Jacopo Fritz, IRyA - UNAM

DataCubes in Astrophysics

Lesson 10

Extracting basic information

- We can use the (more or less…) same procedures to detect and measure emission lines in the cube;
- or… we can use the information that is already available in the kinematics cube (derived with Kubeviz);
- the “interesting” emission lines are (in most cases): Hβ, [OIII], [OI], [NII], Hα, [SII].

The final task: 4.1

- Use the fluxes available for Hα and Hβ to calculate the pixel-by-pixel value of the extinction (extinction map);
- use this extinction map to correct Hα flux and calculate the SFR for each pixel of the galaxy (sfr map);
- use the ratio of the [SII] lines to calculate the electron density (density map);
- use the other available lines to make a diagnostic diagram, and a ionization mechanism map.

Extinction: practically…

1. Which modules to import?
2. define a function to calculate Av

Extinction: practically…

1. Which modules to import?
2. define a function to calculate Av

```
import numpy as np
from astropy.io import fits
```

Extinction: practically…

1. Which modules to import?
2. define a function to calculate Av

```
import numpy as np
from astropy.io import fits

def Av_func(ha,hb):
    av = 1./(Av_b - Av_a)*2.5*(np.log10(ha/hb)-np.log10(2.86))
    if av > 6.:
        av = 0.
    return av
```

Extinction: practically…

1. Calculate Av_a and Av_b

Extinction: practically…

1. Calculate Av_a and Av_b

```
av1 = 0.8136172
av2 = 0.8167424
av3 = 1.271163
av4 = 1.278603
ha_l = 6562.8
hb_l = 4861.3
Av_a = (av1-av2)/(6570.-6550.)*(ha_l-6550.)+av2
Av_b = (av3-av4)/(4870.-4850.)*(hb_l-4850.)+av4
kfile = 'JO36_DATACUBE_FINAL_v1_ec_eo_res_gau_spax_noflag.fits'
```

Extinction: practically…

```
hdukin = fits.open(kfile)
k_head = hdukin[0].header
k_cube = hdukin[0].data
nx = k_head["NAXIS1"]
ny = k_head["NAXIS2"]
Ha_mat = k_cube[3,:,:]
Hb_mat = k_cube[6,:,:]
av_mat = np.zeros([ny,nx])
for _x in range(nx):
    for _y in range(ny):
        if (Ha_mat[_y,_x] > 0.0 and Hb_mat[_y,_x] > 0.0):
            av_mat[_y,_x] = Av_func(Ha_mat[_y,_x],Hb_mat[_y,_x])
        if av_mat[_y,_x] < 0.:
            av_mat[_y,_x] = 0.0
```

Extinction: practically…

```
newheader = k_head.copy()
del newheader[5:19]
del newheader[19:]
newheader["NAXIS"] = 2
newheader["COMMENT"]= "Extinction map for the galaxy JO36"
newheader["COMMENT"]= "Made with the routine av.py"
hdu_ext = fits.PrimaryHDU(av_mat,header=newheader)
hdu_ext.writeto(“ext_map.fits”)
```

SFR: practically…

Use the A_V macro and just add what you need to calculate the SFR

```
def sfr_func(ha):
    sfr = 4.*np.pi*(dist*mpc2cm)**2*ha*4.6e-42*1e-20
    return sfr

## define constants
dist = “put here the value of the distance of the galaxy”
mpc2cm = 3.0856775815E+24
```

SFR: practically…

```
if Ha_mat[_y,_x] > 0.0 :
    Ha_corr = Ha_mat[_y,_x]*10**(0.4*av_mat[_y,_x]*Av_a)
    sfr_map[_y,_x] = sfr_func(Ha_corr)
else:
    sfr_map[_y,_x] = -999.
```

Electron density calculation

It is possible to calculate the electron density from the two [SII] lines, in the hypothesis of T_{GAS}=10.000 K

$$n_e = 0.0543 \cdot \tan(-3.0553 \times R + 2.8506) + 6.98$$

$$-10.6905 \times R + 9.9186 \times R^2 - 3.5422 \times R^3)$$

Where: $R = F_{6714}/F_{6731}$

This is valid in the range 0.4 < R < 1.435

Proxauf+ 2014

ϱ(e⁻): practically…

1. write a function that calculates the electron density;
2. read the two [SII] line fluxes

