PyNeb_manual_4

April 18, 2017

0.1 Recombination lines: the RecAtom class

This object is similar to **Atom**, but some methods differ due to the particularities of the recombination spectrum. It is instantiated with the following command:

```
In [1]: %matplotlib inline
        %config InlineBackend.figure_format = 'svg'
        import numpy as np
        import matplotlib.pyplot as plt
        import pyneb as pn
```

```
In [2]: H1 = pn.RecAtom('H', 1)
```

The data are read from a fits file and interpolated in temperature and density. The name of the file is retrievable through:

```
In [3]: pn.atomicData.getDataFile('H1', 'rec')
```

```
Out[3]: 'h_i_rec_SH95.hdf5'
```

In case the instantiation takes too much time (on some systems, the reading the fits fils is slow), you can use the data store in the hdf5 file using:

```
In [4]: pn.atomicData.setDataFile('h_i_rec_SH95.hdf5')
```

```
In [5]: H1 = pn.RecAtom('H', 1)
```

The ions for which recombination lines are available are listed with the command:

```
In [6]: pn.atomicData.getAllAtoms(coll=False, rec=True)
```

```
Out[6]: ['H1',
         'He1',
         'He2',
         'C1',
         'C2',
         'C3',
         'C4',
         'N1',
         'N2',
         'N3',
         'N4',
         'O1',
         'O2',
         'O3',
         'O4',
         'O5',
         'N5',
         'O6']
```

You can list all the available data files for a given ion by:

```
In [7]: pn.atomicData.getAllAvailableFiles('H1')
```

```
Out[7]: ['h_i_atom.chianti',
         'h_i_coll.chianti',
         'h_i_rec_SH95-caseA.fits',
         'h_i_rec_SH95-caseA.hdf5',
         'h_i_rec_SH95.fits',
         'h_i_rec_SH95.hdf5',
         'h_i_trc_SH95-caseA.dat',
         'h_i_trc_SH95-caseB.dat']
```

The emissivities are obtained by, e.g.:

```
In [8]: Hbeta = H1.getEmissivity(tem=1e4, den=1e3, lev_i=4, lev_j=2)
        Halpha = H1.getEmissivity(tem=1e4, den=1e3, lev_i=3, lev_j=2)
        print(Halpha/Hbeta)
```

```
2.85691188359
```

The argument used to identify the various lines in the data file depend on whether the atom is a hydrogenoid or not. In the first case, the transition can be specified either as a pair of levels **lev_i**, **lev_j**, or as a label (labels are descriptive of level pairs):

```
In [9]: print(H1.getEmissivity([1e4, 1.8e4], [1e4, 1e2], lev_i = 4, lev_j = 2))
```

```
[[  1.24000000e-25   1.23500000e-25]
 [  7.40180000e-26   7.38740000e-26]]
```

```
In [10]: print(H1.getEmissivity([1e4, 1.8e4], [1e4, 1e2], label='4_2'))
```

```
[[  1.24000000e-25   1.23500000e-25]
 [  7.40180000e-26   7.38740000e-26]]
```

As in the case of collisional lines, setting **product=False** modifies the input array of temperature and density values:

```
In [11]: print(H1.getEmissivity([1e4, 1.8e4], [1e4, 1e2], label='4_2', product=False))
```

```
[  1.24000000e-25   7.38740000e-26]
```

In the case of non-hydrogenoids, the transition can be specified either as a wavelength or as a label (labels are descriptive of wavelengths):

```
In [12]: He1 = pn.RecAtom('He', 1)
         print(He1.getEmissivity(1e4, 1e2, wave=4471.0))
         print(He1.getEmissivity(1e4, 1e2, label='4471.0'))
```

```
6.11561306801098e-26
6.11561306801098e-26
```

Use **Atom.labels** to display the valid labels for both kind of recombiation atoms, i.e.:

```
In [13]: H1.labels[0:10]
```

```
Out[13]: ('2_1', '3_1', '3_2', '4_1', '4_2', '4_3', '5_1', '5_2', '5_3', '5_4')
```

```
In [14]: He1.labels[0:10]
```

```
Out[14]: ('2945.0',
          '3188.0',
          '3614.0',
          '3889.0',
          '3965.0',
          '4026.0',
          '4121.0',
          '4388.0',
          '4438.0',
          '4471.0')
```

As an example application, you can easily generate a 2D table of H alpha/ H Beta as a function of Te and Ne:

```
In [15]: tem = np.linspace(5000, 20000, 100)
         den = np.logspace(2, 6, 50)
         im_Hab = (H1.getEmissivity(tem, den, label='3_2') / H1.getEmissivity(tem, den, label='4_2'))
         print(im_Hab.shape)
         print(np.min(im_Hab))
         print(np.max(im_Hab))
```

```
(100, 50)
2.72452373753
3.04092769441
```

