiang-Sehitoglu (1994)
- Dotted line: Merwin-Johnson (1963)
Figure 5: Plot of residual out-of-plane normal stress ($\sigma_{y}/k$)

Figures 4 and 5 match figures 3(a) and 3(b) of [3], respectively.

DIFFERENCE BETWEEN THE TWO SOLUTIONS:

While the procedure highlights few crucial differences between the two approaches, there is one argument which shows it best. Look back at the stress and strain partitioning for the strain-invariant approach. The strains for the next step are updated with the residuals and plastic strains were ignored.

Strain partitioning (strain invariant approach): To get the results in the figures 4 and 5, we choose to update the strains in the following way, for the Merwin-Johnson solution.

$$\epsilon_{mn} \approx \epsilon_{mn}^{n}(contact\ solution) + \epsilon_{mn}^{r}(elastic\ residual)$$

Comparing to the stress-invariant approach, it is obvious that the plastic strain has been omitted. This approach assumes that plastic distortion does not significantly affect the boundary displacements, hence it is neglected.
However, in the same strain invariant approach, if we choose to include the plastic strain as well, then the stress-invariant solution is recovered! The plots are reproduced in figure 6. The modification has also been included as a comment in the Merwin-Johnson python code.

\[ \varepsilon_{mn} = \varepsilon_{mn}^{c} (\text{contact solution}) + \varepsilon_{mn}^{r} (\text{elastic residual}) + \varepsilon_{mn}^{p} \]
REFERENCES:


import math
import matplotlib.pyplot as plt
import numpy as np

def StressPaths(nu,zval):
    #nu=0.3;
    #k=1.0/6.0; #p0/k=6.0

    #Plot is only along the axis of symmetry

    x=[float(0.00001)];
    xmax=30.0;
    diff=0.01;
    count=int(xmax/diff);

    for i in range(count):
        x.append(float(i+1)*diff);

    #z=[float(0.0001)];
    #zval=1.0001; #Depth at which the stress/strain path is computed
    #bet=[];
    #alp=[];
    #sinb=[];
    #sinhalp=[];

    #sigxpath=[];
    #sigypath=[];
    #tauxzpath=[];
    #sigzpath=[];

    sigx=[];
    sigy=[];
    tauxz=[];
    sigz=[];
    tauxmax=[];
    epsx=[];
    epsy=[];
    gamxz=[];
    epsz=[];

    #for zval in z:
    for xval in x:
        b=float(1.0-(xval**2)-(zval**2));
        c=math.sqrt(b**2+4*(zval**2));
        #sinb.append(math.sqrt(0.5*(b+c)));
        #sinhalp.append(math.sqrt(0.5*(-b+c)));

        betaval=math.asin(math.sqrt(0.5*(b+c)));
        #should be between and pi/2
        alphaval=math.asinh(math.sqrt(0.5*(-b+c)));
        #bet.append(betaval);
        #alp.append(alphaval);
term = math.sin(betaval) * math.sinh(alphaval) * (1 - math.sinh(2 * alphaval) / math.cosh(2 * alphaval)) * math.cos(2 * betaval)))

sigzval = -math.exp(-alphaval) * math.sin(betaval) * term;
sigmaval = -math.exp(-alphaval) * math.sin(betaval) * term;

tauzval = math.sin(alphaval) * math.sin(betaval) * (1 - math.sinh(2 * alphaval) / math.cosh(2 * alphaval)) * math.cos(2 * betaval)));
tauzval = 0.5 * (sigzval - sigzval);
sigyval = mu * (sigzval + sigzval);

sigz.append(sigzval);
sigx.append(sigxval);
tau.append(tauzval);
sigy.append(sigyval);
    #taumax.append(tauzval);
    #epsx =
    #sigxpath.append(sigx);
    #sigyval = sigy;
    #tauxzpath.append(tauz);
    #sigzpath.append(sigz);

