basic_example

Unknown Author

October 28, 2015

In [14]:
```
import pylab
import pyasf
import sympy as sp
```

In [4]:
```
cs = pyasf.unit_cell("LiNbO3_R3cH_Abrahams86_ICSD_61118.cif", resonant="Nb")
```

```
Multiple settings found in space group 161
  Identified symbol 'R3c:h' from .cif entry 'r3c:h'
Trigonal (hexagonal setting)
```

In [5]:
```
cs.AU_positions # positions in asymmetric unit
```

Out [5]:
```
{'Li1': array([0, 0, 0.278720000000000], dtype=object),
 'Nb1': array([0, 0, 0], dtype=object),
 'O1': array([0.0475700000000000, 23/67, 0.0633600000000000],
dtype=object)}
```

In [8]:
```
cs.AU_formfactorsDDc["Nb1"] # cartesian representation of dipole dipole form factor tensor in asymmetric unit
```

Out [8]:
```
array([[f_Nb1_dd_11, f_Nb1_dd_12, f_Nb1_dd_13],
       [f_Nb1_dd_12, f_Nb1_dd_22, f_Nb1_dd_23],
       [f_Nb1_dd_13, f_Nb1_dd_23, f_Nb1_dd_33]], dtype=object)
```

In [9]:
```
cs.get_tensor_symmetry() # apply all symmetry constraints of space group to all tensors (debye waller U_ij, f_ij)
```

In [10]:
```
cs.AU_formfactorsDDc["Nb1"] # now with symmetry
```

Out [10]:
```
array([[f_Nb1_dd_22, 0, 0],
       [0, f_Nb1_dd_22, 0],
       [0, 0, f_Nb1_dd_33]], dtype=object)
```

```
cs.U["Nb1"] # symmetry of U is different due to differeny basis
```

In [12]:
```
Matrix([
[  U_Nb1_22, U_Nb1_22/2,         0],
[U_Nb1_22/2,   U_Nb1_22,         0],
[         0,          0, U_Nb1_33]])
```

Out [12]:

```
cs.AU_formfactorsDD["Nb1"] # representation of dipole dipole form factor tensor in crystal basis
```

In [13]:
```
array([[f_Nb1_dd_22, -f_Nb1_dd_22/2, 0],
       [-f_Nb1_dd_22/2, f_Nb1_dd_22, 0],
       [0, 0, f_Nb1_dd_33]], dtype=object)
```

Out [13]:

```
vector = sp.Matrix([1,1,1]) # create random vector
print cs.M * vector # the same vector in cartesian system defined as in Trueblood: doi:10.1107/S0108767396005697
```

In [16]:
```
Matrix([
[        a/2],
[sqrt(3)*a/2],
[          c]])
```

Out [16]:

```
cs.build_unit_cell() # construct unit cell from asymmetric unit
F0 = cs.calc_structure_factor((0,0,3), Temp=False) # calculate structure factor without temperature
```

In [40]:

```
print F0.n().simplify() # evaluate structure factor ==> forbidden reflection
```

In [28]:
```
0
```

Out [28]:

```
cs.transform_structure_factor()
cs.calc_scattered_amplitude() # calculate Structure Factor for higher orders tensors
```

In [34]:
```
True
```

Out [34]:

```
print cs.E["ss"] # also zero for sigma sigma scattering
```

In [32]:
```
0
```

```
print cs.E["sp"] # some dipole quadrupole scattering in sigma pi scattering channel
```

In [33]:
```
-6.0*I*f_Nb1_dq_z21*sqrt(1 -
1801363.07711506/epsilon**2)*sqrt(epsilon**2 -
1801363.07711506)/Abs(epsilon)
```

```
Fhot = cs.calc_structure_factor((0,0,3), Temp=True) # calculate structure factor with temperature
```

In [35]:

```
print Fhot.n().simplify() # also no temperature scattering
```

In [37]:
```
0
```

```
cs.Q * cs.M * vector # same vector as before but in laboratory system as defined in doi:10.1107/S0108767391011509
# others can be defined...
```

In [80]:

```
Out [80]: Matrix([
          [          c],
          [sqrt(3)*a/2],
          [       -a/2]])
```

