PyNeb_manual_6

October 15, 2017

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import pyneb as pn
```

0.1 The EmissionLine class

This is the class characterizing emission lines. An emission line is identified by an element and a spectrum (which identify the emitting ion), a wavelength in Angstrom, a blend flag, a label in the standard PyNeb format, an observed intensity, a reddening-corrected intensity, an expression describing how the intensity depends on the included wavelengths, an observational error and an error on the corrected intensity. Other programs determine one or more of these values.

To instantiate an Emission Line object, use the following:

```
In [2]: line = pn.EmissionLine('O', 3, 5007, obsIntens=[1.4, 1.3])
        line2 = pn.EmissionLine(label = 'O3_5007A', obsIntens=320, corrected=True)
```

obsIntens is a value, a list or a numpy array of values corresponding to the observed intensity of the given emission line.

```
In [3]: print(line)
```

```
Line O3 O3_5007A
```

To know how the label of a given line is exactly spelled, you can print the dictionary pn.LINE_LABEL_LIST

```
In [4]: print(pn.LINE_LABEL_LIST['O3'])
```

```
['4931A', '4959A', '5007A', '2315A', '2321A', '2331A', '4363A', '1658A', '1661A', '
```

It is possible to instantiate a line not contained in the pn.LINE_LABEL_LIST. In this case a warning is issued, but the code doesn't stop.

The observed intensity is stored in **line.obsIntens** and the extinction-corrected intensity is stored in **line.corrIntens**. **line.corrIntens** is set to 0.0 when the line is instantiated, unless the parameter corrected is set to **True**, in which case the observed value **obsIntens** is copied to the **corrIntens** slot (the same applies for **corrError**, which is set to **obsError**).

The **corrIntens** value can also be computed using an instantiation of the **pn.RedCorr** class:

```
In [5]: redcorr = pn.RedCorr(E_BV = 0.87, law = 'F99')
```

```
In [6]: line.correctIntens(redcorr) #redcorr is used to compute line.corrIntens
```

The line information is printed using:

```
In [7]: line.printLine()
```

```
Line O3 O3_5007A evaluated as L(5007)
Observed intensity: [ 1.4  1.3]
Observed error: [ 0.  0.]
Corrected intensity: [ 22.58352855  20.97041937]
Corrected error: [ 0.  0.]
```

Most of the times, users will not need to define or manipulate EmissionLine objects, since most of the work on the EmissionLine objects will be performed from the Observation class (read data, extinction correction); see next section.

WARNING: Note that the wavelengths assigned to EmissionLine objects are simply the numerical part of the label:

```
In [8]: Hb1 = pn.EmissionLine(label='H1r_4861A').wave
        print(Hb1)
```

```
4861.0
```

whereas pn.Atom computes them as the difference from energy levels:

```
In [9]: Hb2 = pn.RecAtom('H', 1).getWave(4, 2)
        print(Hb2)
```

```
4861.33165987
```

This can cause small errors when both methods are used simultaneously. For instance, the extinction correction at Hb1 is slightly different from the expected value of 1:

```
In [10]: rc = pn.RedCorr()
         rc.E_BV = 1.34
         rc.law = 'F99'
         rc.getCorrHb(Hb1)
```

```
Out[10]: 1.000410798530041
```

This happens because the ExtCorr uses the precise Hβ value computed from energy levels. If this roundoff error exceeds your tolerance, a possible workaround is forcing the emission line to have exactly the wavelength computed from the energy levels:

```
In [11]: O3w = pn.EmissionLine('O', 3, wave=5007).wave
         print(rc.getCorrHb(O3w))
```

```
0.836202049828
```

```
In [12]: Hb11 = pn.EmissionLine('H', 1, wave=Hb2).wave
         print(rc.getCorrHb(Hb11))
         # This will generate a warning (as the transition is not included in the
         # but the code won't stop.
```