    # Now we have a half-path - Flip and append to get entire path
xneg = np.asarray(x);
xneg = xneg;
xneg = xneg.tolist();
xneg = xneg.reverse();
x = xneg + x;

# Use symmetry of elastic solution to get negative quadrant path
sigz = sigz[::-1] + sigz;
sigx = sigx[::-1] + sigx;
sigy = sigy[::-1] + sigy;

# tau is antisymmetric
tau = np.asarray(tauz[::-1]);
tau = -tau;
tau = tau.tolist();
tauz = tau + tauz;

# plot
plot.plot(x, sigz, label='$\sigma_{zz}$ / p_0$');
# plot.plot(x, sigx, linestyle='--', label='$\sigma_{xx}$ / p_0$');
# plot.plot(x, tauz, linestyle=':', label='$\tau_{zz}$ / p_0$');
# plot.xlabel('$x / a$');
# plot.ylim(4, 0); # decreasing
# plot.xlim(0.0, -1.0); # decreasing
# plot.legend()
# plot.show();

sigzpath = [x, sigx, sigz, sigy, tauz];

return sigzpath;
import math
import matplotlib.pyplot as plotit
import numpy as np
from GetStressPaths import StressPaths

nu=0.3;
p0=6.0*606.0;
k=1.0/6.0;
G=79600.0; #Normalized against this
G=G/p0;
E=G*2*(1+nu);
h=25.7*G;
M=200; #Number of relaxation steps

z=[float(0.0001)];
zmax=3.0;
diff=0.05;
count=int(zmax/diff);
##
for i in range(count):
    z.append(float(i+1)*diff);

#Compute the elasto-plastic response for the first pass
numpaths=15;
p=0;
flag=0;
sigyrarr=[];
sigxrarr=[];
alpxarr=[];
#Kinematic hardening center
JStfile1=open('JSsigydata.csv','w');
JStfile2=open('JSsigxdata.csv','w');
for zval in z:
    sigpaths=StressPaths(nu,zval);
    #First list is x, then sigx, sigz, sigy and tauxz lists
    x=sigpaths[0];
sigx=sigpaths[1]; # a list
    sigz=sigpaths[2];
sigy=sigpaths[3];
tauxz=sigpaths[4];
    #print sigx[0]
sigxr=0.0;
sigy=0.0;
sigz=0.0;
tauxz=0.0;
alpx=0.0;
alpy=0.0;
alpz=0.0;
alpxz=0.0;
espx=0.0;
epsyp=0.0;
epszp=0.0;
epsxzp=0.0;

p=0;
while p < numpaths:
    p=p+1;
    #Compute residual stresses at each depth
    #sigy[0]=sigr
    for i in range(len(x)-1):
        delsigr=sigx[i+1]-sigx[i];
        delsigz=sigz[i+1]-sigz[i];
        delttauxz=tauxz[i+1]-tauxz[i];

    #Isolate the stress field solution (don't want to alter the listing itself)
    #Add the residual stresses
    if i == 0:
        sigyval=sigy[i]+sigr;
        sigxval=sigx[i]+sigx;
        sigzval=sigz[i];
        tauxzval=tauxz[i];
        
    #Check if yielding
    Sx=(sigxval-(1.0/3.0)*(sigxval+sigyval+sigzval));
    Sy=(sigyval-(1.0/3.0)*(sigxval+sigyval+sigzval));
    Sz=(sigzval-(1.0/3.0)*(sigxval+sigyval+sigzval));
    Sxz=tauxzval;
    nx=(Sx-alpx)/(math.sqrt(2)*k);
    ny=(Sy-alpy)/(math.sqrt(2)*k);
    nz=(Sz-alpz)/(math.sqrt(2)*k);
    nxz=(Sxz-alpxz)/(math.sqrt(2)*k);
    f=(nx**2)+(ny**2)+(nz**2)+2*(nxz**2)-1;
    if f <= 1:
        alpxarr.append(alpx);
    if f > 0:
        flag=1;
        #Plastic yielding
        delsigr=(nu-(E/h)*nx*ny)*delsigr+(nu-(E/h)*ny*nz)*delsigrz-(2*E/h)*nx*ny*deltauxz;
        delsigr=delsigr/(1+(E/h)*(ny**2));