Now lets check some more basic functionality

```
In [42]: cs.subs # this dictionary contains all (momentary) values
Out [42]: {l: 3, a: 5.14739, c: 13.85614, h: 0, k: 0}

In [44]: cs.Uaniso # but here the values for U from the ciffile
Out [44]: {'Li1': Matrix([
          [ 0.0113, 0.00566,    0.0],
          [0.00566, 0.01132,    0.0],
          [    0.0,     0.0, 0.0176]]),
           'Nb1': Matrix([
          [0.00361, 0.00181,     0.0],
          [0.00181, 0.00361,     0.0],
          [    0.0,     0.0, 0.00323]]),
           'O1': Matrix([
          [ 0.00674,  0.00297, -0.00102],
          [ 0.00297,   0.0054, -0.00203],
          [-0.00102, -0.00203,  0.00673]])}

In [45]: cs.hkl() # current reflection?
Out [45]: (0, 0, 3)

In [48]: cs.set_temperature # function that can be used to calculate ADPs from debye temperature or einstein temperature onc
Out [48]: <bound method unit_cell.set_temperature of <pyasf.pyasf.unit_cell
          object at 0x7f83dc305850>>

In [49]: cs.charges
Out [49]: defaultdict(<type 'int'>, {'Nb1': 5, 'Li1': 1, 'O1': -2})

In [50]: cs.elements
Out [50]: {'Li1': 'Li', 'Nb1': 'Nb', 'O1': 'O'}

In [51]: cs.occupancy
Out [51]: {'Li1': 1.0, 'Nb1': 1.0, 'O1': 1.0}

In [54]: cs.positions["Li1"] # all positions in unit cell
```

```
Out [54]: [array([0, 0, 0.278720000000000], dtype=object),
           array([0, 0, 0.778720000000000], dtype=object),
           array([2/3, 1/3, 0.612053333333333], dtype=object),
           array([2/3, 1/3, 0.112053333333333], dtype=object),
           array([1/3, 2/3, 0.945386666666667], dtype=object),
           array([1/3, 2/3, 0.445386666666667], dtype=object)]
```

```
In [55]: cs.get_density()
Out [55]: 4.6329431433622155
```

```
In [56]: cs.get_stoichiometry()
Out [56]: 'NbO3Li'
```

```
In [58]: cs.get_nearest_neighbors("Li1", 8) # returns labels, distance and difference vector
Out [58]: (array(['O1', 'O1', 'O1', 'O1', 'O1', 'O1', 'Nb1', 'Nb1'],
                dtype='|S3'),
           array([ 2.04978164,  2.04978164,  2.04978164,  2.27114128,
          2.27114128,
                   2.27114128,  3.06608666,  3.06667815]),
           array([[-1.00593791, -1.65362372,  0.67470164],
                  [-0.92911119,  1.69797964,  0.67470164],
                  [ 1.9350491 , -0.04435593,  0.67470164],
                  [ 0.92911119, -1.27386736, -1.63465502],
                  [ 0.6386459 ,  1.44156758, -1.63465502],
                  [-1.56775709, -0.16770022, -1.63465502],
                  [ 0.        ,  0.        , -3.06608666],
                  [-2.573695  , -1.4859235 , -0.75672999]]))
```

```
In [61]: cs.get_thermal_ellipsoids("Nb1") # eigenvalues and eigenvectors for ADP of Nb
Out [61]: (array([ 0.0026764 ,  0.00323   ,  0.00486026]),
           array([[-0.14658319,  0.        , -0.98919835],
                  [-0.98919835,  0.        ,  0.14658319],
                  [-0.        ,  1.        ,  0.        ]]))
```

```
In [62]: cs.multiplicity
Out [62]: defaultdict(<type 'int'>, {'Nb1': 6, 'Li1': 6, 'O1': 18})
```

```
In [64]: cs.dE # edge shift
Out [64]: {'Li1': 0, 'Nb1': 0, 'O1': 0}
```

```
In [73]: gen = cs.iter_rec_space(0.3) # iterator for all reflections in a ewald volume 2*sin(th)/lambda < 0.3
```

```
In [71]: print list(gen)
[(1, 0, 2), (1, 0, 1), (1, 0, 0), (1, 0, -1), (1, 0, -2), (1, -1, 2),
(1, -1, 1), (1, -1, 0), (1, -1, -1), (1, -1, -2), (0, 0, 4), (0, 0,
3), (0, 0, 2), (0, 0, 1), (0, 0, 0), (0, 0, -1), (0, 0, -2), (0, 0,
-3), (0, 0, -4)]

In [76]: cs.weights
{'Li': 6.941, 'Nb': 92.906, 'O': 15.999}
Out [76]:

In [77]: cs.metric_tensor
Out [77]: Matrix([
[   a**2, -a**2/2,    0],
[-a**2/2,    a**2,    0],
[      0,       0, c**2]])

In [78]: cs.metric_tensor_inv
Out [78]: Matrix([
[4/(3*a**2), 2/(3*a**2),       0],
[2/(3*a**2), 4/(3*a**2),       0],
[         0,          0, c**(-2)]])

In [79]: cs.metric_tensor_inv.subs(cs.subs) # the generic way to insert all current values
Out [79]: Matrix([
[0.0503227756700119, 0.0251613878350059,                   0],
[0.0251613878350059, 0.0503227756700119,                   0],
[                 0,                  0, 0.00520853365300592]])
```

