```
#!/usr/bin/python

import numpy
import math
import scipy.linalg

#014
########
# coordinates
coord = numpy.array(
[[ 0.35897],
 [ 0.74314],
 [ 0.63191]])

# variances/covariances of coordinates
coordcovar = numpy.array(
[[  1.21000e-08,  3.49272e-10, -7.08697e-10],
 [  3.49272e-10,  1.44000e-08, -3.03336e-10],
 [ -7.08697e-10, -3.03336e-10, 1.21000e-08]])


# Ucif
Ucif = numpy.array(
[[ 0.04682, 0.00258, -0.00735],
 [ 0.00258, 0.02399, -0.00165],
 [-0.00735, -0.00165, 0.05938]])


# 9x9 variances/covariances matrix of Ucif
# index: U11 U12 U13 U12 U22 U23 U13 U23 U33
Ucifcovar = numpy.array(
[[  7.2250e-07, 1.7666e-08,-4.2338e-08, 1.7666e-08,-1.1325e-07, 2.5383e-09,-4.2338e-08, 2.5383e-09, -2.1266e-07],
 [  1.7666e-08, 3.9690e-07,-7.1351e-09, 3.9690e-07, 1.0346e-08,-9.7002e-09,-7.1351e-09,-9.7002e-09, -1.8135e-09],
 [ -4.2338e-08,-7.1351e-09, 5.1840e-07,-7.1351e-09, 1.0800e-08, 1.3041e-08, 5.1840e-07, 1.3041e-08, -3.4615e-08],
 [  1.7666e-08, 3.9690e-07,-7.1351e-09, 3.9690e-07, 1.0346e-08,-9.7002e-09,-7.1351e-09,-9.7002e-09, -1.8135e-09],
 [ -1.1325e-07, 1.0346e-08, 1.0801e-08, 1.0346e-08, 4.9000e-07,-6.8267e-09, 1.0801e-08,-6.8267e-09, -1.6216e-07],
 [  2.5383e-09,-9.7002e-09, 1.3041e-08,-9.7002e-09,-6.8266e-09, 3.9690e-07, 1.3041e-08, 3.9690e-07, -1.3986e-08],
 [ -4.2338e-08,-7.1351e-09, 5.1840e-07,-7.1351e-09, 1.0800e-08, 1.3041e-08, 5.1840e-07, 1.3041e-08, -3.4615e-08],
 [  2.5383e-09,-9.7002e-09, 1.3041e-08,-9.7002e-09,-6.8267e-09, 3.9690e-07, 1.3041e-08, 3.9690e-07, -1.3986e-08],
 [ -2.1266e-07,-1.8135e-09,-3.4615e-08,-1.8135e-09,-1.6216e-07,-1.3986e-08,-3.4615e-08,-1.3986e-08, 1.0201e-06]])

# cell parameters
a = 13.8593
b = 10.5242
c = 14.8044
alpha = 90
beta = 92.0014
gamma = 90
alpha=alpha*math.pi/180.0
beta=beta*math.pi/180.0
gamma=gamma*math.pi/180.0

# transform a 9x9 var./cov. matrix for the U into a 6x6 var./cov. matrix with the shelx format
# Used for pretty printing purpose
conv=numpy.zeros((6,9))
conv[0,0]=1
conv[1,4]=1
conv[2,8]=1
conv[3,3]=1
conv[4,2]=1
conv[5,7]=1

def calculate(transform):
    """ Calculation of all the transformed parameters.
    Take the transformation matrix on the coordinates as an argument"""

    # metric tensor
    # Sands, D. E. (1995). Vectors and Tensors in Crystallography. New York: Dover Publications, Inc., 2nd ed.
    # p. 72
    g=numpy.zeros((3,3))
    g[0,0]=a*a
```

```
g[0,1]=a*b*math.cos(gamma)
g[0,2]=a*c*math.cos(beta)
g[1,0]=g[0,1]
g[1,1]=b*b
g[1,2]=b*c*math.cos(alpha)
g[2,0]=g[0,2]
g[2,1]=g[1,2]
g[2,2]=c*c

# The inverse of the metric tensor
ig=numpy.linalg.inv(g)

