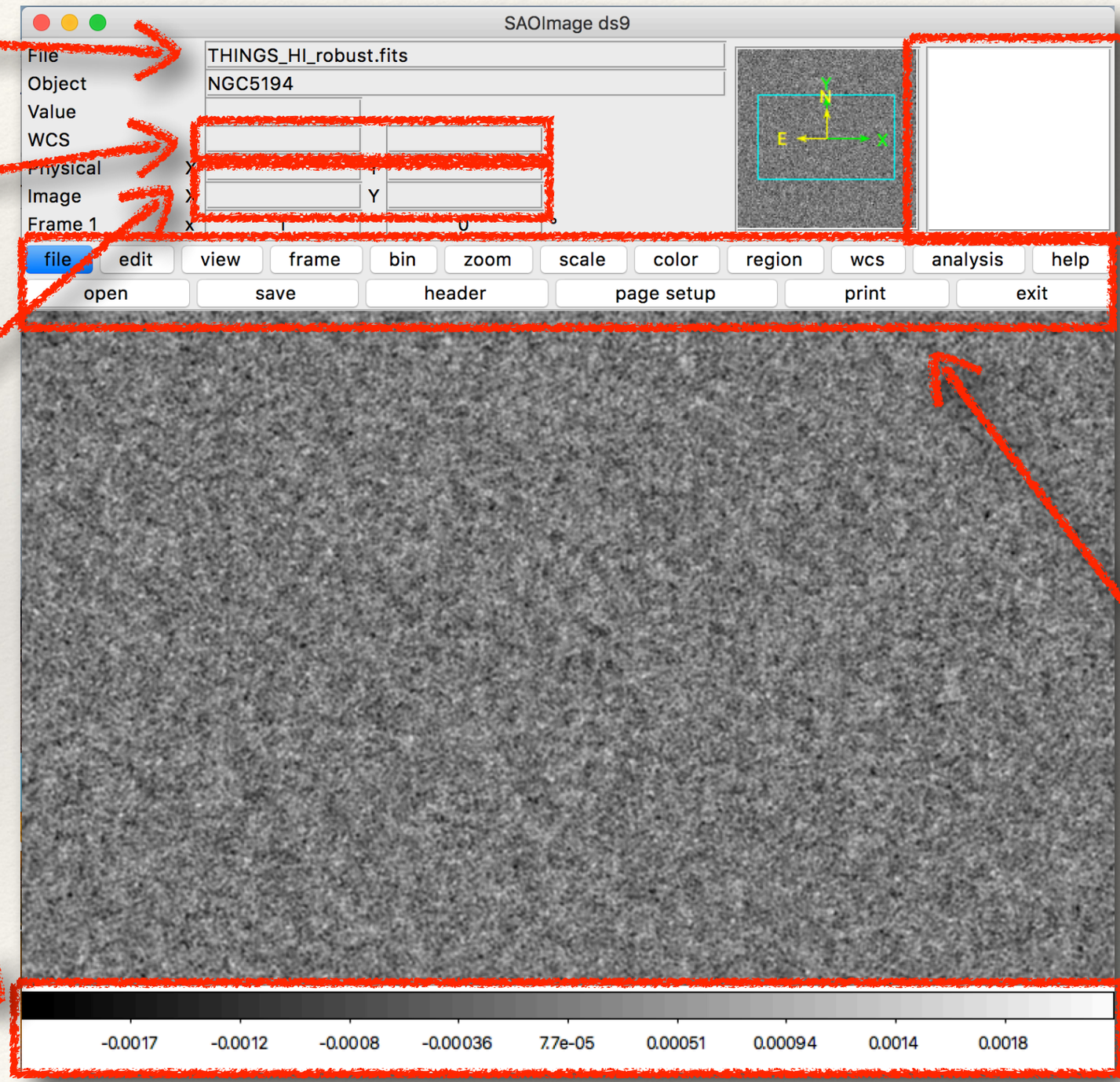
Jacopo Fritz, IRyA - UNAM

DataCubes in Astrophysics

Lesson 3

Tips to work with DS9



Filename

World coordinate system

Physical & Image coordinates

"Values" bar

Zoomed window

Options menu

Various data

Number of axes

Number of points on each axis

Observation details

Units of the pixels

Various other information

THINGS_HI_robust.fits

```
SIMPLE  =                    T /
BITPIX  =                  -32 /
NAXIS   =                    4 /
NAXIS1  =                 1024 /
NAXIS2  =                 1024 /
NAXIS3  =                   77 /
NAXIS4  =                    1 /
EXTEND  =                    T /Tables following main image
BLOCKED =                    T /Tape may be blocked
OBJECT  = 'NGC5194 '           /Source name
TELESCOP= 'VLA     '           /
INSTRUME= 'VLA     '           /
OBSERVER= 'AW605   '           /
DATE-OBS= '2004-04-26'         /Obs start date YYYY-MM-DD
DATE-MAP= '2005-05-19'         /Last processing date YYYY-MM-DD
BSCALE  =    1.00000000000E+00 /REAL = TAPE * BSCALE + BZERO
BZERO   =    0.00000000000E+00 /
BUNIT   = 'JY/BEAM '           /Units of flux
EPOCH   =      2.000000000E+03 /Epoch of RA DEC
VELREF  =                    2 />256 RADIO, 1 LSR 2 HEL 3 OBS
ALTRVAL =    1.41701197043E+09 /Altenate FREQ/VEL ref value
ALTRPIX =     -8.000000000E+00 /Altenate FREQ/VEL ref pixel
OBSRA   =    2.02466666667E+02 /Antenna pointing RA
OBSDEC  =    4.72000000000E+01 /Antenna pointing DEC
RESTFREQ=    1.42040575000E+09 /Rest frequency