        delttx=(delsigr-(1.0/3.0)*(delsigr+delsigr+y+delsigrz);
        delttx=delsigr-(1.0/3.0)*(delsigr+delsigr+y+delsigrz);
        delttx=delsigrz-(1.0/3.0)*(delsigr+delsigr+y+delsigrz);
        delttx=delttauxz;

        term=delttx*nx+delttx*ny+delttx*nz+2*delttx*nxz;
        MacBrac=0.5*(term+math.fabs(term));
        delepsxp=(1.0/h)*MacBrac*nx;
delepszp=(1.0/h)*MacBrac*nx;
delepsyp=(1.0/h)*MacBrac*ny;
delepsxzp=(1.0/h)*MacBrac*nxz;

delalpx=MacBrac*nx;
delalpy=MacBrac*ny;
delalpz=MacBrac*nxz;
delalpxz=MacBrac*nxz;

else:
    flag=0;
delsigy=nu*(delsigx+delsigz);
delepsxp=0.0;
delepszp=0.0;
delepsyp=0.0;
delepsxzp=0.0;

delalpx=0.0;
delalpy=0.0;
delalpz=0.0;
delalpxz=0.0;

    #Update the plastic parameters
    alpx=alpx+delalpx;
alpy=alpy+delalpy;
alpz=alpz+delalpz;
alpxz=alpxz+delalpxz;

    epsxp=epsxp+delepsxp;
    epsyp=epsyp+delepsyp;
    epszp=epszp+delepszp;
    epsxzp=epsxzp+delepsxzp;

    sigyval=sigyval+delsigy;
sigxval=sigxval+delsigx;
sigzval=sigzval+delsigz;
tauxzval=tauxzval+deltauxz;

    #Relaxation procedure
    #Remember that sigx and sigy accumulate residuals
    sigxb=sigxval-sigx[i];
sigyb=sigyval-sigy[i];  #Should be different from the path solutions
    sigzb=sigzval-sigz[i];
tauxzb=tauxzval-tauxz[i];
    #epsx=epsxp+sigxval-nu*(sigzval+sigyval);
    epsxb=epsxp+(1.0/E)*(sigxb-nu*(sigyb+sigzb));
    epsyb=epsyp+(1.0/E)*(sigyb-nu*(sigzb+sigxb));

    #print epsyb
    delepsx=(-epsxb)/M;
    delsizg=(-sigzb)/M;
    deltauxz=(-tauxzb)/M;

    for j in range(M):
# Use total stress values for current stress state
Sx = (sigxval - 1.0/3.0) * (sigxval + sigyval + sigzval);
Sy = (sigyval - 1.0/3.0) * (sigxval + sigyval + sigzval);
Sz = (sigzval - 1.0/3.0) * (sigxval + sigyval + sigzval);
Sxz = tauxzval;