# N matrix
# Grosse-Kunstleve, R.W. & Adams, P. D. (2002). J. Appl. Cryst. 35(4), 477-480.
# numpy.diag extracts the diagonal terms,
# numpy.diagflat makes a diagonal matrix with given diagonal terms
N=numpy.diagflat(numpy.sqrt(numpy.diag(ig)))

# Calculation of the new metric tensor for the transformed cell
# The transpose of the tranformation matrix need to be used
newmetric=numpy.dot(numpy.dot(transform.T,g),transform)
# reciprocal cell parameters for the tranformed cell
newimetric=numpy.linalg.inv(newmetric)
newN=numpy.diagflat(numpy.sqrt(numpy.diag(newimetric)))
newiN=numpy.linalg.inv(newN)
# newiN is a diagonal matrix containing 1/a*, 1/b*, 1/c*
# Used to calculate Ucif from U*

# new cell parameters are derived from the transformed metric tensor
newa = math.sqrt(newmetric[0,0])
newb = math.sqrt(newmetric[1,1])
newc = math.sqrt(newmetric[2,2])
newalpha = math.acos(newmetric[2,1]/(newb*newc))
newbeta = math.acos(newmetric[2,0]/(newa*newc))
newgamma = math.acos(newmetric[1,0]/(newa*newb))
print
print "Cell parameters:"
print "a: % .5f"%newa
print "b: % .5f"%newb
print "c: % .5f"%newc
print "alpha: % .5f"%math.degrees(newalpha)
print "beta: % .5f"%math.degrees(newbeta)
print "gamma: % .5f"%math.degrees(newgamma)

# Orthogonalization matrix
# Dunitz, J. D. X-Ray Analysis and the Structure of Organic Molecules Verlag Helvetica Chimica Acta, 1995
# page 237
newvolume=newa*newb*newc*math.sqrt(1+2*math.cos(newalpha)*math.cos(newbeta)*
    math.cos(newgamma)-math.cos(newalpha)**2-math.cos(newbeta)**2-math.cos(newgamma)**2)
newcrystaltocartesian=numpy.zeros((3,3))
newcrystaltocartesian[0,0]=newa
newcrystaltocartesian[0,1]=newb*math.cos(newgamma)
newcrystaltocartesian[0,2]=newc*math.cos(newbeta)
newcrystaltocartesian[1,1]=newb*math.sin(newgamma)
newcrystaltocartesian[1,2]=newc*(math.cos(newalpha)-math.cos(newbeta)*math.cos(newgamma))/math.sin(newgamma)
newcrystaltocartesian[2,2]=newvolume/(newa*newb*math.sin(newgamma))

# Calculation of U* and variances/covariances matrix of U* for the current setting
Ustar = numpy.dot(numpy.dot(N,Ucif),N.T)
kron = numpy.kron(N,N.T) #kronecker product
Ustarcovar = numpy.dot(numpy.dot(kron,Ucifcovar),kron.T)

# New coordinates in the transformed cell
newcoord = numpy.dot(transform,coord)
# move coordinates in the unit cell
for j in range(3):
    while(newcoord[j,0]<0):
        newcoord[j,0]+=1
    while(newcoord[j,0]>1):
        newcoord[j,0]-=1
print ""
```

```
print "coordinates: % .5f, % .5f, % .5f"%(tuple(newcoord))

# New coordinates variances/covariances matrix
newcoordcovar = numpy.dot(numpy.dot(transform,coordcovar),transform.T)
esd=numpy.sqrt(numpy.diag(newcoordcovar))
print "esds      : % .5f, % .5f, % .5f"%(tuple(esd))

# New U*
newUstar = numpy.dot(numpy.dot(transform,Ustar),transform.T)

# new U* variances/covariances matrix
kron = numpy.kron(transform,transform)
newUstarcovar = numpy.dot(numpy.dot(kron,Ustarcovar),kron.T)

# new Ucif and Ucif variances/covariances matrix
newUcif = numpy.dot(numpy.dot(newiN,newUstar),newiN.T)
kron=numpy.kron(newiN,newiN)
newUcifcovar=numpy.dot(numpy.dot(kron,newUstarcovar),kron.T)
esd=numpy.sqrt(numpy.diag(numpy.dot(numpy.dot(conv,newUcifcovar),conv.T)))

print ""
print "Ucif: % .5f, % .5f, % .5f, % .5f, % .5f, % .5f"%(
    newUcif[0,0], newUcif[1,1], newUcif[2,2],
    newUcif[1,2], newUcif[0,2], newUcif[0,1])
print "esds: % .5f, % .5f, % .5f, % .5f, % .5f, % .5f"%(tuple(esd))

# calculate Ueq
# R. X. Fischer and E. Tillmanns, Acta Cryst. (1988). C44, 775-776
symmetry99a=numpy.kron(newcrystaltocartesian,newcrystaltocartesian)
newUstarcovarUeq=numpy.dot(numpy.dot(symmetry99a,newUstarcovar),symmetry99a.T)
newUvaluesUeq=numpy.dot(numpy.dot(newcrystaltocartesian,newUstar),newcrystaltocartesian.T)