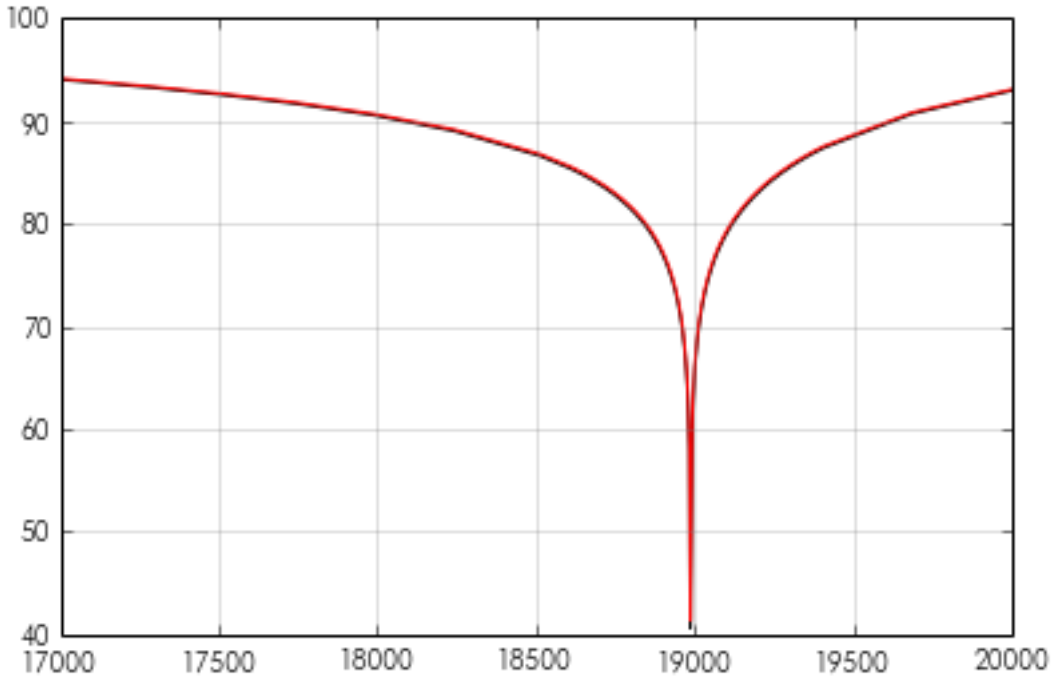
Units used are: Anstrom and eV

```
In [81]: cs.F_DD
Out [81]: array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]], dtype=object)

In [91]: func = cs.get_reflection_angles((0,0,1), 8000) # get function to calculat angle to surface and azimuth for each ref
000
001
0k0
0kl
h00
h0l
hk0
hkl
```

```
In [132]: %pylab inline
Populating the interactive namespace from numpy and matplotlib
WARNING: pylab import has clobbered these variables: ['pylab']
`%pylab --no-import-all` prevents importing * from pylab and numpy

In [133]: map(degrees, func((0,1,1), 8000, 0)) # (h,k,l), energy, second azimuth
Out [133]: [79.973820872792956, -3.2058163933588304]
```

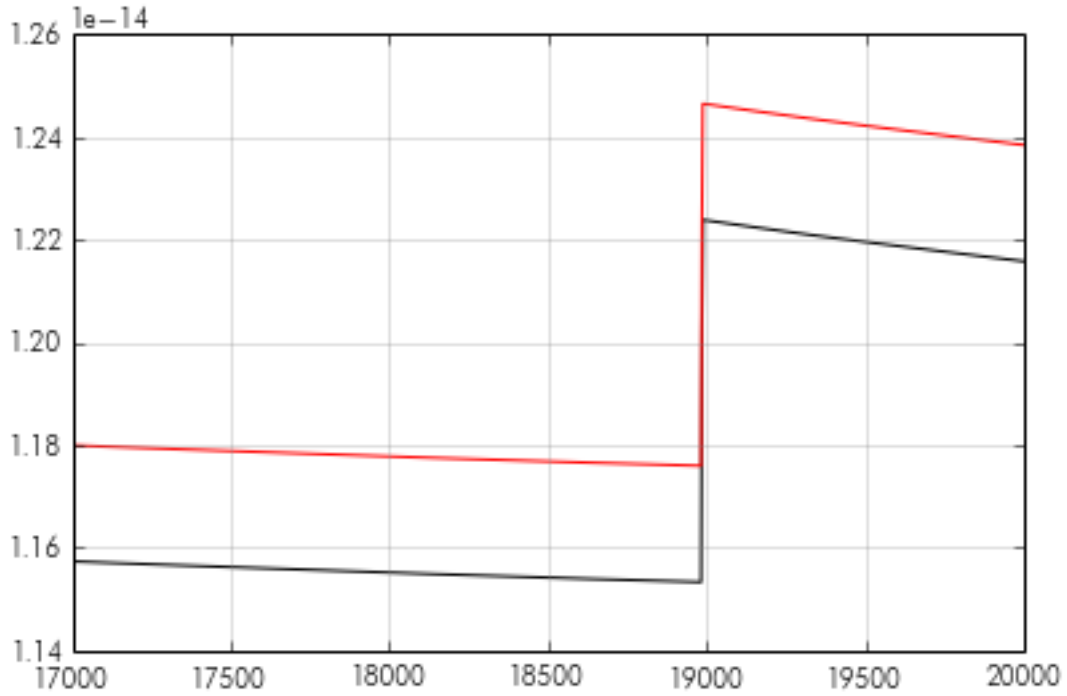
Example to get real intensities:

```
In [125]: cs.DAFS(8048, (0,0,3))
Out [125]: array([ -9.25832124e-15 -5.47299240e-15j])

In [126]: cs.DAFS(8048, (0,0,6))
Out [126]: array([ 103.59889715+75.83713276j])

In [142]: Energy = linspace(17000, 20000, 1001)

In [148]: plot(Energy, cs.DAFS(Energy, (0,0,6), Temp=False))
          plot(Energy, cs.DAFS(Energy, (0,0,6), Temp=True))
Out [148]: [<matplotlib.lines.Line2D at 0x7f83d1186810>]
```



