Using_pyCloudy_4

August 6, 2025

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

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import os
home_dir = os.environ['HOME'] + '/'
```

[2]:
```
import pyCloudy as pc
# Changing the location and version of the cloudy executable.
pc.config.cloudy_exe = '/usr/local/Cloudy/c25.00_rc2/source/cloudy.exe'
from pyCloudy.utils.astro import conv_arc
```

```
warng pyCloudy config: pyCloudy works better with matplotlib Triangulation
```

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

[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
```

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

[6]:
```
emis_tab = ['H  1  4861.32A',
            'H  1  6562.80A',
            'Ca B  5875.64A',
```

```
             'N  2  6583.45A',
             'O  1  6300.30A',
             'O  2  3726.03A',
             'O  2  3728.81A',
             'O  3  5006.84A',
             'O  3  4363.21A',
             'O 3R  4363.00A',
             'O 3C  4363.00A',
             'S  2  6716.44A',
             'S  2  6730.82A',
             'Cl 3  5517.71A',
             'Cl 3  5537.87A',
             'O  1  63.1679m',
             'O  1  145.495m',
             'C  2  157.636m']
```

[7]:
```
abund = {'He' : -0.92, 'C' : 6.85 - 12, 'N' : -4.0, 'O' : -3.40, 'Ne' : -4.00,
         'S' : -5.35, 'Ar' : -5.80, 'Fe' : -7.4, 'Cl' : -7.00}
```

[8]:
```
# Defining the object that will manage the input file for Cloudy
c_input = pc.CloudyInput(full_model_name)
```

[9]:
```
# 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(H)",
 ↪"ionization parameter", etc...
c_input.set_BB(Teff = Teff, lumi_unit = 'q(H)', lumi_value = qH)
```

[10]:
```
# Defining the density. You may also use set_dlaw(parameters) if you have a
 ↪density law defined in dense_fabden.cpp.
c_input.set_cste_density(dens)
```

[11]:
```
# Defining the inner radius. A second parameter would be the outer radius
 ↪(matter-bounded nebula).
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) for N
 ↪iterations.
c_input.set_sphere() # () or (True) : closed geometry, or (False): open
 ↪geometry.
c_input.set_emis_tab(emis_tab) # better use read_emis_file(file) for long list
 ↪of lines, where file is an external file.
c_input.set_distance(dist=dist, unit='kpc', linear=True) # unit can be 'kpc',
 ↪'Mpc', 'parsecs', 'cm'. If linear=False, the distance is in log.
```

[12]:
```
# Writing the Cloudy inputs. to_file for writing to a file (named by␣
↪full_model_name). verbose to print on the screen.
c_input.print_input(to_file = True, verbose = False)
```

[13]:
```
# 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')
```

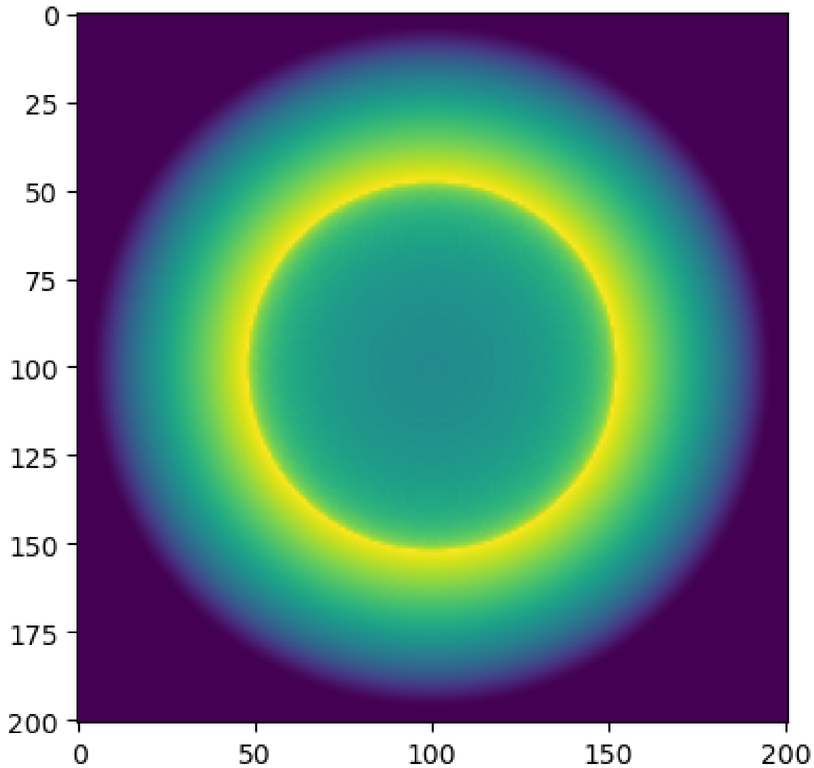
```
test1: Cloudy ended after seconds: in 45.66230607032776
```

[14]:
```
c_output = pc.CloudyModel(full_model_name)
c_output.print_stats()
```

```
Name of the model: /tmp/models/model_4
R_in (cut) = 5.000e+16 (5.000e+16), R_out (cut) = 9.577e+16 (9.576e+16)
Depth_in (cut) = 0.000e+00 (4.114e+11), depth_out (cut) = 4.576e+16 (4.576e+16)
H+ mass = 2.60e-02, H mass = 2.65e-02 N zones: 142
<H+/H> = 0.99, <He++/He> = 0.00, <He+/He> = 0.89
<O+++/O> = 0.00, <O++/O> = 0.56, <O+/O> = 0.42
<N+++/N> = 0.01, <N++/N> = 0.66, <N+/N> = 0.33
T(O+++) = 9108, T(O++) = 8773, T(O+) = 9195
<ne> = 10845,  <nH> = 10000, T0 = 8954, t2=0.0019
<log U> = -2.32
```

[15]:
```
# 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)
```

[16]:
```
# plot the image of the OIII emission
plt.imshow(M_sphere.get_emis('O__3_500684A').sum(0));
```



[17]:
```
# A function in form of lambda to transform size in cm into arcsec, for a␣
↪distance "dist" defined above.
arcsec = lambda cm: conv_arc(dist=dist, dist_proj=cm)
```

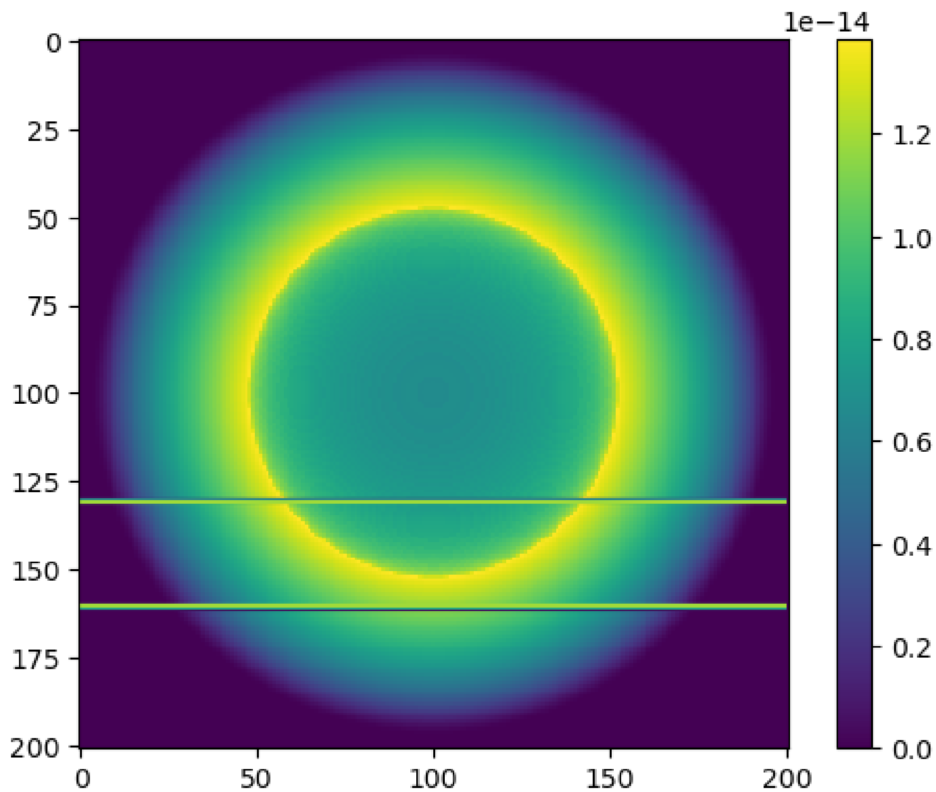
[18]:
```
def make_mask(ap_center=[0., 0.], ap_size=[1., 1.]):
    """
    This returns a mask (values between 0. and 1.) to be multiplied to the␣
↪image to take the flux passing through an aperture.
    An pc.C3D object named M_sphere must exist outside theis function
    """
    x_arc = arcsec(M_sphere.cub_coord.x_vec)
    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
```