```
warng EmissionLine: Atom H1 not valid
1.0
```

0.2 The Observation class: reading and dealing with observations

0.2.1 Reading observation from a file

pn.Observation is the class characterizing observation records. An observation record is composed of an object identifier, the observed intensity of one or more emission lines, and, optionally, the dereddened line intensities and the identifier of the extinction law used, and the value of c(Hbeta).

Observations can be initialized by reading data files, which can be organized with different emission lines either in rows or columns (usually, in a survey of many objects with few emission lines emission lines change across columns; and in a high-resolution observation of a small sample of objects lines change across rows).

The following is an example of how to define an observation:

```
In [13]: obs = pn.Observation()
```

```
In [14]: %%writefile observations1.dat
         LINE SMC_24
         S4_10.5m   7.00000
         Ne2_12.8m  8.3000
         Ne3_15.6m 34.10
         S3_18.7m  10.
         O2_3726A  39.700
         O2_3729A  18.600
         Ne3_3869A 18.90
         Ne3_3968A  6.4
         S2_4069A   0.85
         S2_4076A   0.450
         O3_4363A   4.36
         H1r_4861A 100.00
         O3_5007A 435.09
         N2_5755A   0.510000
         S3_6312A   0.76
         O1_6300A   1.69
         O1_6364A   0.54
         N2_6548A   6.840000
         H1r_6563A  345.00
         N2_6584A  19.00
         S2_6716A   1.220000
         S2_6731A   2.180000
         Ar3_7136A  4.91
         O2_7319A+   6.540000
         O2_7330A+   5.17

Overwriting observations1.dat
```

```
In [15]: obs.readData('observations1.dat', fileFormat='lines_in_rows', err_default=

In [16]: obs.printIntens(returnObs=True)
```

```
S4_10.5m      7.000
Ne2_12.8m     8.300
Ne3_15.6m    34.100
S3_18.7m     10.000
O2_3726A     39.700
O2_3729A     18.600
Ne3_3869A    18.900
Ne3_3968A     6.400
S2_4069A      0.850
S2_4076A      0.450
O3_4363A      4.360
H1r_4861A   100.000
O3_5007A    435.090
N2_5755A      0.510
S3_6312A      0.760
O1_6300A      1.690
O1_6364A      0.540
N2_6548A      6.840
H1r_6563A   345.000
N2_6584A     19.000
S2_6716A      1.220
S2_6731A      2.180
Ar3_7136A     4.910
O2_7319A+     6.540
O2_7330A+     5.170
```

```
In [17]: obs.extinction.law = 'CCM89'  # define the extinction law from Cardelli et
         obs.correctData()                       # the dereddened data are computed
```

The data can be read by the readData method as above or directly while instantiating the object:

```
In [18]: obs = pn.Observation('observations1.dat', fileFormat='lines_in_rows', corr
```

The format of the data file from which the emission line intensities are read can be one of three kinds: “lines_in_rows” as above, or “lines_in_cols” like this one:

```
In [19]: %%writefile observations2.dat
         NAME O2_3726A  O2_3726Ae O2_3729A O2_3729Ae
         NGC3132 0.93000   0.05000   0.17224200 0.10
         IC418 1.28000   0.05000   0.09920000 0.05
         M33 0.03100   0.080     0.03100    0.10

Overwriting observations2.dat
```

```
In [20]: obs2 = pn.Observation('observations2.dat', fileFormat='lines_in_cols', cor
```

or fileFormat=‘lines_in_rows_err_cols’ (errors labeled “err”. Don’t name an observation “err”!) like this one:

```
In [21]: %%writefile observations3.dat
         LINE     TT   err  TT2 err TT3 err
         cHbeta   1.2  0.0  1.5 0.2 1.1 0.2
         O3_5007A 1.5  0.15 1.3  .2 1.1 0.1
         H1_6563A 2.89 0.05 1.6 0.3 1.3 0.1
         N2_6584A 1.   0.20 0.3 0.5 1.5 0.1

Overwriting observations3.dat
```

```
In [22]: #obs3 = pn.Observation('observations3.dat', fileFormat='lines_in_rows_err_
```

The delimiter between the columns is any sequence of spaces or TAB, but it can be changed using the delimiter parameter. The line names are defined by a label starting with the name of the atom (‘O2’), followed by an underscore, followed by a wavelength and ending with a unit (‘A’ or ‘m’). The list of all the lines managed by PyNeb, ordered by atoms, is obtained by entering:

