PyCloudy

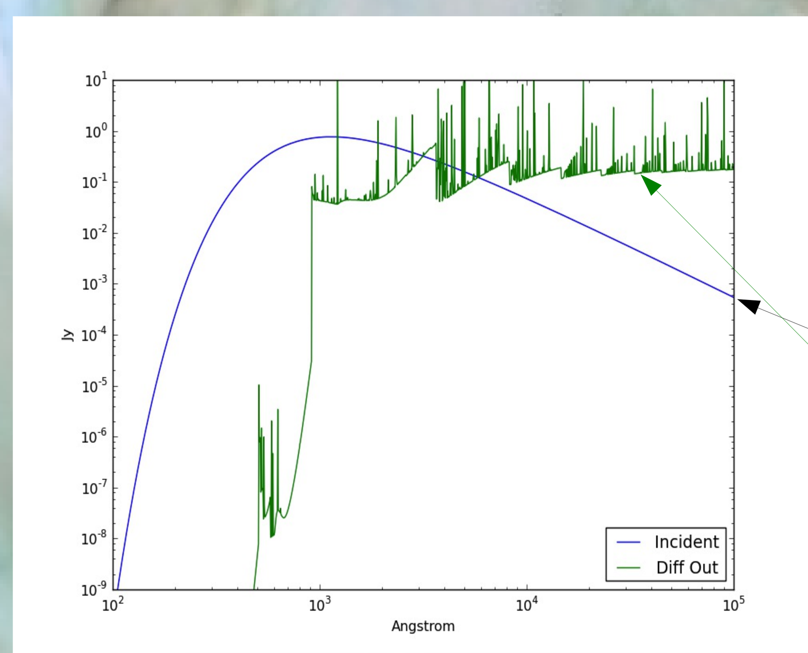
a new tool to 3D-model Planetary Nebulae

Christophe Morisset
Instituto de Astronomía
Universidad Nacional Autónoma de México

The python library pyCloudy is a set of tools to deal with photoionization code Cloudy (www.nublado.org).
This library allows you to:

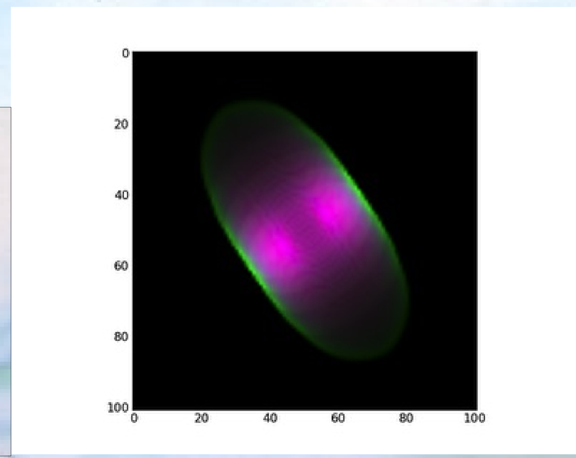
- Define and write input file(s) for Cloudy code. As you can have it in a code, you may generate automatically sets of input files, changing parameters from one to the other: easy grid of models.
- Read the Cloudy output files and play with the data: you will be able to plot line emissivity ratio vs. the radius of the nebula, the electron temperature, or any Cloudy output.
- Build pseudo-3D models, a la Cloudy_3D. This means: run a set of models, changing parameters (e.g. inner radius, density) following angular laws, read the outputs of the set of models and interpolate the results (Te, ne, line emissivities) in a 3D cube.
- If you have a multi-core computer, distinct models can be run at the same time on distinct cores (hard-way parallelisation).

```
import numpy as np
import pyCloudy as pc
import matplotlib.pyplot as plt
# Define verbosity to high level (will print errors, warnings and messages)
pc.log_.level = 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_ = '~/Models/'
# Define some parameters of the model:
model_name = 'model_1'
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
# these are the commands common to all the models (here only one ...)
options = ('no molecules')
emis_tab = ['H  1  4861', 'H  1  6563', 'He 1  5876', 'N  2  6584', 'O II  3726', 'O II  3729','O  3  5007','TOTL  4363']
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}
# Defining the object that will manage the input file for Cloudy
c_input = pc.CloudyInput(full_model_name)
# 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)
# 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)
# 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) : sphere, or (False): open geometry.
c_input.set_emis_tab(emis_tab)
c_input.set_distance(dist=dist, unit='kpc', linear=True) # unit can be 'kpc', 'Mpc', 'parsecs', 'cm'.
# 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)
c_input.run_cloudy()
# Reading the Cloudy outputs in the Mod CloudyModel object
Mod = pc.CloudyModel(full_model_name)
# Plotting Spectral energy distribution
plt.figure()
plt.loglog(Mod.get_cont_x(unit='Ang'), Mod.get_cont_y(cont = 'incid', unit = 'Jy'), label = 'Incident')
plt.loglog(Mod.get_cont_x(unit='Ang'), Mod.get_cont_y(cont = 'diffout', unit = 'Jy'), label = 'Diff Out')
plt.xlim((100, 100000))
plt.ylim((1e-9, 1e1))
plt.xlabel('Angstrom')
plt.ylabel('Jy')
plt.legend(loc=4)
plt.show()
```