nx = (Sx - alpx) / (math.sqrt(2) * k);
ny = (Sy - alpy) / (math.sqrt(2) * k);
nz = (Sxz - alpxz) / (math.sqrt(2) * k);
f = nx**2 + ny**2 + nz**2 + 2 * nxz**2 - 1;
if f > 0:
    # Plastic relaxation
    # print "Plastic relaxation"
    rhs1 = E * delepsx + (nu - E * nx * nz / h) * delsigz - 2 * (E / h) * nx * nxz * deltauxz;
    rhs2 = (nu - E * ny * nz / h) * delsigz - 2 * E * ny * nxz * deltauxz / h;
    coeff1 = (1 + E * nx**2 / h);
    coeff2 = (E * nx * ny / h) - nu;
    coeff3 = (E * nx * ny / h) - nu;
    coeff4 = (1 + E * ny**2 / h);
    det = coeff1 * coeff4 - coeff3 * coeff2;
    # print det
    delsigx = (rhs1 * coeff4 - rhs2 * coeff2) / (coeff1 * coeff4 - coeff3 * coeff2);
    delsigy = (rhs1 * coeff3 - rhs2 * coeff1) / (coeff2 * coeff3 - coeff4 * coeff1);
    delSx = delsigx - (1.0/3.0) * (delsigx + delsigy + delsigz);
    delSy = delsigy - (1.0/3.0) * (delsigx + delsigy + delsigz);
    delSz = delsigz - (1.0/3.0) * (delsigx + delsigy + delsigz);
    delSxz = deltauxz;
    term = delSx * nx + delSy * ny + delSz * nz + 2 * delSxz * nxz;
    MacBrac = 0.5 * (term + math.fabs(term));
    delepsxp = (l / h) * MacBrac * nx;
    delepszp = (l / h) * MacBrac * nz;
    delepsyp = (l / h) * MacBrac * ny;
    delepsxzp = (l / h) * MacBrac * nxz;

else:
    # print "Elastic Relaxation"
    # print i
    rhs1 = E * delepsx + nu * delsigz;
rhs2=nu*delsigz;
coeff1=1;
coeff2=-nu;
coeff3=-nu;
coeff4=1;
# print (coeff1*coeff4-coeff3*coeff2)
delsigx=(rhs1*coeff4-rhs2*coeff2)/(coeff1*coeff4-coeff3*coeff2);
delsigy=nu*(delsigx+delsigz);
delepsxp=0.0;
delepszp=0.0;
delepsyp=0.0;
delepsxzp=0.0;

delalpx=0.0;
delalpy=0.0;
delalpz=0.0;
delalpxz=0.0;

# Update plastic parameters
alpx=alpx+delalpx;
alpy=alpy+delalpy;
alpz=alpz+delalpz;
alpxz=alpxz+delalpxz;

epsxp=epsxp+delepsxp;
epsyp=epsyp+delepsyp;
epszp=epszp+delepszp;
epsxzp=epsxzp+delepsxzp;

# Update stress state for next iteration of relaxation
sigyval=sigyval+delsigy;
sigxval=sigxval+delsigx;
sigzval=sigzval+delsigz;
tauxzval=tauxzval+deltau;
sigr=sigxval-sigx[i]; # Residual accumulates
sigy=sigyval-sigy[i]; # Residual accumulates
sigz=sigzval-sigz[i];
tauz=tauzval-tauz[i];
epsx=epsx+(1.0/E)*(sigr-0.0*(sigy+sigz)); # Should be brought to zero

sigyrarr.append(6*sigyr);
sigxarr.append(6*sigx);
JSfile1.write('%%
JSfile2.write('%%
JSfile1.close();
JSfile2.close();
#plotit.xlabel('{$\sigma_{y,r}/k$}');
plotit.ylabel('{$z/a$}');
plotit.legend();
#plotit.plot(sigxarr,z)
plotit.ylim(3,0);
##plotit.plot(x[0:(len(x)-1)],alpxarr);
plotit.show();
```python
def StrainPaths(nu, zval, G):
    E = 2 * G * (1 + nu);
    x = [float(0.00001)];
    xmax = 30.0;
    diff = 0.01;
    count = int(xmax / diff);
    for i in range(count):
        x.append(float(i) * diff);

    epsx = [];
    epsy = [];
    gamx = [];
    epsz = [];
    sigx = [];
    sigy = [];
    taux = [];
    sigz = [];
    for zval in z:
        for xval in x:
            b = float(1.0 - (xval ** 2) - (zval ** 2));
            c = math.sqrt(b ** 2 + 4 * (zval ** 2));
            sinb = math.sin(math.sqrt(0.5 * (b + c)));
            sinh = math.sinh(math.sqrt(0.5 * (-b + c)));
            betaval = math.asin(math.sqrt(0.5 * (b + c)));
            alphaval = math.asinh(math.sqrt(0.5 * (-b + c)));
            bet.append(betaval);
            alph.append(alphaval);
            term = math.sin(betaval) * math.sinh(alphaval) * (1 - (math.sinh(2 * alphaval) / (math.cosh(2 * alphaval) ** 2)));
            sigzval = math.exp(-alphaval) * math.sin(betaval) - term;
            sigxval = math.exp(-alphaval) * math.sin(betaval) + term;
            tauxzval = math.sin(alphaval) * math.sin(betaval) * (math.sin(2 * betaval) / (math.cos(2 * betaval) ** 2));
            tauxmaxval = 0.5 * (sigxval - sigzval);
            sigyval = nu * (sigxval + sigzval);
            epsxval = (1.0 / E) * (sigxval - nu * (sigzval + sigyval));
            epsyval = (1.0 / E) * (sigyval - nu * (sigzval + sigxval));
            epszval = (1.0 / E) * (sigzval - nu * (sigxval + sigyval));
            gamxval = tauxzval / G;
```