```
In [23]: for atom in pn.LINE_LABEL_LIST:
             print(atom, pn.LINE_LABEL_LIST[atom])

H1r ['1216A', '1026A', '973A', '6563A', '4861A', '4341A', '4102A', '3970A', '3889A'
He1r ['5876A', '2945A', '3188A', '3614A', '3889A', '3965A', '4026A', '4121A', '4388
He2r ['1640A', '1215A', '1084A', '4686A', '3203A', '6560A', '5411A', '4859A']
3He2 ['3.50c']
Al2 ['2674A', '2670A', '2661A', '1671A', '4451A', '4463A', '4488A', '164.2m', '54.1
Ar2 ['7.0m']
Ar3 ['7136A', '7751A', '8036A', '3005A', '3109A', '3154A', '5192A', '9.0m', '6.37m'
Ar4 ['4740A', '4711A', '2868A', '7263A', '7332A', '2854A', '7170A', '7237A', '77.4m
Ar5 ['6133A', '6435A', '7005A', '2637A', '2692A', '2786A', '4626A', '1218A', '1229A
Ba2 ['4935A', '6497A', '6854A', '4555A', '5854A', '6142A', '2361A', '2668A', '2726A
Ba4 ['5697A']
Br3 ['6646A', '6133A', '3714A', '8420A', '9419A', '3498A', '7385A', '8142A', '7.94m
```