DATAMAX =      4.857982043E-03 /Maximum pixel value
DATAMIN =     -3.028604435E-03 /Minimum pixel value
```

Coordinates

Coordinates type
for the various axes

```
CTYPE1  = 'RA---SIN'           /
CRVAL1  =    2.02466666667E+02 /
CDELT1  =     -4.166666768E-04 /
CRPIX1  =      5.120000000E+02 /
CROTA1  =      0.000000000E+00 /
CTYPE2  = 'DEC--SIN'           /
CRVAL2  =    4.72000000000E+01 /
CDELT2  =      4.166666768E-04 /
CRPIX2  =      5.130000000E+02 /
CROTA2  =      0.000000000E+00 /
CTYPE3  = 'FELO-HEL'           /
CRVAL3  =    5.87102000000E+05 /
CDELT3  =     -5.173266602E+03 /
CRPIX3  =      1.500000000E+01 /
CROTA3  =      0.000000000E+00 /
CTYPE4  = 'STOKES  '           /
CRVAL4  =    1.00000000000E+00 /
CDELT4  =      1.000000000E+00 /
CRPIX4  =      1.000000000E+00 /
CROTA4  =      0.000000000E+00 /
```

Coordinates

Coordinate value at the center of the reference pixel

```
CTYPE1  = 'RA---SIN'           /
CRVAL1  =    2.02466666667E+02 /
CDELT1  =     -4.166666768E-04 /
CRPIX1  =      5.120000000E+02 /
CROTA1  =      0.000000000E+00 /
CTYPE2  = 'DEC--SIN'           /
CRVAL2  =    4.72000000000E+01 /
CDELT2  =      4.166666768E-04 /
CRPIX2  =      5.130000000E+02 /
CROTA2  =      0.000000000E+00 /
CTYPE3  = 'FELO-HEL'           /
CRVAL3  =    5.87102000000E+05 /
CDELT3  =     -5.173266602E+03 /
CRPIX3  =      1.500000000E+01 /
CROTA3  =      0.000000000E+00 /
CTYPE4  = 'STOKES  '           /
CRVAL4  =    1.00000000000E+00 /
CDELT4  =      1.000000000E+00 /
CRPIX4  =      1.000000000E+00 /
CROTA4  =      0.000000000E+00 /
```

Coordinates

Coordinate increment along the axis

```
CTYPE1  = 'RA---SIN'           /
CRVAL1  =    2.02466666667E+02 /
CDELT1  =     -4.166666768E-04 /
CRPIX1  =      5.120000000E+02 /
CROTA1  =      0.000000000E+00 /
CTYPE2  = 'DEC--SIN'           /
CRVAL2  =    4.72000000000E+01 /
CDELT2  =      4.166666768E-04 /
CRPIX2  =      5.130000000E+02 /
CROTA2  =      0.000000000E+00 /
CTYPE3  = 'FELO-HEL'           /
CRVAL3  =    5.87102000000E+05 /
CDELT3  =     -5.173266602E+03 /
CRPIX3  =      1.500000000E+01 /
CROTA3  =      0.000000000E+00 /
CTYPE4  = 'STOKES  '           /
CRVAL4  =    1.00000000000E+00 /
CDELT4  =      1.000000000E+00 /
CRPIX4  =      1.000000000E+00 /
CROTA4  =      0.000000000E+00 /
```

Coordinates

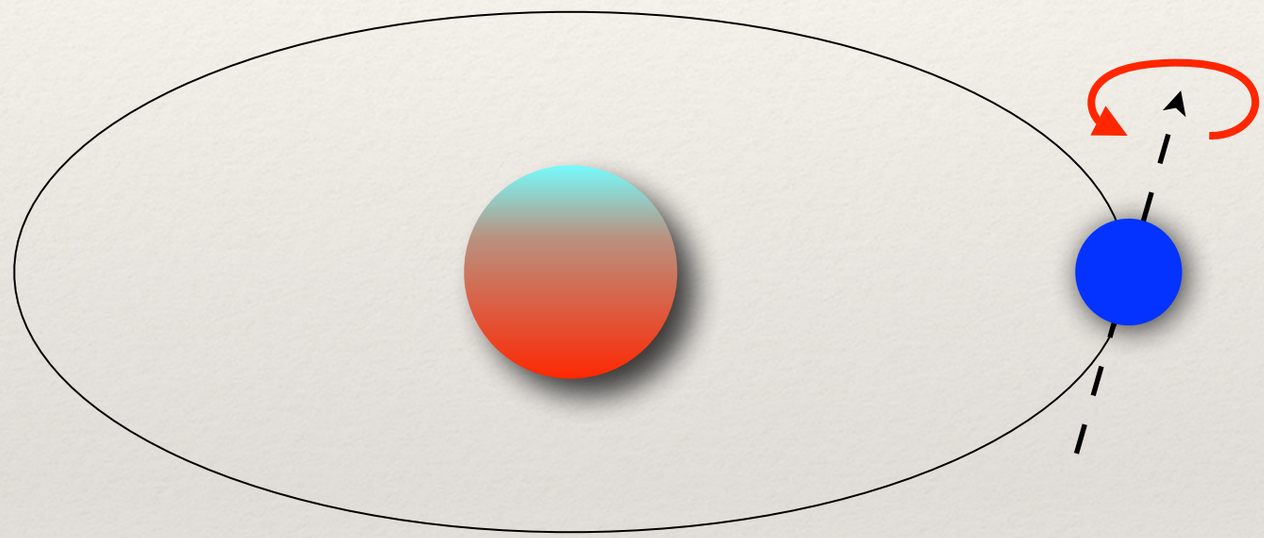
```
CTYPE1  = 'RA---SIN'           /
CRVAL1  =    2.02466666667E+02 /
CDELT1  =     -4.166666768E-04 /
CRPIX1  =      5.120000000E+02 /
CROTA1  =      0.000000000E+00 /
CTYPE2  = 'DEC--SIN'           /
CRVAL2  =    4.72000000000E+01 /
CDELT2  =      4.166666768E-04 /
CRPIX2  =      5.130000000E+02 /
CROTA2  =      0.000000000E+00 /
CTYPE3  = 'FELO-HEL'           /
CRVAL3  =    5.87102000000E+05 /
CDELT3  =     -5.173266602E+03 /
CRPIX3  =      1.500000000E+01 /
CROTA3  =      0.000000000E+00 /
CTYPE4  = 'STOKES  '           /
CRVAL4  =    1.00000000000E+00 /
CDELT4  =      1.000000000E+00 /
CRPIX4  =      1.000000000E+00 /
CROTA4  =      0.000000000E+00 /
```

Image coordinate of the reference pixel

Radio and mm datacubes

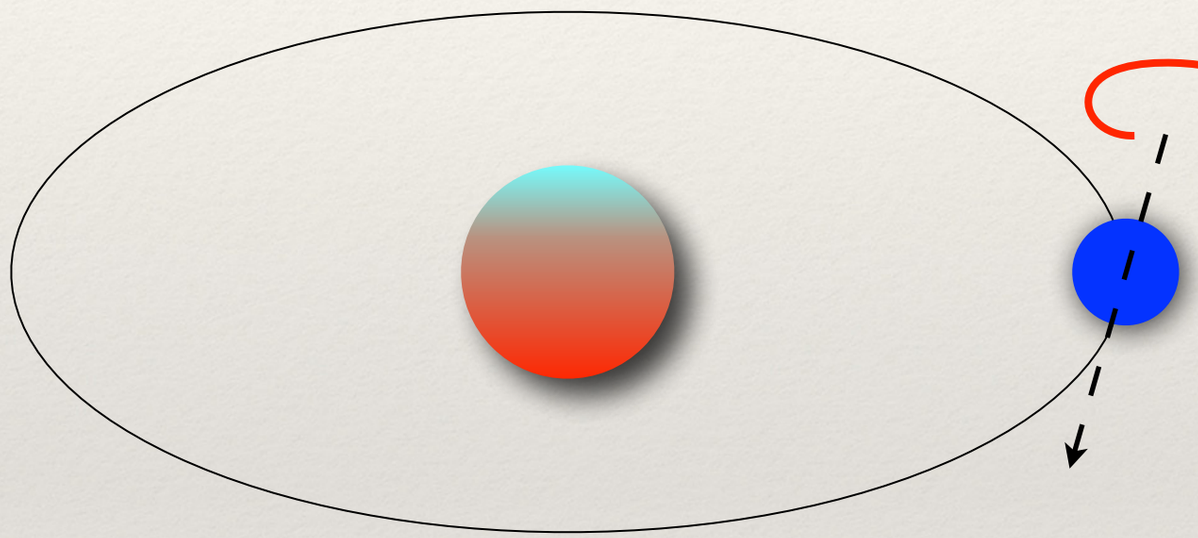
Atomic gas mass

The spin-inversion line: 21 cm (1420 MHz)



