PyNeb_manual_2

April 20, 2017

0.1 Using Atom to compute populations and emissivities

The **Atom** class is equipped with method which give access to the populations, emissivities, and other atomic quantities:

```
In [1]: import pyneb as pn
        O3 = pn.Atom('O', 3)
        N2 = pn.Atom('N', 2)
        S2 = pn.Atom('S', 2)

In [2]: O3.getEnergy(4, unit='eV')

Out[2]: 2.5135650188001009

In [3]: O3.getStatWeight(level=4)

Out[3]: 5.0

In [4]: O3.getPopulations(tem=1.0e4, den=1e2)

Out[4]: array([  7.81643685e-01,   1.93870586e-01,   2.44814197e-02,
                 4.30930821e-06,   2.97166313e-10])

In [5]: O3.getPopulations(tem=1.5e4, den=1e2)

Out[5]: array([  8.02454816e-01,   1.75280839e-01,   2.22544005e-02,
                 9.94241529e-06,   2.05075367e-09])

In [6]: O3.getPopulations(tem=1e4, den=1e2).sum()

Out[6]: 1.0

In [7]: print('Critical densities')
        print('N2' + ' '.join('{:8.2e}'.format(cd) for cd in N2.getCritDensity(tem=12000)))
        print('O3' + ' '.join('{:8.2e}'.format(cd) for cd in O3.getCritDensity(tem=12000)))

Critical densities
N20.00e+00 4.06e+01 2.47e+02 9.55e+04 1.74e+07 9.83e+09
O30.00e+00 5.31e+02 3.70e+03 7.31e+05 2.61e+07

In [8]: O3.getEmissivity(tem=1e4, den=1e2, wave=5007) # in erg.s-1.cm3

Out[8]: array(3.4970737098460715e-21)
```

Wavelengths should be entered with enough precision to avoid confusion with nearby lines. Use **printTransition** to check if the transition selected by the code actually correponds to the one you intended:

```
In [9]: O3.printTransition(5007)
```

```
Input wave: 5007.0
Closest wave found: 5006.8
Relative error: 3E-05
Transition: 4 -> 3
```

getTransition is an abridged version which only returns the tuple of levels rather than an extended output, and is therefore apt to be used in scripts.

```
In [10]: O3.getTransition(5007)
```

```
Out[10]: (4, 3)
```

Or you can explicitly use the levels corresponding to the transition:

```
In [11]: O3.getEmissivity(tem=1e4, den=1e2, lev_i=4, lev_j=3)
```

```
Out[11]: array(3.4970737098460715e-21)
```

In the case of **getEmissivity**, tem and den can be arrays. In such a case, if they have different dimensions N and M, the function will return an array of NxM emissivities corresponding to all tem-den combinations; if both arrays have the same dimension, you can obtain the emissivities of either the NxN array of tem-den combinations as in the previous case, or of the 1D, N-length array obtained pairing tem and den element by element. This is controlled by the "product" parameter, the default being True (results is NxN matrix):

```
In [12]: O3.getEmissivity([10000, 12000], [100, 500], 4, 2, product=True)
```

```
Out[12]: array([[  1.17195376e-21,   1.18350278e-21],
                [  1.79323749e-21,   1.80805851e-21]])
```

```
In [13]: O3.getEmissivity([10000, 12000], [100, 500], 4, 2, product=False)
```

```
Out[13]: array([  1.17195376e-21,   1.80805851e-21])
```

0.2 Physical conditions determined from line ratios

The **Atom** object also contains a method to compute the electron temperature or density given a line ratio:

```
In [14]: O3.getTemDen(int_ratio=150., den=100., wave1=5007, wave2=4363)
```

```
Out[14]: 10070.108613008601
```

The keyword tem (or den) specifies the supplied value of the temperature (or density). Which quantity is computed (temperature or density) is determined by which quantity is provided to the method: if "den" is given, then "tem" is computed, and vice versa.