```
C1 ['9808A', '9824A', '9850A', '4618A', '4621A', '4627A', '8728A', '2963A', '2965A'
C2 ['2325A', '2328A', '2323A', '2327A', '2322A', '2325A', '1335A', '1336A', '3131A'
C3 ['1910A', '1909A', '1907A', '977A', '2000A', '2001A', '2003A', '422.0m', '124.9m
C4 ['1551A', '1548A', '92.8m']
Ca2 ['7292A', '7324A']
Ca5 ['5309A', '6087A', '6428A', '2280A', '2413A', '2464A', '3998A', '4.16m', '3.05m
Cl2 ['8579A', '9124A', '9381A', '3586A', '3678A', '3719A', '6162A', '14.4m', '10.0m
Cl3 ['5538A', '5518A', '3353A', '8500A', '8548A', '3343A', '8434A', '8481A', '151.5
Cl4 ['7261A', '7531A', '8046A', '3071A', '3119A', '3204A', '5323A', '1463A', '1474A
Fe3 ['4009A', '4659A', '4668A', '4701A', '4734A', '4755A', '5011A', '5085A', '5270A
Fe4 ['4491A', '5685A', '5735A', '6740A']
Fe5 ['3783A', '3795A', '3822A', '3891A', '3895A', '3911A', '4071A', '4181A', '4227A
Fe6 ['3556A', '3929A', '5146A', '5176A', '5278A', '5335A', '5370A', '5424A', '5427A
Fe7 ['5159A', '5276A', '5721A', '6087A']
K4 ['6102A', '6796A', '7109A', '2594A', '2711A', '2760A', '4511A', '6.0m', '4.3m',
K5 ['4163A', '4123A', '2514A', '6349A', '6446A', '2495A', '6222A', '6316A', '42.2m'
K6 ['5602A', '6229A']
Kr3 ['6827A', '9902A', '3022A', '3504A', '3600A', '5423A', '2.2m', '1.88m', '13.1m'
Kr4 ['5868A', '5346A', '3219A', '7131A', '8091A', '2993A', '6108A', '6798A', '6.0m'
Kr5 ['5069A', '6256A', '8243A', '2550A', '2819A', '3163A', '5132A', '2.67m', '1.32m
Mg4 ['4.5m']
Mg5 ['2783A', '2929A', '2992A', '1294A', '1325A', '1338A', '2418A', '5.6m', '3.96m'
Mg7 ['2441A', '2509A', '2629A', '1174A', '1190A', '1216A', '2261A', '943A', '953A',
N1 ['5200A', '5198A', '3467A', '3466A']
N2 ['6527A', '6548A', '6584A', '3058A', '3063A', '3071A', '5755A', '2137A', '2139A'
N3 ['1749A', '1754A', '1747A', '1752A', '1744A', '1750A', '990A', '992A', '2280A',
N4 ['1488A', '1487A', '1483A', '765A', '1575A', '1576A', '1580A', '158.4m', '48.3m'
Na3 ['7.3m']
Na4 ['3242A', '3362A', '3416A', '1504A', '1529A', '1540A', '2804A', '9.0m', '6.34m'
Na6 ['2816A', '2872A', '2972A', '1343A', '1356A', '1378A', '2569A', '14.39m', '5.4m
Ne2 ['12.8m']
Ne3 ['3869A', '3968A', '4012A', '1794A', '1815A', '1824A', '3343A', '15.6m', '10.9m
Ne4 ['2425A', '2422A', '1602A', '4716A', '4726A', '1601A', '4714A', '4724A', '224.9
Ne5 ['3300A', '3346A', '3426A', '1565A', '1575A', '1592A', '2973A', '1132A', '1137A
Ne6 ['997A', '1010A', '993A', '1006A', '986A', '999A', '559A', '563A', '1271A', '12
Ni3 ['7890A', '8500A', '6000A', '6401A', '6534A', '6682A', '6797A', '7125A', '6946A
O1 ['6300A', '6364A', '6392A', '2959A', '2973A', '2979A', '5577A', '63.2m', '44.1m'
O2 ['3729A', '3726A', '2470A', '7319A', '7320A', '7330A', '7331A', '2470A', '834A',
O2r ['4638.86A', '4641.81A', '4649.13A', '4650.84A', '4661.63A', '4673.73A', '4676.
O3 ['4931A', '4959A', '5007A', '2315A', '2321A', '2331A', '4363A', '1658A', '1661A'
O4 ['1400A', '1407A', '1397A', '1405A', '1394A', '1401A', '788A', '1801A', '1806A',
O5 ['1220A', '1218A', '1214A', '630A', '1301A', '1303A', '1309A', '73.5m', '22.6m',
Rb4 ['5760A', '9009A', '9604A', '2603A', '3110A', '3178A', '4750A', '1.6m', '1.44m'
Rb5 ['5364A', '4742A', '2873A', '6188A', '7290A', '2609A', '5080A', '5800A', '4.1m'
Rb6 ['4210A', '5373A', '7220A', '2212A', '2495A', '2832A', '4660A', '1.95m', '1.01m
S2 ['6731A', '6716A', '4076A', '4069A', '1260A', '1549A', '1550A', '1823A', '1824A'
S3 ['8829A', '9069A', '9531A', '3681A', '3722A', '3798A', '6312A', '33.5m', '12.0m'
S4 ['1405A', '1424A', '1398A', '1417A', '1387A', '1406A', '10.5m', '29.0m', '11.2m'
```

```
Se3 ['7671A', '8854A', '3516A', '3746A', '4082A', '6493A', '5.74m', '2.54m', '4.55m
Se4 ['2.28m']
Si2 ['2335A', '2351A', '2329A', '2345A', '2320A', '1808A', '1817A', '8007A', '8077A
Si3 ['1897A', '1892A', '1883A', '1206A', '3315A', '3329A', '3359A', '77.7m', '25.7m
Xe3 ['5847A', '2769A', '3574A', '3800A', '5261A', '1.23m', '1.02m', '6.0m', '1.11m'
Xe4 ['7535A', '5709A', '3566A', '6769A', '9498A', '2804A', '4467A', '5511A', '2.36m
Xe6 ['6409A']
```

The presence of a trailing "e" at the end of the label points to the error associated to the line. The error is considered to be relative to the intensity (i.e., 0.05 means 5% of the intensity), unless the parameter errIsRelative is set to False. A common value for all the errors can be defined by the parameter **err_default** (0.10 is the default value).

0.2.2 Extinction correction in Observation class

Once the data have been read, they have to be corrected from extinction. An instantiation of **RedCorr()** is available inside the **Observation** object as **obs.extinction**.

If the data file contains **cHbeta** or **E(B-V)** alongside of line labels, the corresponding information on extinction is transmitted to the extinction correction object. Otherwise, the extinction parameters must be set manually; for example:

```
In [24]: obs = pn.Observation('observations1.dat', fileFormat='lines_in_rows', corr
         obs.extinction.cHbeta = 1.2
         obs.extinction.E_BV = 0.34
```

An extinction law has to be specified in either case:

```
In [25]: obs.extinction.law = 'F99'
```

To correct all the lines at once:

```
In [26]: obs.correctData()

In [27]: obs.printIntens(returnObs=True)
```

```
S4_10.5m       7.000
Ne2_12.8m      8.300
Ne3_15.6m     34.100
S3_18.7m      10.000
O2_3726A      39.700
O2_3729A      18.600
Ne3_3869A     18.900
Ne3_3968A      6.400
S2_4069A       0.850
S2_4076A       0.450
O3_4363A       4.360
H1r_4861A    100.000
O3_5007A     435.090
```

```
N2_5755A       0.510
S3_6312A       0.760
O1_6300A       1.690
O1_6364A       0.540
N2_6548A       6.840
H1r_6563A    345.000
N2_6584A      19.000
S2_6716A       1.220
S2_6731A       2.180
Ar3_7136A      4.910
O2_7319A+      6.540
O2_7330A+      5.170
```

```
In [28]: obs.printIntens()
```

```
S4_10.5m       7.120
Ne2_12.8m      8.415
Ne3_15.6m     34.483
S3_18.7m      10.093
O2_3726A     171.242
O2_3729A      80.156
Ne3_3869A     78.134
Ne3_3968A     25.717
S2_4069A       3.320
S2_4076A       1.754
O3_4363A      15.704
H1r_4861A    310.254
O3_5007A    1289.851
N2_5755A       1.244
S3_6312A       1.662
O1_6300A       3.704
O1_6364A       1.170
N2_6548A      14.346
H1r_6563A    721.747
N2_6584A      39.607
S2_6716A       2.488
S2_6731A       4.435
Ar3_7136A      9.384
O2_7319A+     12.180
O2_7330A+      9.614
```

If you want the corrected line intensities to be normalized to a given wavelength, use the following:

```
In [29]: obs.correctData(normWave=4861.)
```

The extinction correction can be determined by comparing the observed values to a theoretical ratio, as in the following:

```
In [30]: obs.printIntens()
```

```
S4_10.5m       2.295
Ne2_12.8m      2.712
Ne3_15.6m     11.115
S3_18.7m       3.253
O2_3726A      55.194
O2_3729A      25.836
Ne3_3869A     25.184
Ne3_3968A      8.289
S2_4069A       1.070
S2_4076A       0.565
O3_4363A       5.062
H1r_4861A    100.000
O3_5007A     415.740
N2_5755A       0.401
S3_6312A       0.536
O1_6300A       1.194
O1_6364A       0.377
N2_6548A       4.624
H1r_6563A    232.631
N2_6584A      12.766
S2_6716A       0.802
S2_6731A       1.429
Ar3_7136A      3.025
O2_7319A+      3.926
O2_7330A+      3.099
```

```
In [31]: obs.def_EBV(label1="H1r_6563A", label2="H1r_4861A", r_theo=2.85)
         print(obs.extinction.E_BV)
         obs.correctData(normWave=4861.)
```

```
[ 0.16483175]
```

```
In [32]: obs.printIntens()
```

```
S4_10.5m       4.076
Ne2_12.8m      4.826
Ne3_15.6m     19.803
S3_18.7m       5.802
O2_3726A      46.576
O2_3729A      21.812
Ne3_3869A     21.722
Ne3_3968A      7.255
S2_4069A       0.950
S2_4076A       0.503
O3_4363A       4.687
```