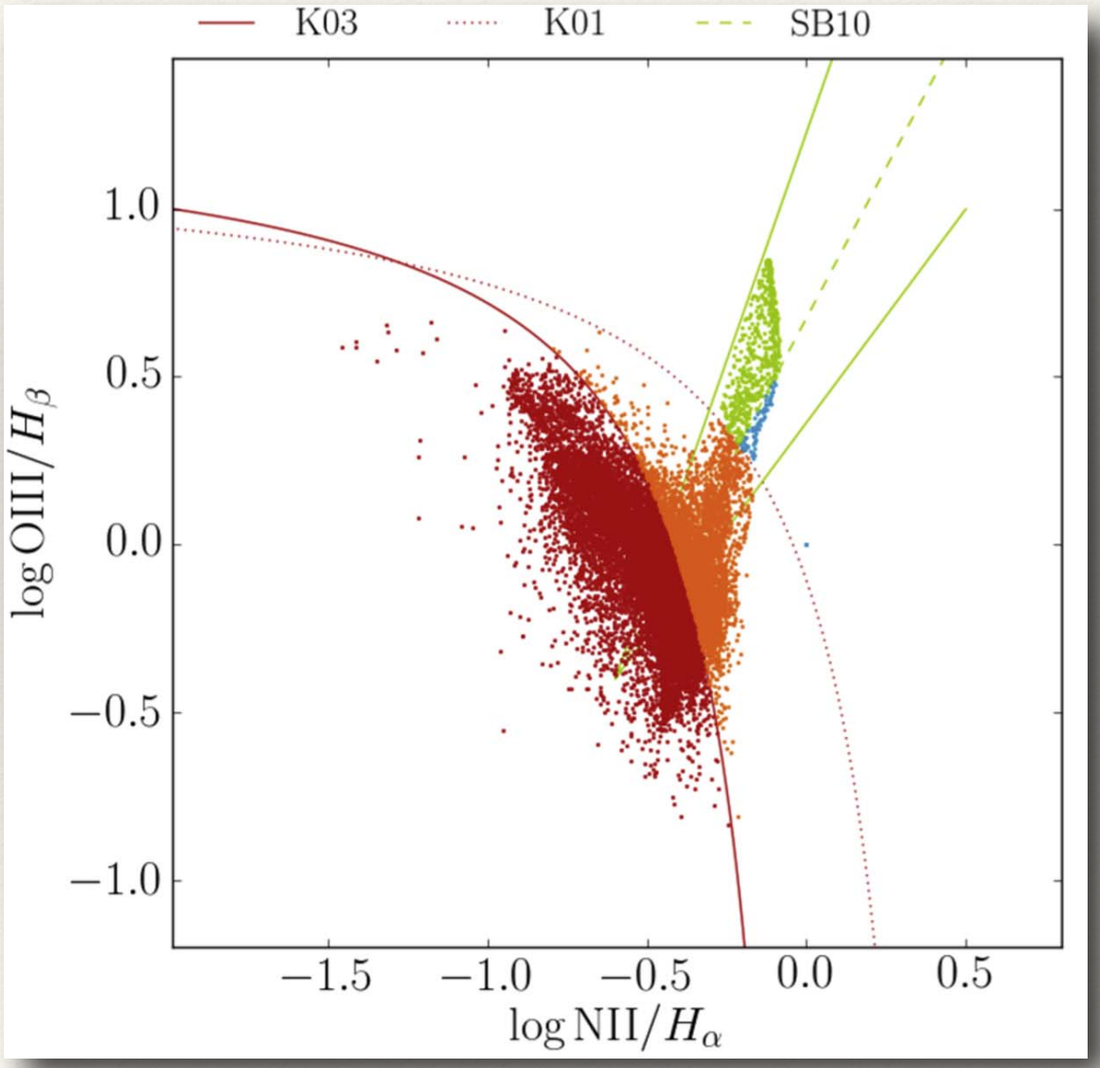
```
def edens(R):
    if R < 0.4:
        R = 0.4
    if R > 1.435:
        R = 1.435
    rho = 0.0543*np.tan(-3.0553*R+2.8506)+6.98-10.6905*R+
          9.9186*R**2-3.5422*R**3
    return rho
```

Diagnostic diagrams

- Emission lines are powered by ionizing photons;
- these are produced by a variety of mechanisms and sources (massive stars, AGN, hot evolved stars);
- shocks, thermal heating might be as well responsible for line ionization;
- diagnostic diagrams (or BPT, from Baldwin, Phillips & Terlevich 1981) use line ratios to disentangle different ionizing mechanisms.

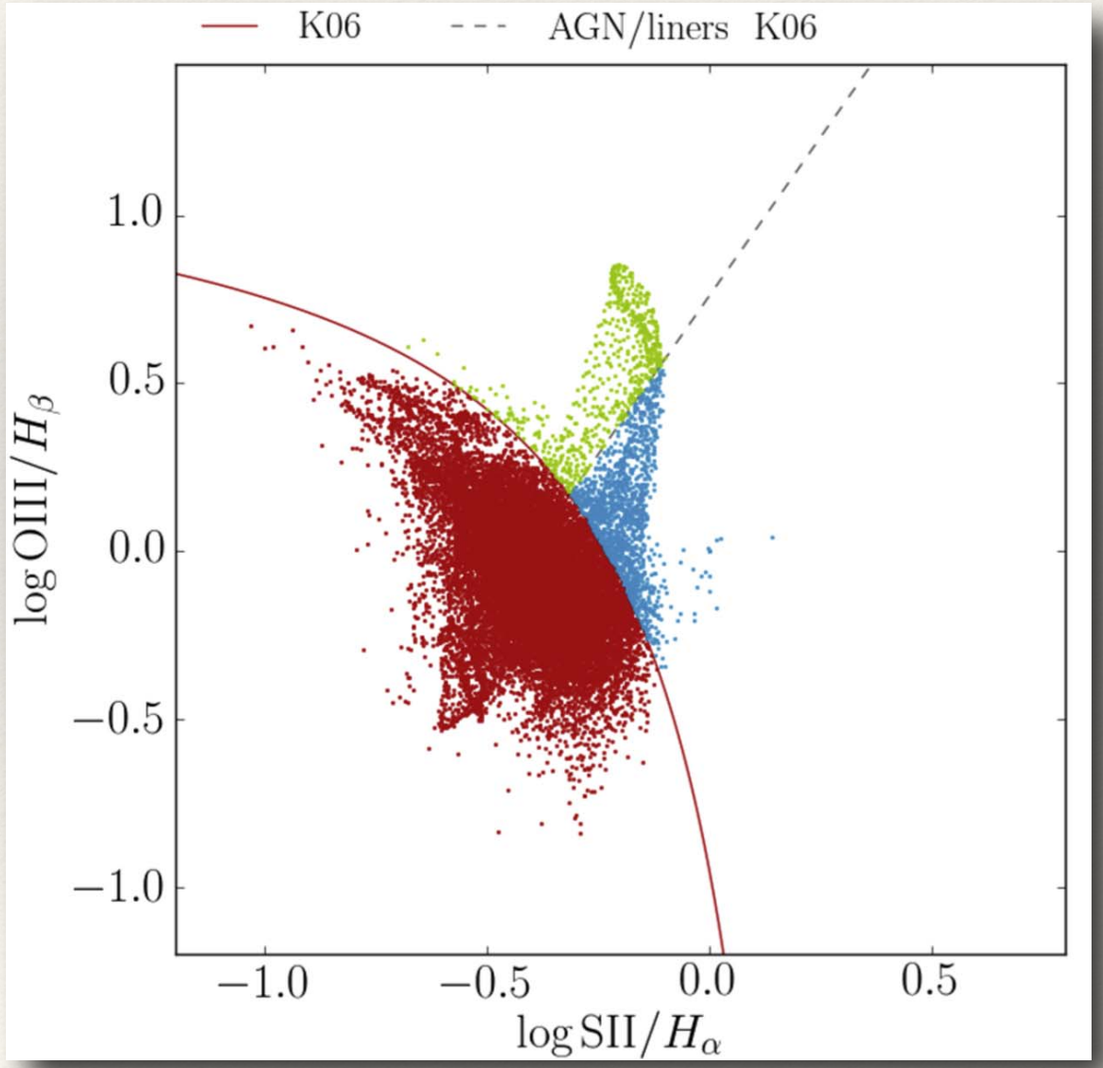
BPT 1

[NII] @ 6583 Å



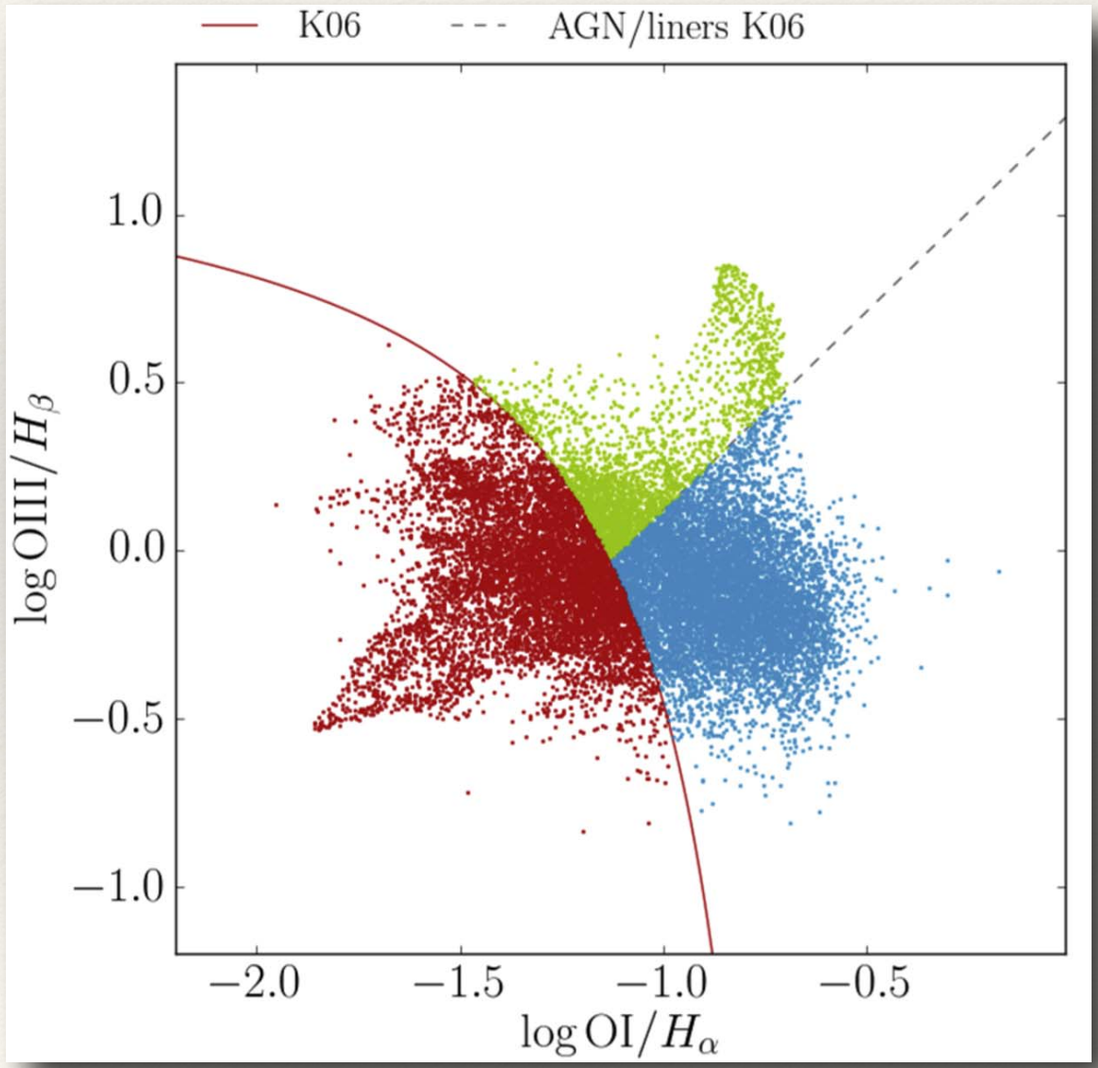
BPT 2

[SII] @ 6714+6731 Å



BPT 3

[OI] @ 6300 Å



BPT: practically…

```
import numpy as np
from astropy.io import fits
import pyregion
import matplotlib.pyplot as plt
from matplotlib import rc, rcParams
```

```
def f_k03(x):
    return ((0.61/(x-0.05))+1.3)

def f_k01(x):
    return ((0.61/(x-0.47))+1.19)

def f_s10(x):
    return (-0.4-1.5)/(-0.6-0.46)*(x-.46)+1.5
```

BPT: practically…

```
xk03=np.arange(-2., 0.04, 0.01)
xk01=np.arange(-2., 0.46, 0.01)
xs10=np.arange(-0.65, 0.5, 0.01)
k03=f_k03(xk03)                                  Kauffman