If the intensity ratio is a simple ratio of two transitions, you can:

- either give the wavelengths of the two transitions involved:

```
In [15]: O3.getTemDen(0.02, den=1.e4,  wave1=4363, wave2=5007)
```

```
Out[15]: 14923.313659752961
```

- or give the four levels that define the two transitions, in the following order: (upper level of numerator) (lower level of numerator) (upper level of denominator) (lower level of denominator); e. g.:

```
In [16]: O3.getTemDen(0.02, den=1.e4, lev_i1=5, lev_j1=4, lev_i2=4, lev_j2=3)
```

```
Out[16]: 14923.313659752961
```

In the general case of an intensity ratio formed by any number of transitions, an algebraic expression must be supplied as the argument of the keyword **to_eval**:

```
In [17]: O3.getTemDen(0.02, den=1.e4, to_eval="I(5, 4) / (I(4, 3) + I(4, 2))" )

Out[17]: 17201.456195718863

In [18]: N2.getTemDen(150., den=100., to_eval = "(L(6584) + L(6548)) / L(5755)")

Out[18]: 8303.6441184966006
```

The **to_eval** argument accepts either I(i, j) or L(wavelength) to identify the transitions involved in the diagnostic. Both can be mixed in the same string. If you do not know what transition corresponds to a given wavelength, use **printTransition** to find it.

The parameters **tem** and **den**, as well as the line ratio, may be arrays (1D or 2D, as in the case of observations obtained from IFUs), in which case the result will have the same shape. Some restrictions can be set to the domain explored by the method when looking for the solution; see the method's documentation for further details.

```
In [19]: O3.getTemDen([0.015, 0.019], den=[1.e4, 1.1e4], to_eval="I(5, 4) / (I(4, 3) + I(4, 2))")

Out[19]: array([ 14936.84544261,  16724.87927208])

In [20]: O3.getTemDen([0.015, 0.019], den=1.e4, to_eval="I(5, 4) / (I(4, 3) + I(4, 2))")

Out[20]: array([ 14936.84544261,  16755.22374476])
```

Notice that if you want to simultaneously determine both temperature and density combining two diagnostics (from two different atoms), you need to use the **getCrossTemDen** method of the **Diagnostic** class

Example of use in the case of density determination:

```
In [21]: S2.getTemDen(1.1, tem = 1e4, wave1=6730, wave2 = 6716)

Out[21]: 710.31244523435123
```

0.3 Ionic abundance determination

The ionic abundance is obtained from the intensity of a line normalized to Hbeta=100.

```
In [22]: O3.getIonAbundance(int_ratio=127, tem=1.5e4, den=100., wave=5007)

Out[22]: 1.3536707442595606e-05

In [23]: S2.getIonAbundance(int_ratio=72, tem=1.5e4, den=100., to_eval='L(6716)+L(6731)')

Out[23]: 7.5706366241087909e-07
```

The ionic abundance from recombination lines is treated below

0.4 Creating a dictionary of Atom objects

You can define all the atoms at once and put them in a dictionary by creating each atom at a time through the commands:

```
In [24]: O3 = pn.Atom('O', '3')
         O2 = pn.Atom('O', '2')
         N2 = pn.Atom('N', '2')
```