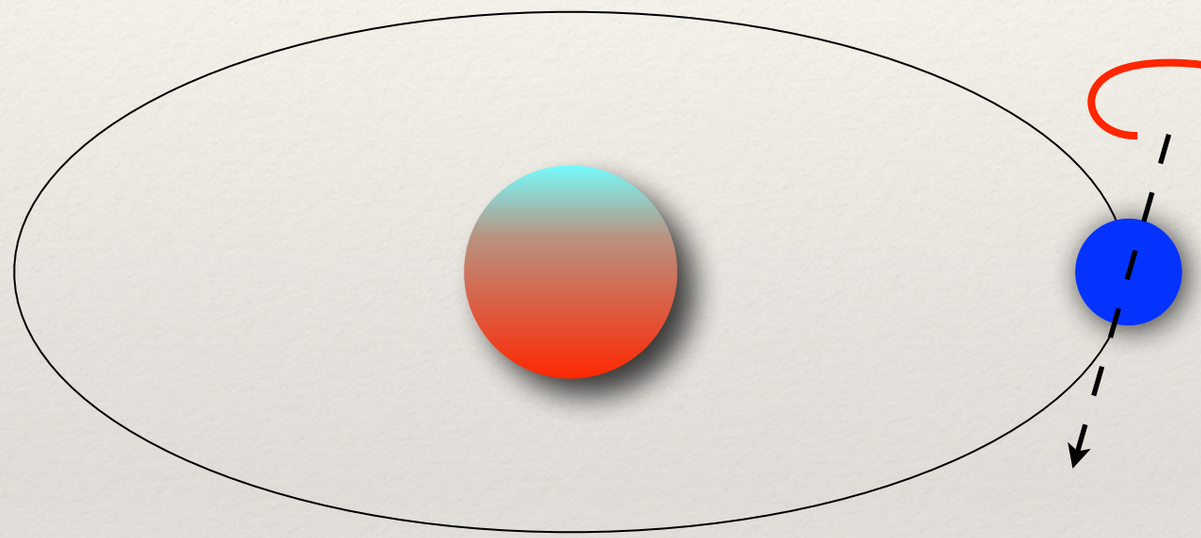
Atomic gas mass

The spin-inversion line: 21 cm (1420 MHz)



Atomic gas mass

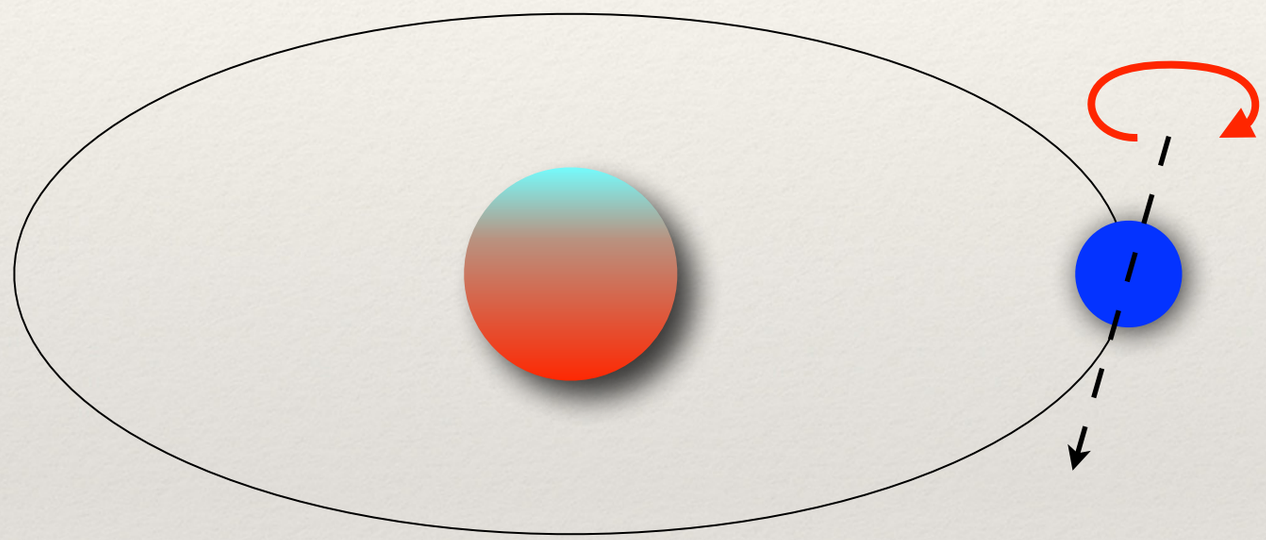
The spin-inversion line: 21 cm (1420 MHz)



- It is a collisionally-excited line;
- it is a forbidden one;
- has a very low probability (2×10^{-15} s^{-1});
- hence, one transition every 15.8×10^{6} yr!!