k01=f_k01(xk01)                                  Kauffman
s10=f_s10(xs10)

kfile = ‘filename_in_KINEMATICS/’
hdukin = fits.open(kfile)
k_head = hdukin[0].header
k_cube = hdukin[0].data
nx = k_head["NAXIS1"]
ny = k_head["NAXIS2"]
size = (ny,nx)
```

BPT: practically…

```
Ha = k_cube[3,:,:]
Hb = k_cube[6,:,:]
NIIr = k_cube[5,:,:]
OIII = k_cube[8,:,:]
source =  pyregion.open("diag3.reg")
mask = source.get_mask(shape=size)
x_rat1 = []
y_rat1 = []
x_rat2 = []
y_rat2 = []
x_rat3 = []
y_rat3 = []
x_rat4 = []
y_rat4 = []
```

BPT: practically…

```
NII_dia = np.zeros([ny,nx])
for n in range(nx):
    for i in range(ny):
        if mask[i,n] == True:
            if Ha[i,n] * Hb[i,n] * OIII[i,n] * NIIr[i,n] > 0.:
                xrat = 0.0
                yrat = 0.0
                xrat = np.log10(NIIr[i,n]/Ha[i,n])
                yrat = np.log10(OIII[i,n]/Hb[i,n])
                cond1=((xrat<0.05) and (yrat<f_k03(xrat)))
                cond3=((xrat<0.47) and (yrat>f_k01(xrat))
                        and (yrat>f_s10(xrat)))
                cond4=((xrat<0.47) and (yrat>f_k01(xrat))
                        and (yrat<f_s10(xrat)))
                cond2=((~cond1) and (~cond3) and (~cond4))
```

BPT: practically…

```
if cond1 == True:
    x_rat1.append(xrat)
    y_rat1.append(yrat)
    NII_dia[i,n] = 1
if cond3 == True:
    x_rat3.append(xrat)
    y_rat3.append(yrat)
    NII_dia[i,n] = 2
if cond4 == True:
    x_rat4.append(xrat)
    y_rat4.append(yrat)
    NII_dia[i,n] = 3
if cond2 == True:
    x_rat2.append(xrat)
    y_rat2.append(yrat)
    NII_dia[i,n] = 4
```

BPT: practically…

```
plt.plot(x_rat1, y_rat1, 'ro', color='red')
plt.plot(x_rat2, y_rat2, 'ro', color='orange')
plt.plot(x_rat3, y_rat3, 'ro', color='blue')
plt.plot(x_rat4, y_rat4, 'ro', color='green')
plt.plot(xk03,k03,linestyle='-', color='blue')
plt.plot(xk01,k01,linestyle=':', color='black',linewidth=4)
plt.plot(xs10,s10,linestyle='--', color='green',linewidth=3)
plt.xlabel(r'log([NII]/H$\alpha$)',fontsize=16)
plt.ylabel(r'log([OIII]/H$\beta$)',fontsize=16)
plt.xlim(-1,0.4)
plt.ylim(-1.15,1.)
```

END OF THE COURSE

END OF THE COURSE

…OR NOT…?