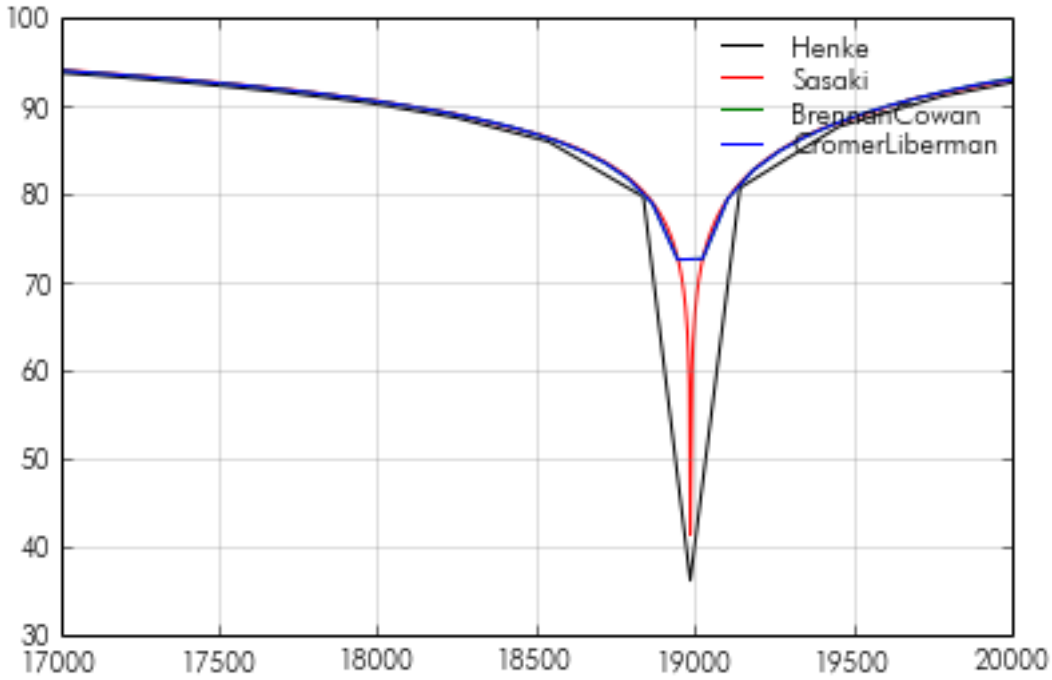
In [149]:
```
plot(Energy, cs.DAFS(Energy, (0,2,6), Temp=False))
plot(Energy, cs.DAFS(Energy, (0,2,6), Temp=True))
```

Out [149]:
```
[<matplotlib.lines.Line2D at 0x7f83d0e16ed0>]
```



In [169]:
```
for table in ["Henke", "Sasaki", "BrennanCowan", "CromerLiberman"]:
    del cs._ftab
    plot(Energy, cs.DAFS(Energy, (0,0,6), table=table), label=table)
legend()
```

Out [169]:
```
<matplotlib.legend.Legend at 0x7f83d2445e50>
```


100
90
80
70
60
50
40
30
17000
17500
18000
18500
19000
19500
20000
Henke
Sasaki
BrennanCowan
CromerLiberman

In [155]:

In []: