Using PyCloudy 4

June 22, 2016

1 How to take account of the slit position when computing line intensities (even for a spherical nebula)

```
In [1]: %matplotlib inline
        import numpy as np
        import matplotlib.pyplot as plt
```

```
In [2]: import pyCloudy as pc
        from pyCloudy.utils.astro import conv_arc
```

```
In [3]: # The directory in which we will have the model
        # You may want to change this to a different place so that the current dire
        # will not receive all the Cloudy files.
        dir_ = '/DATA/NEBULATOM/'
```

```
In [4]: # Define some parameters of the model:
        model_name = 'model_4'
        full_model_name = '{0}{1}'.format(dir_, model_name)
        dens = 4. #log cm-3
        Teff = 45000. #K
        qH = 47. #s-1
        r_min = 5e16 #cm
        dist = 1.26 #kpc
```

```
In [5]: # these are the commands common to all the models (here only one ...)
        options = ('no molecules',
                   'COSMIC RAY BACKGROUND',
                   )
```

```
In [6]: emis_tab = ['H  1  4861',
                   'H  1  6563',
                   'He 1  5876',
                   'N  2  6584',
                   'O  1  6300',
                   'O II  3726',
                   'O II  3729',
                   'O  3  5007',
                   'TOTL  4363']
```

```
In [7]: abund = {'He' : -0.92, 'C' : 6.85 - 12, 'N' : -4.0, 'O' : -3.40, 'Ne' : -4.
                 'S' : -5.35, 'Ar' : -5.80, 'Fe' : -7.4, 'Cl' : -7.00}
```

```
In [54]: # Defining the object that will manage the input file for Cloudy
         c_input = pc.CloudyInput(full_model_name)
```

```
In [55]: # Filling the object with the parameters
         # Defining the ionizing SED: Effective temperature and luminosity.
         # The lumi_unit is one of the Cloudy options, like "luminosity solar", "q
         c_input.set_BB(Teff = Teff, lumi_unit = 'q(H)', lumi_value = qH)
```

```
In [56]: # Defining the density. You may also use set_dlaw(parameters) if you have
         c_input.set_cste_density(dens)
```

```
In [57]: # Defining the inner radius. A second parameter would be the outer radius
         c_input.set_radius(r_in=np.log10(r_min))
         c_input.set_abund(ab_dict = abund, nograins = True)
         c_input.set_other(options)
         c_input.set_iterate() # (0) for no iteration, () for one iteration, (N) fo
         c_input.set_sphere() # () or (True) : closed geometry, or (False): open ge
         c_input.set_emis_tab(emis_tab) # better use read_emis_file(file) for long
         c_input.set_distance(dist=dist, unit='kpc', linear=True) # unit can be 'kp
```

```
In [58]: # Writing the Cloudy inputs. to_file for writing to a file (named by full_
         c_input.print_input(to_file = True, verbose = False)
```

```
In [59]: # Running Cloudy with a timer. Here we reset it to 0.
         pc.log_.timer('Starting Cloudy', quiet = True, calling = 'test1')
         c_input.run_cloudy()
         pc.log_.timer('Cloudy ended after seconds:', calling = 'test1')
```

```
test1: Cloudy ended after seconds: in 53.3455228806
```

```
In [60]: c_output = pc.CloudyModel(full_model_name)
         c_output.print_stats()
```

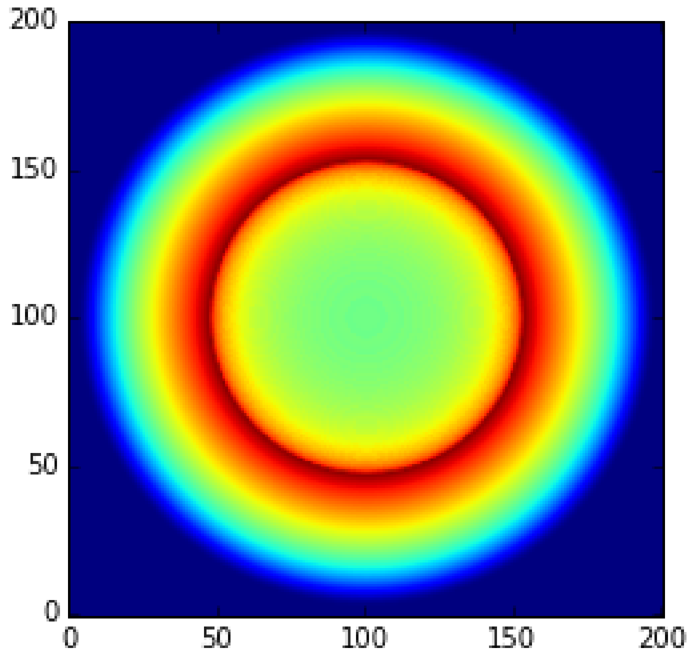
```
Name of the model: /DATA/NEBULATOMmodel_4
R_in (cut) = 5.000e+16 (5.000e+16), R_out (cut) = 9.521e+16 (9.521e+16)
H+ mass = 2.53e-02, H mass = 2.58e-02
<H+/H> = 0.99, <He++/He> = 0.00, <He+/He> = 0.90
<O+++/O> = 0.00, <O++/O> = 0.57, <O+/O> = 0.42
<N+++/O> = 0.01, <N++/O> = 0.67, <N+/O> = 0.32
T(O+++) = 8880, T(O++) = 8562, T(O+) = 9042
<ne> = 10858, T0 = 8767, t2=0.0025
<log U> = -2.31
```

```
In [53]: c_output = pc.CloudyModel(full_model_name)
         c_output.print_stats()
```

```
Name of the model: /DATA/NEBULATOMmodel_4
R_in (cut) = 5.000e+16 (5.000e+16), R_out (cut) = 9.526e+16 (9.526e+16)
H+ mass = 2.53e-02, H mass = 2.59e-02
<H+/H> = 0.99, <He++/He> = 0.00, <He+/He> = 0.90
<O+++/O> = 0.00, <O++/O> = 0.57, <O+/O> = 0.42
<N+++/O> = 0.01, <N++/O> = 0.67, <N+/O> = 0.32
T(O+++) = 8882, T(O++) = 8573, T(O+) = 9062
<ne> = 10859, T0 = 8782, t2=0.0025
<log U> = -2.31
```

In [46]:
```
# define the size of the 3D cube and instanciate the object that manage it
cube_size = 201
M_sphere = pc.C3D(c_output, dims=cube_size, center=True, n_dim=1)
```

In [47]:
```
# plot the image of the OIII emission
plt.imshow(M_sphere.get_emis('O__3__5007A').sum(0));
```



In [38]:
```
# A function in form of lambda to transform size in cm into arcsec, for a
arcsec = lambda cm: conv_arc(dist=dist, dist_proj=cm)
```

In [39]:
```
def make_mask(ap_center=[0., 0.], ap_size=[1., 1.]):
    """
    This returns a mask (values between 0. and 1.) to be multiplied to the
    An pc.C3D object named M_sphere must exist outside theis function
    """
```