By default, H1 emissivities are computed under case B; to compute Case A emissivities, the atomic data file must be changed:

```
In [16]: pn.atomicData.setDataFile('h_i_rec_SH95-caseA.hdf5')
```

```
In [17]: H1_A = pn.RecAtom('H', 1)
```

Note that the previously defined H1 atom is still available and use the default values.

```
In [18]: im_caseAoverB = (H1_A.getEmissivity(tem, den, label='4_2')/H1.getEmissivity(tem, den, label='4_
```

plt.imshow(im_caseAoverB) plt.colorbar();

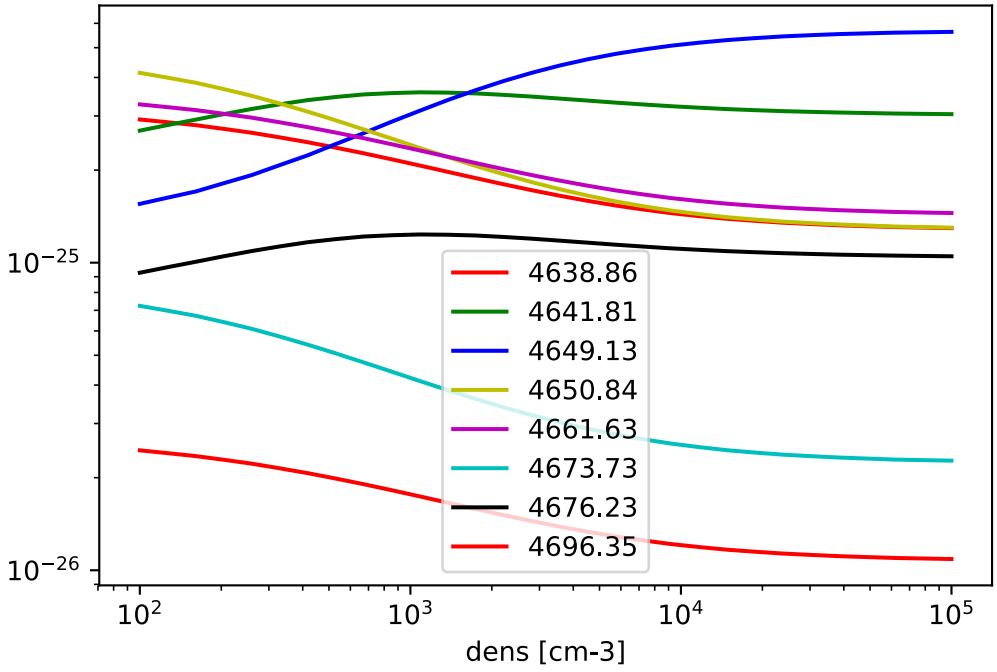
0.1.1 Emissivities of OII recombination lines

```
In [20]: O2 = pn.RecAtom('O', 2)
```

```
In [29]: V1_mult = ('4638.86', '4641.81', '4649.13', '4650.84', '4661.63', '4673.73', '4676.23', '4696.3
```

```
In [31]: cols = 'rgbymckrgbymckr'
         temps = 1e4
         dens = np.logspace(2, 5, 30)
         f, ax = plt.subplots()
         for j, label in enumerate(V1_mult):
             ax.loglog(dens, O2.getEmissivity(temps, dens, label=label), label=label, c=cols[j])
         ax.set_xlabel('dens [cm-3]')
         ax.legend(loc='best')
```

```
Out[31]: <matplotlib.legend.Legend at 0x7fbd49d23a20>
```



0.1.2 Ionic abundances from collisional and recombination lines, ADF

```
In [45]: O3c = pn.Atom('O',3)
         O2r = pn.RecAtom('O',2)
         tem = 1e4
         den = 1e2
         Opp_col = O3c.getIonAbundance(int_ratio=127, tem=tem, den=den, wave=5007)
         Opp_rec = O2r.getIonAbundance(int_ratio=.023, tem=tem, den=den, label='4649.13')
         print("""12+log(O++/H) from recombination line: {:.2f},
         12+log(O++/H) from collisional line: {:.2f},
         ADF: {:.1f}""".format(12+np.log10(Opp_rec), 12+np.log10(Opp_col), Opp_rec / Opp_col))
```

```
12+log(O++/H) from recombination line: 8.09,
12+log(O++/H) from collisional line: 7.48,
ADF: 4.1
```

```
In [ ]:
```