Atomic gas mass

The spin-inversion line: 21 cm (1420 MHz)

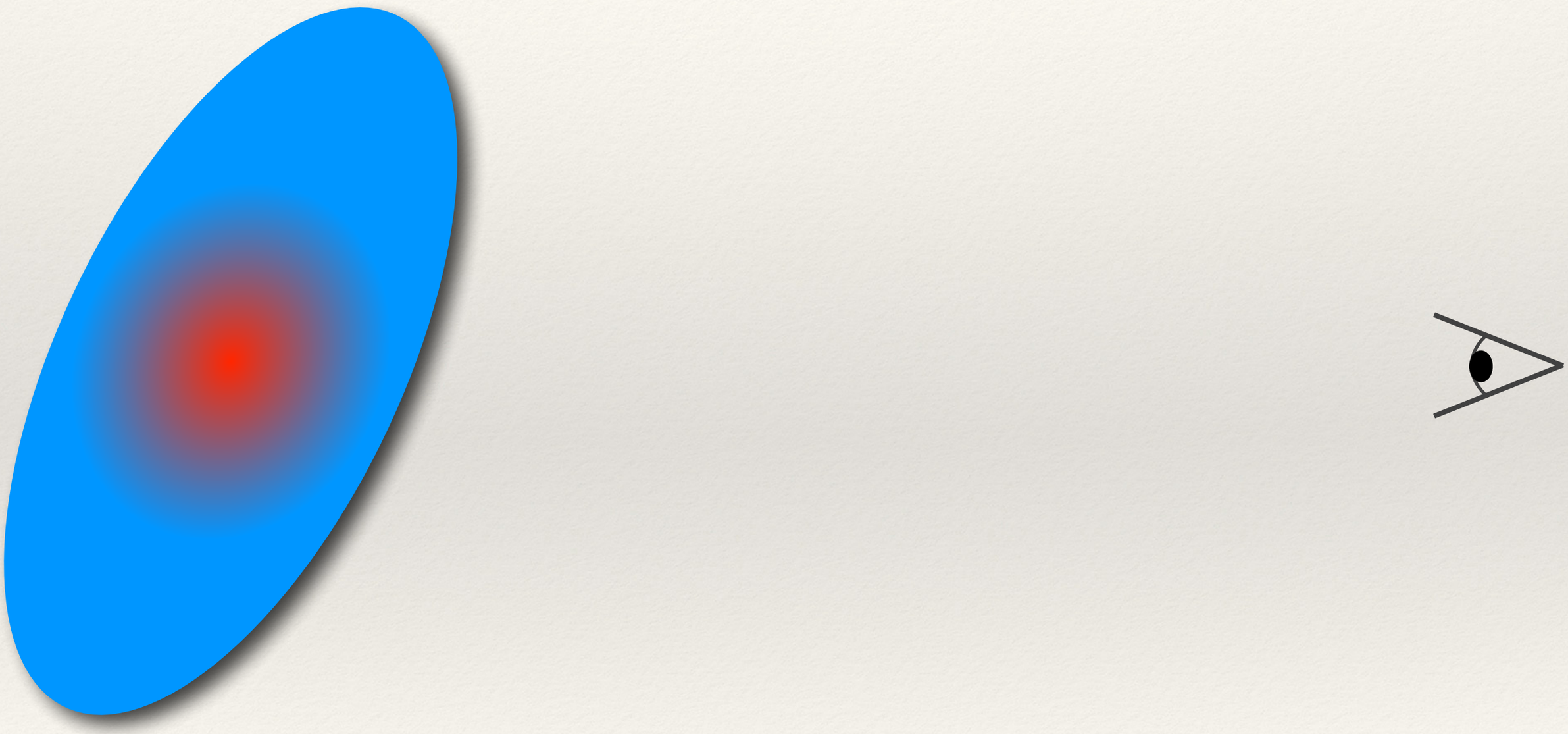


- It is a collisionally-excited line;
- it is a forbidden one;
- has a very low probability (2×10^{-15} s^{-1});
- hence, one transition every 15.8×10^{6} yr!!