Scanned by CamScanner
epsx.append(epsxval);
epsz.append(epszval);
epsy.append(epsyval);
gamxz.append(gamxzval);

sigz.append(sigzval);
sigx.append(sigxval);
tauxz.append(tauxzval);
sigy.append(sigyval);
# Now we have a half-path - Flip and append to get entire path
xneg=np.array(x);
xneg=-xneg;
xneg=xneg.tolist();
xneg.reverse();
x=xneg+x;

# Use symmetry of elastic solution to get negative quadrant path
epsz=epsz[::-1]+epsz;
epsx=epsx[::-1]+epsx;
epsy=epsy[::-1]+epsy;
# tau is antisymmetric
gammeg=np.array(gamxz[::-1]);
gammeg=-gammeg;
gammeg=gammeg.tolist();
gamxz=gammeg+gamxz;

# Use symmetry of elastic solution to get negative quadrant path
sigz=sigz[::-1]+sigz;
sigx=sigx[::-1]+sigx;
sigy=sigy[::-1]+sigy;
# tau is antisymmetric
tauneg=np.array(tauxz[::-1]);
tauneg=-tauneg;
tauneg=tauneg.tolist();
tauxz=tauneg+tauxz;

strpaths=[x,epsx,epsz,epsy,gamxz,sigx,sigy,sigz,tauxz];
return strpaths;
import math
import matplotlib.pyplot as plotit
import numpy as np
from scipy.optimize import fsolve
from GetStrainPaths import StrainPaths

def SolveStressPlastic(siginc):
    delsigt=siginc[0];
    delsigy=siginc[1];
    delsigtz=siginc[2];
    deltauxz=siginc[3];

    # Plane strain condition

    eq1=(nu-(E/h)*nx*ny)*delsigt+(nu-(E/h)*ny*nz)*delsigtz-(2*E/h)*nx*ny*deltauxz-(1+(E/h))*(ny**2))*delsigy;
    # Total Strain increment equations
    delSx=delsigt-(1.0/3.0)*(delsigt+delsigy+delsigtz);
    delSy=delsigy-(1.0/3.0)*(delsigt+delsigy+delsigtz);
    delSz=delsigtz-(1.0/3.0)*(delsigt+delsigy+delsigtz);
    delSxz=deltauxz;

    term=delSx*nx+delSy*ny+delSz*nz+2*delSxz*nxz;
    MacBrac=0.5*(term+math.fabs(term));