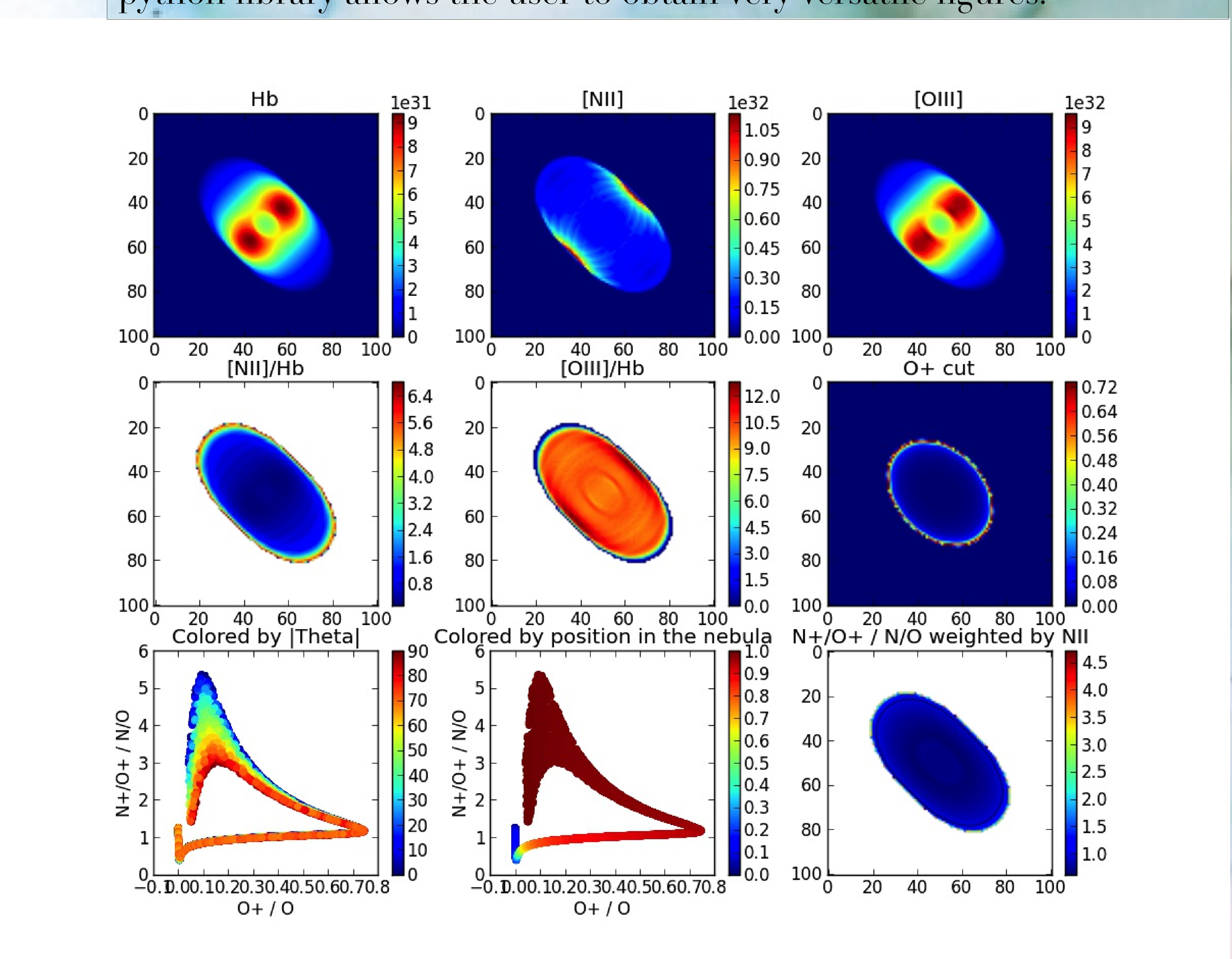


The pyCloudy package provides easy ways to obtain some Cloudy outputs in many different units. No more questions about erg/s/cm2 vs. Jy :-)

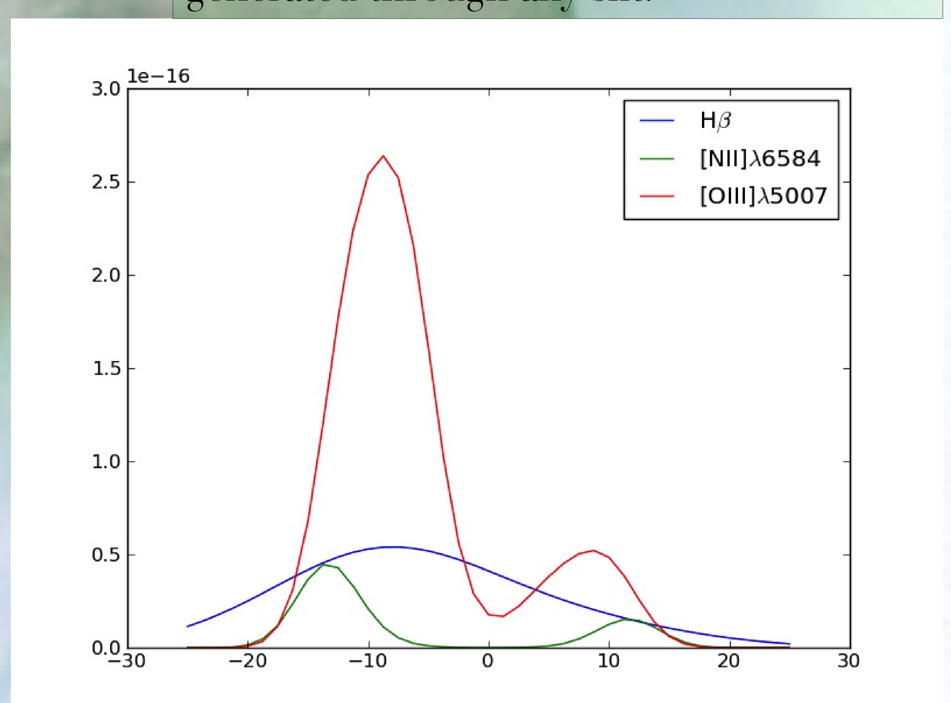
Once the 3D cube of emissivities is obtained, any rotation and projection on the sky plane can be obtained. False colors images (RGB) can be obtained, mixing any monochromatic images obtained by selecting emission lines.



Monochromatic images can be generated, as well as line ratio maps, cut within the volume of the nebula. From the 3D cells or the image spaxels, plots of any variable can be easily obtained. Matplotlib graphic python library allows the user to obtain very versatile figures.