or rather use one of the following shortcuts:

```
In [25]: atoms = pn.getAtomDict() # a method always requires parenthesis, even without argument
```

```
warng _ManageAtomicData: rec data not available for Al2
warng _ManageAtomicData: rec data not available for Ar2
warng _ManageAtomicData: rec data not available for Ar3
warng _ManageAtomicData: rec data not available for Ar4
warng _ManageAtomicData: rec data not available for Ar5
warng _ManageAtomicData: rec data not available for Ba2
warng _ManageAtomicData: rec data not available for Ba4
warng _ManageAtomicData: rec data not available for Ca5
warng _ManageAtomicData: rec data not available for Cl2
warng _ManageAtomicData: rec data not available for Cl3
warng _ManageAtomicData: rec data not available for Cl4
warng _ManageAtomicData: rec data not available for Fe3
warng _ManageAtomicData: rec data not available for K4
warng _ManageAtomicData: rec data not available for K5
warng _ManageAtomicData: rec data not available for Mg5
warng _ManageAtomicData: rec data not available for Mg7
warng _ManageAtomicData: rec data not available for Na4
warng _ManageAtomicData: rec data not available for Na6
warng _ManageAtomicData: rec data not available for Ne2
warng _ManageAtomicData: rec data not available for Ne3
warng _ManageAtomicData: rec data not available for Ne4
warng _ManageAtomicData: rec data not available for Ne5
warng _ManageAtomicData: rec data not available for Ne6
warng _ManageAtomicData: rec data not available for Ni3
warng _ManageAtomicData: rec data not available for S2
warng _ManageAtomicData: rec data not available for S3
warng _ManageAtomicData: rec data not available for S4
warng _ManageAtomicData: rec data not available for Si2
warng _ManageAtomicData: rec data not available for Si3
warng _ManageAtomicData: rec data not available for Xe3
warng _ManageAtomicData: rec data not available for Xe4
warng _ManageAtomicData: rec data not available for Xe6
warng _ManageAtomicData: rec data not available for Kr3
warng _ManageAtomicData: rec data not available for Kr4
warng _ManageAtomicData: rec data not available for Kr5
warng _ManageAtomicData: rec data not available for Se3
warng _ManageAtomicData: rec data not available for Se4
warng _ManageAtomicData: rec data not available for Br3
warng _ManageAtomicData: rec data not available for Br4
warng _ManageAtomicData: rec data not available for Rb4
warng _ManageAtomicData: rec data not available for Rb5
warng _ManageAtomicData: rec data not available for Rb6
warng _ManageAtomicData: rec data not available for Fe4
```

```
warng _ManageAtomicData: rec data not available for Fe5
warng _ManageAtomicData: rec data not available for Fe6