    # Only elastic and plastic increments
    eq2=delxpsz-(1.0/h)*MacBrac*nx-(1.0/E)*delsigt+(nu/E)*(delsigt+delsigtz);
    eq3=delxpsz-(1.0/h)*MacBrac*nx-(1.0/E)*delsigtz+(nu/E)*(delsigt+delsigtz);
    eq4=delxpsz-(1.0/h)*MacBrac*nxz-(1.0/G)*deltauxz;

    eq=[eq1,eq2,eq3,eq4];
    stressinc=[delsigt,delsigy,delsigtz,deltauxz];
    return eq;

global nu;
geglobal G;
geglobal E;
geglobal h;
num=0.3;
p0=6.0*606.0;
k=1.0/6.0;
G=79600.0; # Normalized against this
G=G/p0;
E=G*2*(1+nu);
h=25.7*G;
M=200; # Number of relaxation steps

z=[float(0.0001)];
zmax=3.0;
diff=0.05;
count=int(zmax/diff);
for i in range(count):
    z.append(float(i+1)*diff);

#Compute the elasto-plastic response for the first pass
numpaths=15;
p=0;
flag=0;
sigyrarr=[];
sigxarr=[];
#alpxarr=[];
#Kinematic hardening center

global delepsx;
global delepsz;
global gamxz;
global nx;
global ny;
global nz;
global nxz;

MJfile1=open('MJsigydata.csv','w');
MJfile2=open('MJsigxdata.csv','w');

for zval in z:
    strpaths=StrainPaths(nu,zval,G);
    #First list is x, then sigx, sigz, sigy and tauxz lists
    x=strpaths[0];
    epsx=strpaths[1]; # a list
    epsz=strpaths[2];
    #epsy=strpaths[3]; #All zeroes
    gamxz=strpaths[4];

    #Also get the elastic stress solutions
    sigx=strpaths[5];
    sigy=strpaths[6];
    sigz=strpaths[7];
    tauxz=strpaths[8];
    #print sigx[0]
    sigxr=0.0;
    sigyr=0.0;
    sigzr=0.0;
    tauxxr=0.0;
    epsxr=0.0;
    epsyr=0.0;
    gamxzar=0.0;
    epszr=0.0;

    alpx=0.0;
    alpy=0.0;
    alpz=0.0;
alpxz=0.0;
epsxp=0.0;
epsyp=0.0;
epszpz=0.0;
gamxzp=0.0;
delsigx=0.0;
delsigy=0.0;
delsigz=0.0;
deltauxz=0.0;

p=0;
while p < numpaths:
  p=p+1;
  # Compute residual stresses at each depth
  # sigy[0]=sigr
  for i in range(len(x)-1):
    delepsx=epsx[i+1]-epsx[i];
    delepsz=epsz[i+1]-epsz[i];
    delgamxz=gammxz[i+1]-gammxz[i];
    #delepsy=0.0 is required by the formulation
    if i == 0:
      # Merwin-Johnson
      epsxval=epsx[i]+(1.0/E)*(sigr-nu*sigy);
      epszval=epsz[i]+(nu/E)*(sigr+sigy);
      # Include Plastic strain - Recover Jiang-Sehitoglu Solution
      epsxval=epsx+epsx[i]+(1.0/E)*(sigr-nu*sigy);
      epszval=epsz+epsz[i]+(nu/E)*(sigr+sigy);
      # epsyval is always zero - plane strain
      gamxval=gammxz[i]+gammzp;
      sigxval=sigx+siggx[i];
      sigyval=sigy+siggz[i];
      sigzval=sigz[i];  # No residuals accumulated
      tauxxval=tauxx[i];  # No residuals accumulated
    #
    else:
      sigyval=sigy[i];  # Previous increment already includes the
      # residual stress
      # For the remaining steps the increment is computed
      # automatically
      # Check if yielding
      Sx=(sigxval-(1.0/3.0)*(sigxval+sigyval+sigzval));
      Sy=(sigyval-(1.0/3.0)*(sigxval+sigyval+sigzval));
      Sz=(sigzval-(1.0/3.0)*(sigxval+sigyval+sigzval));
      Sxz=tauxxval;