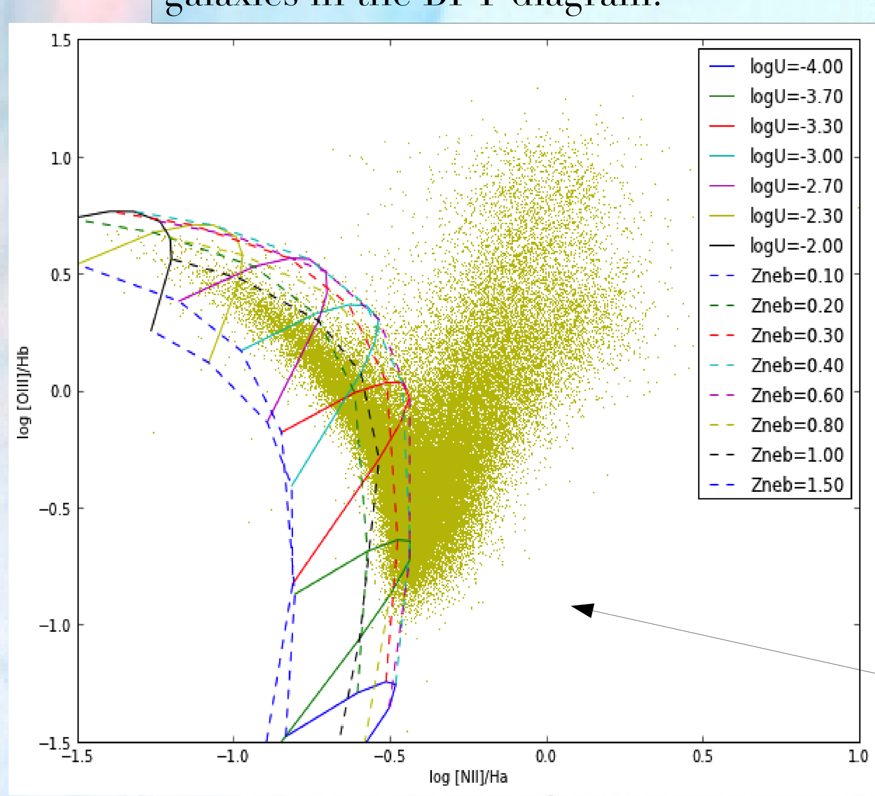


Once 3D emissivity cubes have been obtained, one can define a velocity field and compute line profiles through any aperture. PV-diagrams can also be generated through any slit.