Ueq=1.0/3.0*numpy.sum(newUvaluesUeq.diagonal())
sUeq=1.0/3.0*numpy.sqrt(newUstarcovarUeq[0,0]+newUstarcovarUeq[4,4]+
    newUstarcovarUeq[8,8]+2*newUstarcovarUeq[0,4]+2*newUstarcovarUeq[0,8]+
    2*newUstarcovarUeq[4,8])
print 'Ueq(sUeq): % .5f(% .5f)'%(Ueq,sUeq)

# new eigenvalues
# scipy.linalg.eigh solved generalised eigenvalue/eigenvector problem on hermitian (symmetric) matrices
eva = scipy.linalg.eigh(newUstar, newimetric)[0] #eigenvalues
eve = scipy.linalg.eigh(newUstar, newimetric)[1] #eigenvectors
ieve = numpy.linalg.inv(eve) #inverse
print
print "eigenvalues : % .5f, % .5f, % .5f"%(tuple(eva))

# new eigenvalues variances/covariances matrix
kron = numpy.kron(eve.T,numpy.linalg.inv(numpy.dot(newimetric,eve))) #kronecker product
evacovar=numpy.dot(numpy.dot(kron,newUstarcovar),kron.T)
esd=numpy.sqrt(numpy.diag(evacovar))
print "esds      : % .5f, % .5f, % .5f"%(esd[0],esd[4], esd[8])

print
print "Correlation matrix (shelx order)"
print "[[ x     y     z     U11   U22   U33   U12   U13   U32 ]"
numpy.set_printoptions(precision=3, linewidth=150, suppress=True)
tmp=1./numpy.sqrt(numpy.diag(newUcifcovar))
tmp2=numpy.diagflat(tmp)
# tmp2 is a diagonal matrix with the sigmas,
# this matrix is used to convert the var./cov/ matrix to a correlation matrix
# 9x9 correlation matrix: corrmat = tmp2 * newUcifcovar * tmp2
# 6x6 shelx format correlation matrix = conv * corrmat * conv.T
a1=numpy.dot(numpy.dot(conv, numpy.dot(numpy.dot(tmp2,newUcifcovar),tmp2)),conv.T)
# idem for the coordinates:
tmp=1./numpy.sqrt(numpy.diag(newcoordcovar))
tmp2=numpy.diagflat(tmp)
a2=numpy.dot(numpy.dot(tmp2,newcoordcovar),tmp2)
result=numpy.zeros((9,9))
result[0:3,0:3]=a2
result[3:9,3:9]=a1
print result
```

```
print "======================="
print "  atom O14"
print "======================="
print

# Identity matrix, just to ouput original data
transform = numpy.array(
[[1,0,0],
 [0,1,0],
 [0,0,1]])

print
print '---------------------------'
print "No transformation - P 21/n"
print '---------------------------'
print
calculate(transform)

# transformation matrix of the coordinates from P21/n to P21/c
transform = numpy.array(
[[1,0,-1],
 [0,-1,0],
 [0,0,-1]])

print
print '---------------------------'
print "After transformation P 21/c"
print '---------------------------'
print
calculate(transform)

# original data in P21/c for comparison
print
print '---------------------------'
print "Original data in P 21/c"
print '---------------------------'
print

print "Cell parameters:"
print "a: 13.8594(2)"
print "b: 10.5243(2)"
print "c: 19.9230(3)"
print "alpha: 90.00"
print "beta: 132.0440(10)"
print "gamma: 90.00"
print
print "coordinates: 0.72703, 0.25684, 0.36807"
print "esds     :  0.00016, 0.00012, 0.00011"
print
print "Ucif: 0.0623,  0.0237,  0.0592, -0.0019,  0.0468, -0.0031"
print "esds: 0.0010,  0.0007,  0.0010,  0.0006,  0.0009,  0.0006"
print "Ueq(sUeq): 0.0434( 0.0004)"
print
print "eigenvalues : 0.02341, 0.04214, 0.06479"
print "esds      :  0.00069, 0.00083, 0.00103"
print
print "[[ x    y     z     U11   U22   U33   U12   U13   U32 ]"
print "\
[[ 1.   -0.034 0.728 0.    0.    0.    0.    0.    0.  ] \n\
 [-0.034 1.   -0.025 0.    0.    0.    0.    0.    0.  ] \n\
 [ 0.728 -0.025 1.    0.    0.    0.    0.    0.    0.  ] \n\
 [ 0.    0.    0.    1.   -0.218 0.410 -0.039 0.797 -0.029] \n\
 [ 0.    0.    0.   -0.218 1.   -0.227 -0.032 -0.188 -0.020] \n\
 [ 0.    0.    0.    0.410 -0.227 1.   -0.013 0.806 -0.024] \n\
 [ 0.    0.    0.   -0.039 -0.032 -0.013 1.  -0.029 0.696] \n\
 [ 0.    0.    0.    0.797 -0.188 0.806 -0.029 1.   -0.035] \n\
 [ 0.    0.    0.   -0.029 -0.020 -0.024 0.696 -0.035 1. ]]"
```