      nx=(Sx-alpx)/(math.sqrt(2)*k);
      ny=(Sy-alpy)/(math.sqrt(2)*k);
      nz=(Sz-alpz)/(math.sqrt(2)*k);
      nxz=(Sxz-alpxz)/(math.sqrt(2)*k);
      f=(nx**2)+(ny**2)+(nz**2)+2*(nxz**2)-1;
if f >= 0:
    # Plastic increment procedures
    flag=1;
    # Plastic yielding

delsigsol=fsove(SolvStressPlastic,[delsigx,delsigy,delsigz, deltauxz]);
    # err=SolvStressPlastic(delsigsol);
    # print max(err) ; # Always zero - Solver Ok
    delsigx=delsigsol[0];
    delsigy=delsigsol[1];
    delsigz=delsigsol[2];
    deltauxz=delsigsol[3];

delSx=delsigx-(1.0/3.0)*(delsigx+delsigy+delsigz);
    delSy=delsigy-(1.0/3.0)*(delsigx+delsigy+delsigz);
    delSz=delsigz-(1.0/3.0)*(delsigx+delsigy+delsigz);
    delSxz=deltauxz;

term=delSx*nx+delSy*ny+delSz*nz+2*delSxz*nxz;
    MacBrac=0.5*(term+math.fabs(term));

delepsxp=(1.0/h)*MacBrac(nx;
    delepszp=(1.0/h)*MacBrac*ny;
    delepsy=(1.0/h)*MacBrac*ny;
    delgamxzp=(2.0/h)*MacBrac*nxz ; # Engineering strain

delalpx=MacBrac*nx;
    delalpy=MacBrac*ny;
    delalpz=MacBrac*nz;
    delalpxz=MacBrac*nxz;
else:
    flag=0;
    delsigx=(E/((1+nu)*((1-2*nu))*(1-nu)*delepsx+nu*delepsz);
    delsigy=(E/((1+nu)*((1-2*nu))*(nu*delepsx+nu*delepsz);
    delsigz=(E/((1+nu)*((1-2*nu))*(1-nu)*delepsz+nu*delepsx);
    deltauxz=G*delgamxz;

delepsxp=0.0;
    delepszp=0.0;
    delepsy=0.0;
    delgamxzp=0.0;

delalpx=0.0;
    delalpy=0.0;
    delalpz=0.0;
    delalpxz=0.0;

    # Update the plastic parameters
    alpx=alpx+delalpx;
    alpy=alpy+delalpy;
    alpz=alpz+delalpz;
alpxz=alpxz+delalpxz;

epsxp=epsxp+delepsxp;
epsyp=epsyp+delepsyp;
epszp=epszp+delepszp;
gamxzp=gamxzp+delgamxzp;
sigyval=sigyval+delsigy;
sigxval=sigxval+delsigx;
sigzval=sigzval+delsigz;
tauxzval=tauxzval+deltauxz;

epsxval=epsxval+delepsx;
epszval=epszval+delepsz;
gamxzval=gamxzval+delgamxz;

# Relaxation procedure
# Remember that sigx and sigy accumulate residuals

sigzb=sigzval-sigz[i];
tauxzb=tauxzval-tauxz[i];
#epsx=epsxp+sigxval-nu*(sigzval+sigyval);
epsxb=epsxval-epsx[i];