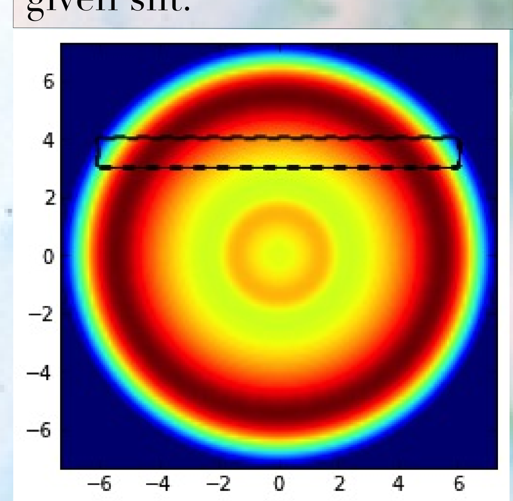



```
import numpy as np
import pyCloudy as pc
import matplotlib.pyplot as plt

def make_model(name, logU, logZ, models_dir='./'):

   abund_AGSS09 = {'He' : 10.93, 'C' : 8.43, 'N' : 7.83, 'O' : 8.69, 'Ne' : 7.93, 'Mg' : 7.6,
         'S' : 7.12, 'Ar' : 6.40, 'Fe' : 7.5, 'Cl' : 5.5, 'Si' : 7.51}
   for elem in abund_AGSS09:
      abund_AGSS09[elem] -= 12
      if elem != 'He':
         abund_AGSS09[elem] += logZ

   options = ('no molecules',
         'no level2 lines',
         'no fine opacities',
         'atom h-like levels small',
         'atom he-like levels small',
         'COSMIC RAY BACKGROUND',
         'element limit off -8',
         )

   c_input = pc.CloudyInput('{0}/{1}'.format(models_dir, name))
   c_input.set_star(SED = 'table star "ISB_008.mod"', SED_params = 1000000,
            lumi_unit = 'ionization parameter', lumi_value=logU)
   # 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(2)
   c_input.set_abund(ab_dict = abund_AGSS09, nograins = True)
   c_input.set_other(options)
   c_input.set_line_file('line_list.dat')
   c_input.set_iterate() # (0) for no iteration, () for one iteration, (N) for N iterations.
   c_input.set_sphere(False) # () or (True) : sphere, or (False): open geometry.
   c_input.set_distance(dist=10., unit='Mpc', linear=True) # unit can be 'kpc', 'Mpc', 'parsecs', 'cm'. If linear=False, the distance is in log.
   c_input.print_input()

def run_grid(models_dir, n_proc):
   pc.print_make_file(models_dir)

   # Metallicity table
   Zs = np.array([0.1, 0.2, 0.3, 0.4, 0.6, 0.8, 1.0, 1.5])
   # Ionization parameter table
   logUs = [-2, -2.3, -2.7, -3, -3.3, -3.7, -4]

   # loop on the tables to write the input files
   for Z in Zs:
      for logU in logUs:
         make_model(name='G_{0:.1f}_{1:.1f}'.format(logU, Z), logU=logU,
               logZ=np.log10(Z), models_dir=models_dir)

   # Run all the models
   pc.run_cloudy(dir_ = models_dir, n_proc = n_proc)

def plot_obs():
   obs = np.genfromtxt('BPT4Graz_f4.dat', names=True)
   plt.plot(obs['xN2Ha'], obs['yO3Hb'], 'y,', linestyle='None')

def plot_grid(Ms):
   # This plots the results of the grid.
   # A small function to extract a line intensity from all the models
   extract_line = lambda label:np.array([M.get_line(label) for M in Ms])
   Ha = extract_line('H__1__6563A')
   Hb = extract_line('H__1__4861A')
   O3 = extract_line('O__3__5007A')
   N2 = extract_line('N__2__6584A')
   # Recover the list of unique values of the input parameters from the name of the model:
   Z = np.array([float(M.model_name_s.split('_')[2]) for M in Ms])
   Z_u = np.sort(np.unique(Z))
   logU = np.array([float(M.model_name_s.split('_')[1]) for M in Ms])
   logU_u = np.sort(np.unique(logU))

   for logU1 in logU_u:
      s = np.where(logU == logU1)[0] # find the indices where logU is the current logU1
      indx = s[Z[s].argsort()] #find and sort the indices of the models with the
      plt.plot(np.log10(N2/Ha)[indx], np.log10(O3/Hb)[indx], linestyle='-', label = 'logU={0:.2f}'.format(logU1))
   for Z1 in Z_u:
      s = np.where(Z == Z1)[0] #
      indx = s[logU[s].argsort()]
      plt.plot(np.log10(N2/Ha)[indx], np.log10(O3/Hb)[indx], linestyle='--', label = 'Zneb={0:.2f}'.format(Z1))
   plt.legend()

   plt.xlim((-1.5, 1))
   plt.ylim((-1.5, 1.5))

###   HERE STARTS THE MAIN PROGRAM  ###
models_dir = '/Users/christophemorisset/DATAS/Choroni'
pc.config.cloudy_exe = '/usr/local/Cloudy/c10.00/cloudy.exe' # point to the location of Cloudy.exe

# this will run 56 models...
run_grid(models_dir, n_proc=3)
# this read and plot some SDSS data
plot_obs()
# reading all the models
Ms = pc.load_models(models_dir + '/G',
            read_lin = True, read_emis = False, read_cont = False,
            list_elem = [], read_phy = False, read_rad=False)
# over-plotting the grid on the observations
plot_grid(Ms)
```

Grid of models can easily be obtain. Example: building a grid of Cloudy models to separate pure star forming galaxies in the BPT diagram.



Once emission line images have been obtained, it is simple to put a mask on it to only extract the intensity observed through a given slit.



PyCloudy has ability to work with **PyNeb** library (Luridiana, Morisset & Shaw, 2013): using Te(r), Ne(r) and $X^i/X(r)$, PyNeb is used to re-compute the line emissivities according to any atomic data available from its large library.
Energetic balance is obviously slightly broken, but it still may be very useful to check effects of changing atomic data (not that easy from within Cloudy).

https://sites.google.com/site/pycloudy/