```
H1r_4861A   100.000
O3_5007A    425.599
N2_5755A      0.454
S3_6312A      0.641
O1_6300A      1.428
O1_6364A      0.454
N2_6548A      5.657
H1r_6563A   285.000
N2_6584A     15.668
S2_6716A      0.995
S2_6731A      1.777
Ar3_7136A     3.882
O2_7319A+     5.106
O2_7330A+     4.034
```

By default, this method prints out the corrected intensities. To print the observed intensities, use the **returnObs=True** parameter.

The method **getSortedLines** returns the lines sorted in alphabetical order according to either the emitting atoms (default) or the wavelength (using the **crit='wave'** parameter):

```
In [33]: for line in obs.getSortedLines():
             print(line.label, line.corrIntens[0])
```

```
Ar3_7136A 3.88214978929
H1r_4861A 100.0
H1r_6563A 285.0
N2_5755A 0.453846550278
N2_6548A 5.65746807125
N2_6584A 15.6684748528
Ne2_12.8m 4.82602971256
Ne3_15.6m 19.8027022262
Ne3_3869A 21.7218621627
Ne3_3968A 7.25498470993
O1_6300A 1.42792388246
O1_6364A 0.45369747694
O2_3726A 46.5764398853
O2_3729A 21.8120580007
O2_7319A+ 5.1064244648
O2_7330A+ 4.03377032255
O3_4363A 4.68705107403
O3_5007A 425.599154595
S2_4069A 0.95040451703
S2_4076A 0.502681231482
S2_6716A 0.995405412218
S2_6731A 1.77656378862
S3_18.7m 5.80184591117
S3_6312A 0.641461012117
```

```
S4_10.5m 4.07647872134
```

The following method, which gives the list of all the atoms implied in the observed emission lines, will be useful later:

```
In [34]: atomList = obs.getUniqueAtoms()

In [35]: atomList

Out[35]: array(['Ar3', 'H1r', 'N2', 'Ne2', 'Ne3', 'O1', 'O2', 'O3', 'S2', 'S3', 'S4
               dtype='<U3')
```

0.2.3 Adding observations and lines

Once an **Observation** object is instantiated, you can add a new observation (corresponding, e.g., to a new object or a new fiber) by using:

```
In [36]: obs.addObs('test', np.random.rand(25))
```

where 'test' is the name of the new observation. The new observation must have the same size of **obs**, that is, it must contain **obs.n_lines** lines.

```
In [37]: obs.printIntens()

S4_10.5m       4.076    0.192
Ne2_12.8m      4.826    0.023
Ne3_15.6m     19.803    0.427
S3_18.7m       5.802    0.337
O2_3726A      46.576    0.548
O2_3729A      21.812    0.576
Ne3_3869A     21.722    0.975
Ne3_3968A      7.255    0.532
S2_4069A       0.950    0.235
S2_4076A       0.503    0.971
O3_4363A       4.687    0.519
H1r_4861A    100.000    0.709
O3_5007A     425.599    0.616
N2_5755A       0.454    0.884
S3_6312A       0.641    0.522
O1_6300A       1.428    0.391
O1_6364A       0.454    0.297
N2_6548A       5.657    0.624
H1r_6563A    285.000    0.251
N2_6584A      15.668    0.926
S2_6716A       0.995    0.728
S2_6731A       1.777    0.452
Ar3_7136A      3.882    0.448
O2_7319A+      5.106    0.160
O2_7330A+      4.034    0.715
```

```
In [38]: line = pn.EmissionLine(label='Cl3_5518A', obsIntens=[3.5, 2.5])
         obs.addLine(line)

In [39]: obs.printIntens()

S4_10.5m       4.076    0.192
Ne2_12.8m      4.826    0.023
Ne3_15.6m     19.803    0.427
S3_18.7m       5.802    0.337
O2_3726A      46.576    0.548
O2_3729A      21.812    0.576
Ne3_3869A     21.722    0.975
Ne3_3968A      7.255    0.532
S2_4069A       0.950    0.235
S2_4076A       0.503    0.971
O3_4363A       4.687    0.519
H1r_4861A    100.000    0.709
O3_5007A     425.599    0.616
N2_5755A       0.454    0.884
S3_6312A       0.641    0.522
O1_6300A       1.428    0.391
O1_6364A       0.454    0.297
N2_6548A       5.657    0.624
H1r_6563A    285.000    0.251
N2_6584A      15.668    0.926
S2_6716A       0.995    0.728
S2_6731A       1.777    0.452
Ar3_7136A      3.882    0.448
O2_7319A+      5.106    0.160
O2_7330A+      4.034    0.715
Cl3_5518A      5.534    3.953
```

0.2.4 Getting line intensities

You can extract the line intensities from an **Observation** object by, for example:

```
In [40]: obs.names

Out[40]: ['SMC_24', 'test']

In [41]: obs.getIntens(obsName='SMC_24')

Out[41]: {'Ar3_7136A': 3.8821497892916716,
          'Cl3_5518A': 5.5338388981739151,
          'H1r_4861A': 100.0,
          'H1r_6563A': 284.99999999999994,
          'N2_5755A': 0.45384655027842291,
          'N2_6548A': 5.6574680712548835,
          'N2_6584A': 15.668474852815743,
          'Ne2_12.8m': 4.8260297125551359,
          'Ne3_15.6m': 19.802702226158502,
          'Ne3_3869A': 21.721862162658756,
          'Ne3_3968A': 7.2549847099308185,
          'O1_6300A': 1.4279238824617235,
          'O1_6364A': 0.45369747693998408,
          'O2_3726A': 46.576439885253663,
          'O2_3729A': 21.812058000703153,
          'O2_7319A+': 5.1064244648045349,
          'O2_7330A+': 4.0337703225472392,
          'O3_4363A': 4.6870510740329472,
          'O3_5007A': 425.59915459530959,
          'S2_4069A': 0.95040451703046003,
          'S2_4076A': 0.50268123148205257,
          'S2_6716A': 0.99540541221817869,
          'S2_6731A': 1.7765637886180774,
          'S3_18.7m': 5.8018459111708029,
          'S3_6312A': 0.64146101211689832,
          'S4_10.5m': 4.0764787213393321}
```

```
In [42]: obs.getIntens()['O2_7330A+']
```

```
Out[42]: array([ 4.03377032,  0.71466343])
```

0.3 Using Observation to determine ionic abundances

Once the electron temperature and density are determined, it is easy to obtain the ionic abundances from a set of emission lines included in an **Observation** object:

```
In [43]: obs = pn.Observation()
         obs.readData('observations1.dat', fileFormat='lines_in_rows', err_default=
         obs.def_EBV(label1="H1r_6563A", label2="H1r_4861A", r_theo=2.85)
         obs.correctData(normWave=4861.)
         Te = [10000.]
         Ne = [1e3]
         # Define a dictionary to hold all the Atom objects needed
         all_atoms = pn.getAtomDict(atom_list=obs.getUniqueAtoms())
         # define a dictionary to store the abundances
         ab_dict = {}
         # we  use the following lines to determine the ionic abundances
         ab_labels = ['N2_6584A', 'O2_3726A', 'O3_5007A', 'S2_6716A',
                      'S3_6312A', 'Ar3_7136A', 'Ne3_3869A']
         for line in obs.getSortedLines():
             if line.label in ab_labels:
                 ab = all_atoms[line.atom].getIonAbundance(line.corrIntens, Te, Ne,
                                                           to_eval=line.to_eval, Hb
                 ab_dict[line.atom] = ab
```

```
warng _ManageAtomicData: rec data not available for Ar3
warng _ManageAtomicData: atom data not available for H1
warng _ManageAtomicData: coll data not available for H1
warng _ManageAtomicData: rec data not available for Ne2
warng _ManageAtomicData: rec data not available for Ne3
warng _ManageAtomicData: rec data not available for S2
warng _ManageAtomicData: rec data not available for S3
warng _ManageAtomicData: rec data not available for S4
```

```
In [44]: ab_dict
```

```
Out[44]: {'Ar3': array([  3.21287725e-07]),
          'N2': array([  3.22649264e-06]),
          'Ne3': array([  2.36596811e-05]),
          'O2': array([  3.64228102e-05]),
          'O3': array([ 0.00014833]),
          'S2': array([  6.71633117e-08]),
          'S3': array([  1.42901804e-06])}
```