# Compute relaxation increments
delepsx=(-epsxb)/M;
delsigz=(-sigzb)/M;
deltauxz=(-tauxzb)/M;

for j in range(M):
    # Use total stress values for current stress state
    Sx=(sigxval-1.0/3.0)*((sigxval+sigyval+sigzval));
    Sy=(sigyval-1.0/3.0)*((sigxval+sigyval+sigzval));
    Sz=(sigzval-1.0/3.0)*((sigxval+sigyval+sigzval));
    Sxz=tauxzval;
    nx=(Sx-alpx)/(math.sqrt(2)*k);
    ny=(Sy-alpy)/(math.sqrt(2)*k);
    nz=(Sz-alpz)/(math.sqrt(2)*k);
    nxz=(Sxz-alpxz)/(math.sqrt(2)*k);
    f=nx**2+ny**2+nz**2+2*nxz**2-1;
    if f >= 0:
        # Plastic relaxation
        # Print "Plastic relaxation"
        rhs1=E*delepsx+(nu-E*nx*nz/h)*delsigz-2*(E/h)*nx*nxz*deltau;
        rhs2=(nu-E*ny*nz/h)*delsigz-2*E*ny*nxz*deltauxz/h;
        coeff1=(1+E*nx**2/h);
        coeff2=(E*nx*ny/h)-nu;
        coeff3=(E*nx*ny/h)-nu;
        coeff4=(1+E*ny**2/h);
        det=coeff1*coeff4-coeff3*coeff2;
# print det

delsigx=(rhs1*coeff4-rhs2*coeff2)/(coeff1*coeff4-coeff3*coeff2);
delsigy=(rhs1*coeff3-rhs2*coeff1)/(coeff2*coeff3-coeff4*coeff1);
delSx=delsigx-(1.0/3.0)*(delsigx+delsigy+delsigz);
delSy=delsigy-(1.0/3.0)*(delsigx+delsigy+delsigz);
delSz=delsigz-(1.0/3.0)*(delsigx+delsigy+delsigz);
delSxzn=delSx*nx+delSy*ny+delSz*nz+2*delSxz*nxz;
MacBrac=0.5*(term+math.fabs(term));
delepsxp=(1.0/h)*MacBrac*nx;
delepszp=(1.0/h)*MacBrac*nz;
delepsyp=(1.0/h)*MacBrac*ny;
delgamxz=(2.0/h)*MacBrac*nxz;
delalpx=MacBrac*nx;
delalpy=MacBrac*ny;
delalpz=MacBrac*nz;
delalpxz=MacBrac*nxz;

else:

# print "Elastic Relaxation"
# print i
rhs1=E*delepsx+nu*delsigz;
rhs2=nu*delsigz;
coeff1=1;
coeff2=-nu;
coeff3=-nu;
coeff4=1;

# print (coeff1*coeff4-coeff3*coeff2)

delsigx=(rhs1*coeff4-rhs2*coeff2)/(coeff1*coeff4-coeff3*coeff2);
delsigy=nu*(delsigx+delsigz);
delepsxp=0.0;
delepszp=0.0;
delepsyp=0.0;
delgamxz=0.0;
delalpx=0.0;
delalpy=0.0;
delalpz=0.0;
delalpxz=0.0;

# Update plastic parameters
alpx=alpx+delalpx;
alpy=alpy+delalpy;
alpz=alpz+delalpz;
alpxz=alpxz+delalpxz;

epsxp=epsxp+delepsxp;
epsyp=epsyp+delepsyp;
epszp=epszp+delepszp;
gamxzp=gamxzp+delgamxzp;

#Update stress state for next iteration of relaxation
sigyval=sigyval+delsigy;
sigxval=sigxval+delsigx;
sigzval=sigzval+delsigz;
tauxzval=tauxzval+deltauxz;

sigxrsigxval-sigx[i]; #Residual accumulates
sigrs=sigyval-sigy[i]; #Residual accumulates

sigyrarr.append(6.0*sigyr);
sigxarr.append(6.0*sigxr);
MJfile1.write('{},{}
'.format(6.0*sigyr,zval));
MJfile2.write('{},{}
'.format(6.0*sigxr,zval));

MJfile1.close();
MJfile2.close();

plotit.plot(sigyrarr,z,label='$$\sigma_{y,r}/k$$');
plotit.plot(sigxarr,z,label='$$\sigma_{x,r}/k$$');
#plotit.xlabel('$$\sigma_{y,r}/k$$');
plotit.ylabel('$$\sigma/a$$');
plotit.legend();
#plotit.plot(sigxarr,z)
plotit.ylim(3,0);
#plotit.plot(x[0:(len(x)-1)],alpxarr);
plotit.show();