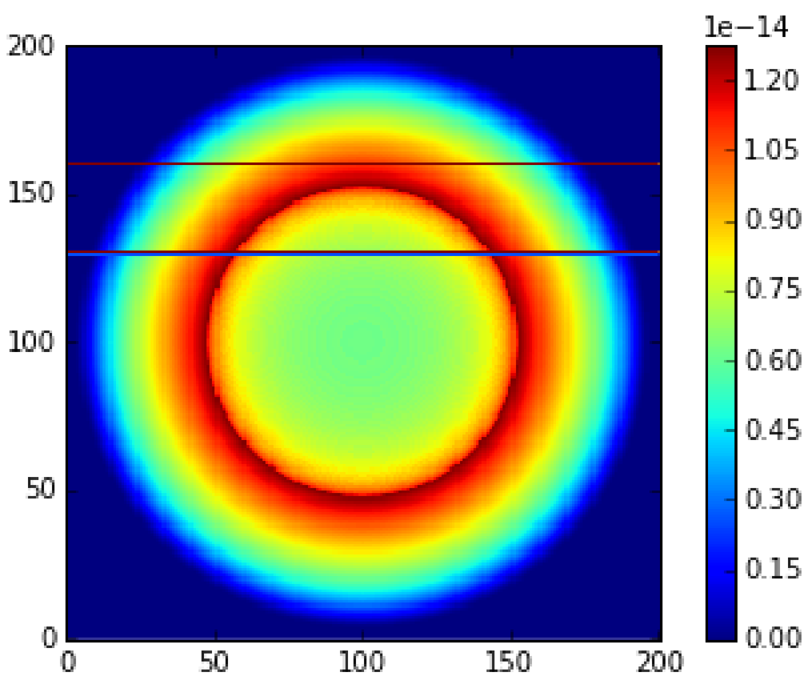
```
        x_arc = arcsec(M_sphere.cub_coord.x_vec)
        y_arc = arcsec(M_sphere.cub_coord.y_vec)
        z_arc = arcsec(M_sphere.cub_coord.z_vec)
        X, Y = np.meshgrid(y_arc, x_arc)
        bool_mask = ((X > ap_center[0] - ap_size[0]/2.) &
                    (X <= ap_center[0] + ap_size[0]/2.) &
                    (Y > ap_center[1] - ap_size[1]/2.) &
                    (Y <= ap_center[1] + ap_size[1]/2.))
        mask = np.zeros_like(X)
        mask[bool_mask] = 1.0
        return mask
```

In [48]:
```
# we define the mask. Can be change to see the effect of the aperture on l
mask = make_mask(ap_center=[1.5, 2.3], ap_size=[50, 1.5])
```

In [49]:
```
# Check that the mask is not empty
print mask.size
print mask.sum()
```

```
40401
6030.0
```

In [65]:
```
# We plot the OIII image and overplot the mask.
plt.imshow(M_sphere.get_emis('O__3__5007A').sum(0), interpolation='None')
plt.colorbar()
plt.contour(mask);
```



```
In [51]: # Hbeta is computed for the whole object and throught the aperture
         Hb_tot = (M_sphere.get_emis('H__1__4861A')*M_sphere.cub_coord.cell_size).s
         Hb_slit = ((M_sphere.get_emis('H__1__4861A')*M_sphere.cub_coord.cell_size)
         print Hb_tot, Hb_slit
```

```
4.60368236161e+34 8.73390462065e+33
```

```
In [63]: # For every line, we compute the intensity for the whole object and throug
         # We also print out the difference due to the slit.
         for label in M_sphere.m[0].emis_labels:
             I_tot = (M_sphere.get_emis(label).sum()*M_sphere.cub_coord.cell_size)
             I_slit = ((M_sphere.get_emis(label).sum(1) * mask).sum()*M_sphere.cub_
             print('line: {0:12s} I/Ib Total: {1:6.4f} I/Ib Slit: {2:6.4f} Delta: {
```

```
line: H__1__4861A  I/Ib Total: 1.0000 I/Ib Slit: 1.0000 Delta: -0.0%
line: H__1__6563A  I/Ib Total: 2.8910 I/Ib Slit: 2.8912 Delta:  0.0%
line: HE_1__5876A  I/Ib Total: 0.1843 I/Ib Slit: 0.1867 Delta:  1.3%
line: N__2__6584A  I/Ib Total: 1.1552 I/Ib Slit: 1.0025 Delta: -13.2%
line: O__1__6300A  I/Ib Total: 0.0197 I/Ib Slit: 0.0159 Delta: -19.0%
line: O_II__3726A  I/Ib Total: 0.7688 I/Ib Slit: 0.6799 Delta: -11.6%
line: O_II__3729A  I/Ib Total: 0.3430 I/Ib Slit: 0.3031 Delta: -11.6%
line: O__3__5007A  I/Ib Total: 3.7908 I/Ib Slit: 4.0056 Delta:  5.7%
line: TOTL__4363A  I/Ib Total: 0.0154 I/Ib Slit: 0.0161 Delta:  4.7%
```

```
In [ ]:
```