warng _ManageAtomicData: rec data not available for Fe7
warng _ManageAtomicData: rec data not available for Fe2
warng _ManageAtomicData: rec data not available for P2
```

```
In [26]: atoms # All the available atoms
```

```
Out[26]: {'3He2': Atom 3He2 from 3he_ii_atom_cloudy.dat and 3he_ii_coll_cloudy.dat,
          'Al2': Atom Al2 from al_ii_atom_JSP86-HK87-VVF96-KS86.dat and al_ii_coll_KHAF92-TBK85-TBK84.dat
          'Ar2': Atom Ar2 from ar_ii_atom_Bal06.dat and ar_ii_coll_PB95.dat,
          'Ar3': Atom Ar3 from ar_iii_atom_MB09.dat and ar_iii_coll_MB09.dat,
          'Ar4': Atom Ar4 from ar_iv_atom_MZ82.dat and ar_iv_coll_RB97.dat,
          'Ar5': Atom Ar5 from ar_v_atom_LL93-MZ82-KS86.dat and ar_v_coll_GMZ95.dat,
          'Ba2': Atom Ba2 from ba_ii_atom_C04.dat and ba_ii_coll_SB98.dat,
          'Ba4': Atom Ba4 from ba_iv_atom_BHQZ95.dat and ba_iv_coll_SB98.dat,
          'Br3': Atom Br3 from br_iii_atom_BH86.dat and br_iii_coll_S97.dat,
          'Br4': Atom Br4 from br_iv_atom_BH86.dat and br_iv_coll_S97.dat,
          'C1': Atom C1 from c_i_atom_FFS85.dat and c_i_coll_JBK87-PA76.dat,
          'C1r': Atom C1 from c_i_rec_P91.func,
          'C2': Atom C2 from c_ii_atom_GMZ98.dat and c_ii_coll_BP92.dat,
          'C2r': Atom C2 from c_ii_rec_P91.func,
          'C3': Atom C3 from c_iii_atom_G83-NS78-WFD96.dat and c_iii_coll_Bal85.dat,
          'C3r': Atom C3 from c_iii_rec_P91.func,
          'C4': Atom C4 from c_iv_atom_WFD96.dat and c_iv_coll_AK04.dat,
          'C4r': Atom C4 from c_iv_rec_P91.func,
          'Ca5': Atom Ca5 from ca_v_atom_M83-KS86.dat and ca_v_coll_GMZ95.dat,
          'Cl2': Atom Cl2 from cl_ii_atom_MZ83.dat and cl_ii_coll_T04.dat,
          'Cl3': Atom Cl3 from cl_iii_atom_M83-KS86.dat and cl_iii_coll_BZ89.dat,
          'Cl4': Atom Cl4 from cl_iv_atom_KS86-MZ82-EM84.dat and cl_iv_coll_GMZ95.dat,
          'Fe2': Atom Fe2 from fe_ii_atom_B15.dat and fe_ii_coll_B15.dat,
          'Fe3': Atom Fe3 from fe_iii_atom_Q96_J00.dat and fe_iii_coll_Z96.dat,
          'Fe4': Atom Fe4 from fe_iv_atom_FFRR08.dat and fe_iv_coll_ZP97.dat,
          'Fe5': Atom Fe5 from fe_v_atom_Nal00.dat and fe_v_coll_BGMcL07.dat,
          'Fe6': Atom Fe6 from fe_vi_atom_CP00.dat and fe_vi_coll_CP99.dat,
          'Fe7': Atom Fe7 from fe_vii_atom_WB08.dat and fe_vii_coll_WB08.dat,
          'K4': Atom K4 from k_iv_atom_M83-KS86.dat and k_iv_coll_GMZ95.dat,
          'K5': Atom K5 from k_v_atom_M83-KS86.dat and k_v_coll_BZL88.dat,
          'Kr3': Atom Kr3 from kr_iii_atom_BH86.dat and kr_iii_coll_S97.dat,
          'Kr4': Atom Kr4 from kr_iv_atom_BH86.dat and kr_iv_coll_S97.dat,
          'Kr5': Atom Kr5 from kr_v_atom_BH86.dat and kr_v_coll_S97.dat,