[19]:
```python
# we define the mask. Can be change to see the effect of the aperture on line␣
↪intensities
mask = make_mask(ap_center=[1.5, 2.3], ap_size=[50, 1.5])
```

[20]:
```python
# Check that the mask is not empty
print(mask.size)
print(mask.sum())
```

```
40401
6030.0
```

[21]:
```python
# We plot the OIII image and overplot the mask.
plt.imshow(M_sphere.get_emis('O__3_500684A').sum(0), interpolation='None')
plt.colorbar()
plt.contour(mask);
```



[22]:
```python
# Hbeta is computed for the whole object and throught the aperture
Hb_tot = (M_sphere.get_emis('H__1_486132A')*M_sphere.cub_coord.cell_size).sum()
Hb_slit = ((M_sphere.get_emis('H__1_486132A')*M_sphere.cub_coord.cell_size).
↪sum(1) * mask).sum()
print(Hb_tot, Hb_slit)
```

```
4.644737852644873e+34 8.8146345401228e+33
```

[23]:
```
# For every line, we compute the intensity for the whole object and throught
 ↪the aperture.
# 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) /
 ↪Hb_tot
    I_slit = ((M_sphere.get_emis(label).sum(1) * mask).sum()*M_sphere.cub_coord.
 ↪cell_size) / Hb_slit
    print('line: {0:12s} I/Ib Total: {1:6.4f} I/Ib Slit: {2:6.4f} Delta: {3:4.
 ↪1f}%'.format(label, I_tot, I_slit,

 ↪          (I_slit-I_tot)/I_tot*100))
```

```
line: H__1_486132A I/Ib Total: 1.0000 I/Ib Slit: 1.0000 Delta: -0.0%
line: H__1_656280A I/Ib Total: 2.7941 I/Ib Slit: 2.7942 Delta:  0.0%
line: CA_B_587564A I/Ib Total: 0.1649 I/Ib Slit: 0.1680 Delta:  1.8%
line: N__2_658345A I/Ib Total: 1.1505 I/Ib Slit: 1.0013 Delta: -13.0%
line: O__1_630030A I/Ib Total: 0.0152 I/Ib Slit: 0.0124 Delta: -19.0%
line: O__2_372603A I/Ib Total: 0.8308 I/Ib Slit: 0.7371 Delta: -11.3%
line: O__2_372881A I/Ib Total: 0.3699 I/Ib Slit: 0.3279 Delta: -11.4%
line: O__3_500684A I/Ib Total: 4.0836 I/Ib Slit: 4.3307 Delta:  6.1%
line: O__3_436321A I/Ib Total: 0.0179 I/Ib Slit: 0.0189 Delta:  5.3%
line: O_3R_436300A I/Ib Total: 0.0000 I/Ib Slit: 0.0000 Delta: 11.8%
line: O_3C_436300A I/Ib Total: 0.0000 I/Ib Slit: 0.0000 Delta: 11.8%
line: S__2_671644A I/Ib Total: 0.0226 I/Ib Slit: 0.0196 Delta: -13.0%
line: S__2_673082A I/Ib Total: 0.0460 I/Ib Slit: 0.0400 Delta: -12.9%
line: CL_3_551771A I/Ib Total: 0.0023 I/Ib Slit: 0.0023 Delta:  1.4%
line: CL_3_553787A I/Ib Total: 0.0046 I/Ib Slit: 0.0047 Delta:  1.6%
line: O__1_631679M I/Ib Total: 0.0026 I/Ib Slit: 0.0021 Delta: -19.6%
line: O__1_145495M I/Ib Total: 0.0001 I/Ib Slit: 0.0001 Delta: -19.8%
line: C__2_157636M I/Ib Total: 0.0000 I/Ib Slit: 0.0000 Delta: -13.1%
```