          'Mg5': Atom Mg5 from mg_v_atom_GMZ97.dat and mg_v_coll_BZ94.dat,
          'Mg7': Atom Mg7 from mg_vii_atom_GMZ97.dat and mg_vii_coll_LB94-U.dat,
          'N1': Atom N1 from n_i_atom_KS86-WFD96.dat and n_i_coll_PA76-DMR76.dat,
          'N1r': Atom N1 from n_i_rec_P91.func,
          'N2': Atom N2 from n_ii_atom_FFT04.dat and n_ii_coll_T11.dat,
          'N2r': Atom N2 from n_ii_rec_P91.func,
          'N3': Atom N3 from n_iii_atom_GMZ98.dat and n_iii_coll_BP92.dat,
          'N3r': Atom N3 from n_iii_rec_P91.func,
          'N4': Atom N4 from n_iv_atom_WFD96.dat and n_iv_coll_RBHB94.dat,
          'N4r': Atom N4 from n_iv_rec_P91.func,
          'Na4': Atom Na4 from na_iv_atom_GMZ97.dat and na_iv_coll_BZ94.dat,
          'Na6': Atom Na6 from na_vi_atom_GMZ97.dat and na_vi_coll_LB94.dat,
          'Ne2': Atom Ne2 from ne_ii_atom_Bal06.dat and ne_ii_coll_GMB01.dat,
          'Ne3': Atom Ne3 from ne_iii_atom_GMZ97.dat and ne_iii_coll_McLB00.dat,
```

```
'Ne4': Atom Ne4 from ne_iv_atom_BBZ89-BK88.dat and ne_iv_coll_G81.dat,
'Ne5': Atom Ne5 from ne_v_atom_GMZ97-U-BD93.dat and ne_v_coll_DPNP13.dat,
'Ne6': Atom Ne6 from ne_vi_atom_GMZ98.dat and ne_vi_coll_ZGP94.dat,
'Ni3': Atom Ni3 from ni_iii_atom_B01.dat and ni_iii_coll_B01.dat,
'O1': Atom O1 from o_i_atom_WFD96.dat and o_i_coll_BK95.dat,
'O1r': Atom O1 from o_i_rec_P91.func,
'O2': Atom O2 from o_ii_atom_FFT04.dat and o_ii_coll_Kal09.dat,
'O2r': Atom O2 from o_ii_rec_SSB17-B-opt.hdf5,
'O3': Atom O3 from o_iii_atom_FFT04-SZ00.dat and o_iii_coll_SSB14.dat,
'O3r': Atom O3 from o_iii_rec_P91.func,
'O4': Atom O4 from o_iv_atom_GMZ98.dat and o_iv_coll_BP92.dat,
'O4r': Atom O4 from o_iv_rec_P91.func,
'O5': Atom O5 from o_v_atom_H80-NS79.dat and o_v_coll_BBDK85.dat,
'O5r': Atom O5 from o_v_rec_P91.func,
'P2': Atom P2 from p_ii_atom_MZ82.dat and p_ii_coll_T04.dat,
'Rb4': Atom Rb4 from rb_iv_atom_BH86.dat and rb_iv_coll_S97.dat,
'Rb5': Atom Rb5 from rb_v_atom_BH86.dat and rb_v_coll_S97.dat,
'Rb6': Atom Rb6 from rb_vi_atom_BH86.dat and rb_vi_coll_S97.dat,
'S2': Atom S2 from s_ii_atom_PKW09.dat and s_ii_coll_TZ10.dat,
'S3': Atom S3 from s_iii_atom_PKW09.dat and s_iii_coll_TG99.dat,
'S4': Atom S4 from s_iv_atom_JKD86-DHKD82.dat and s_iv_coll_DHKD82.dat,
'Se3': Atom Se3 from se_iii_atom_BH86.dat and se_iii_coll_S97.dat,
'Se4': Atom Se4 from se_iv_atom_B05.dat and se_iv_coll_B05.dat,
'Si2': Atom Si2 from si_ii_atom_BL93-CSB93-N77.dat and si_ii_coll_DK91.dat,
'Si3': Atom Si3 from si_iii_atom_M83-OKH88-FW90-KS86.dat and si_iii_coll_DK94.dat,
'Xe3': Atom Xe3 from xe_iii_atom_BHQZ95.dat and xe_iii_coll_SB98.dat,
'Xe4': Atom Xe4 from xe_iv_atom_BHQZ95.dat and xe_iv_coll_SB98.dat,
'Xe6': Atom Xe6 from xe_vi_atom_BHQZ95.dat and xe_vi_coll_SB98.dat}
```

It is also possible to select only a subset of the elements or ions available by specifying the arguments **elem_list** or **atom_list**:

```
In [27]: print(len(atoms))
```

```
75
```

```
In [28]: atoms = pn.getAtomDict(elem_list=['C', 'N', 'O']) # all the ions with spectra from 1 to 6 are
```

```
warng _ManageAtomicData: data for C5 not available
warng _ManageAtomicData: data for C5 not available
warng _ManageAtomicData: data for C5 not available
warng _ManageAtomicData: data for C6 not available
warng _ManageAtomicData: data for C6 not available
warng _ManageAtomicData: data for C6 not available
warng _ManageAtomicData: data for C7 not available
warng _ManageAtomicData: data for C7 not available
warng _ManageAtomicData: data for C7 not available
warng _ManageAtomicData: atom data not available for N5
warng _ManageAtomicData: coll data not available for N5
warng _ManageAtomicData: data for N6 not available
warng _ManageAtomicData: data for N6 not available
warng _ManageAtomicData: data for N6 not available
warng _ManageAtomicData: data for N7 not available
warng _ManageAtomicData: data for N7 not available
warng _ManageAtomicData: data for N7 not available
```

```
warng _ManageAtomicData: atom data not available for O6
warng _ManageAtomicData: coll data not available for O6
warng _ManageAtomicData: data for O7 not available
warng _ManageAtomicData: data for O7 not available
warng _ManageAtomicData: data for O7 not available
```

```
In [29]: atoms # All the CNO available atoms
```

```
Out[29]: {'C1': Atom C1 from c_i_atom_FFS85.dat and c_i_coll_JBK87-PA76.dat,
          'C1r': Atom C1 from c_i_rec_P91.func,
          'C2': Atom C2 from c_ii_atom_GMZ98.dat and c_ii_coll_BP92.dat,
          'C2r': Atom C2 from c_ii_rec_P91.func,
          'C3': Atom C3 from c_iii_atom_G83-NS78-WFD96.dat and c_iii_coll_Bal85.dat,
          'C3r': Atom C3 from c_iii_rec_P91.func,
          'C4': Atom C4 from c_iv_atom_WFD96.dat and c_iv_coll_AK04.dat,
          'C4r': Atom C4 from c_iv_rec_P91.func,
          'N1': Atom N1 from n_i_atom_KS86-WFD96.dat and n_i_coll_PA76-DMR76.dat,
          'N1r': Atom N1 from n_i_rec_P91.func,
          'N2': Atom N2 from n_ii_atom_FFT04.dat and n_ii_coll_T11.dat,
          'N2r': Atom N2 from n_ii_rec_P91.func,
          'N3': Atom N3 from n_iii_atom_GMZ98.dat and n_iii_coll_BP92.dat,
          'N3r': Atom N3 from n_iii_rec_P91.func,
          'N4': Atom N4 from n_iv_atom_WFD96.dat and n_iv_coll_RBHB94.dat,
          'N4r': Atom N4 from n_iv_rec_P91.func,
          'N5r': Atom N5 from n_v_rec_P91.func,
          'O1': Atom O1 from o_i_atom_WFD96.dat and o_i_coll_BK95.dat,
          'O1r': Atom O1 from o_i_rec_P91.func,
          'O2': Atom O2 from o_ii_atom_FFT04.dat and o_ii_coll_Kal09.dat,
          'O2r': Atom O2 from o_ii_rec_SSB17-B-opt.hdf5,
          'O3': Atom O3 from o_iii_atom_FFT04-SZ00.dat and o_iii_coll_SSB14.dat,
          'O3r': Atom O3 from o_iii_rec_P91.func,
          'O4': Atom O4 from o_iv_atom_GMZ98.dat and o_iv_coll_BP92.dat,
          'O4r': Atom O4 from o_iv_rec_P91.func,
          'O5': Atom O5 from o_v_atom_H80-NS79.dat and o_v_coll_BBDK85.dat,
          'O5r': Atom O5 from o_v_rec_P91.func,
          'O6r': Atom O6 from o_vi_rec_P91.func}
```

```
In [30]: atoms = pn.getAtomDict(atom_list=['O2', 'O3', 'Ar3', 'N2'])
```

```
warng _ManageAtomicData: rec data not available for Ar3
```

```
In [31]: atoms
```

```
Out[31]: {'Ar3': Atom Ar3 from ar_iii_atom_MB09.dat and ar_iii_coll_MB09.dat,
          'N2': Atom N2 from n_ii_atom_FFT04.dat and n_ii_coll_T11.dat,
          'N2r': Atom N2 from n_ii_rec_P91.func,
          'O2': Atom O2 from o_ii_atom_FFT04.dat and o_ii_coll_Kal09.dat,
          'O2r': Atom O2 from o_ii_rec_SSB17-B-opt.hdf5,
          'O3': Atom O3 from o_iii_atom_FFT04-SZ00.dat and o_iii_coll_SSB14.dat,
          'O3r': Atom O3 from o_iii_rec_P91.func}
```

In all these cases, a dictionary is created whose keys are the conventional atom names <element><spectrum>, and the corresponding entries the atoms themselves; e. g.:

```
In [32]: atoms['N2']
```

Out[32]: Atom N2 from n_ii_atom_FFT04.dat and n_ii_coll_T11.dat

```python
In [33]: atoms['N2'].getEmissivity(tem=1e4, den=1e2, wave=6584)   # example of use
```

Out[33]: array(5.9490128036162234e-21)

This can be useful if you need to loop on a list of atoms, to plot atomic data for example. To see what atoms have been created (which is limited by the data included in the selected atomic data set), enter:

```python
In [34]: atoms.keys()
```

Out[34]: dict_keys(['O2', 'O2r', 'O3', 'O3r', 'Ar3', 'N2', 'N2r'])

If you want to be able to access them directly rather than through a dictionary, input from the command line:

```python
In [35]: for key in atoms.keys():
             vars()[key]=atoms[key]
```

and then you will be able to do the following:

```python
In [36]: Ar3.NLevels
```

Out[36]: 5