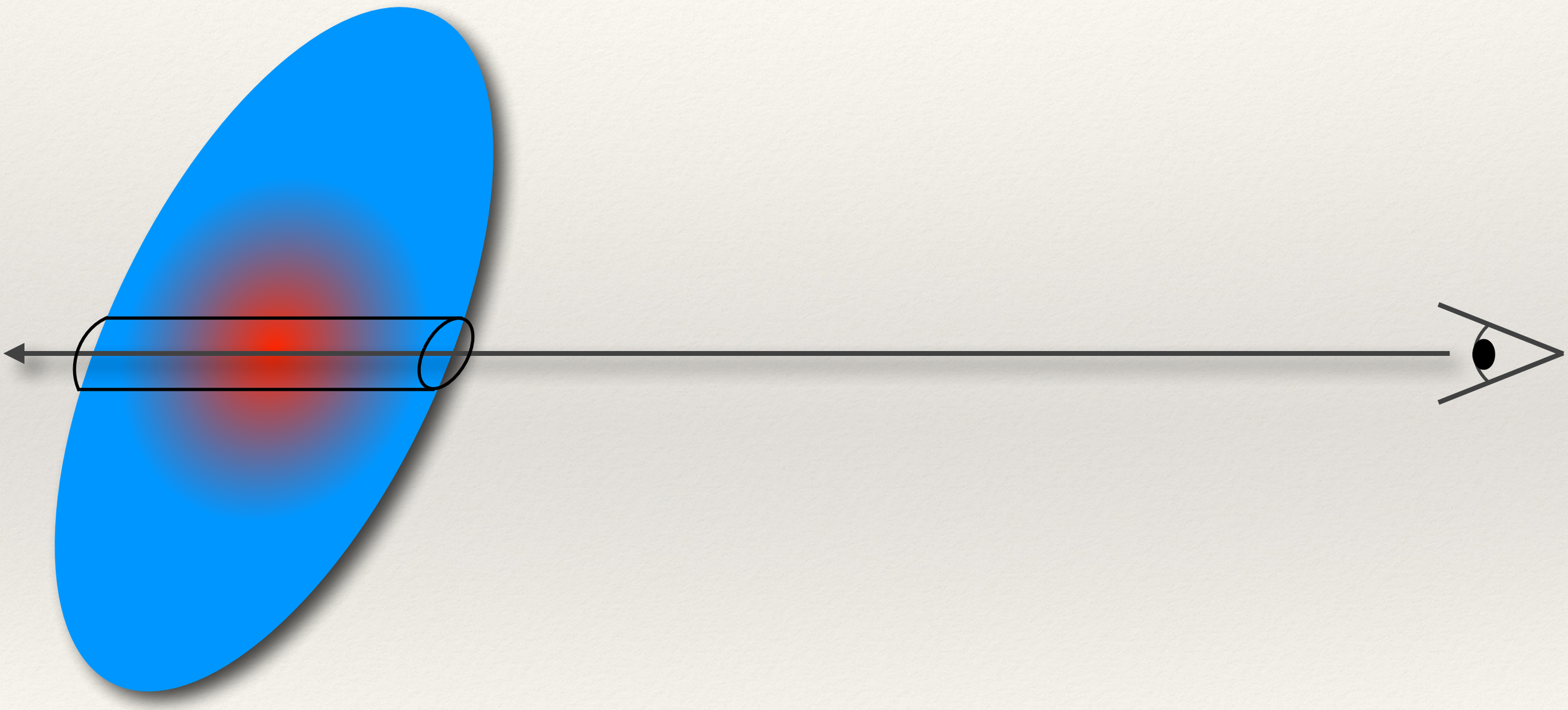
BUT:

Hydrogen is the most abundant element in the universe!!

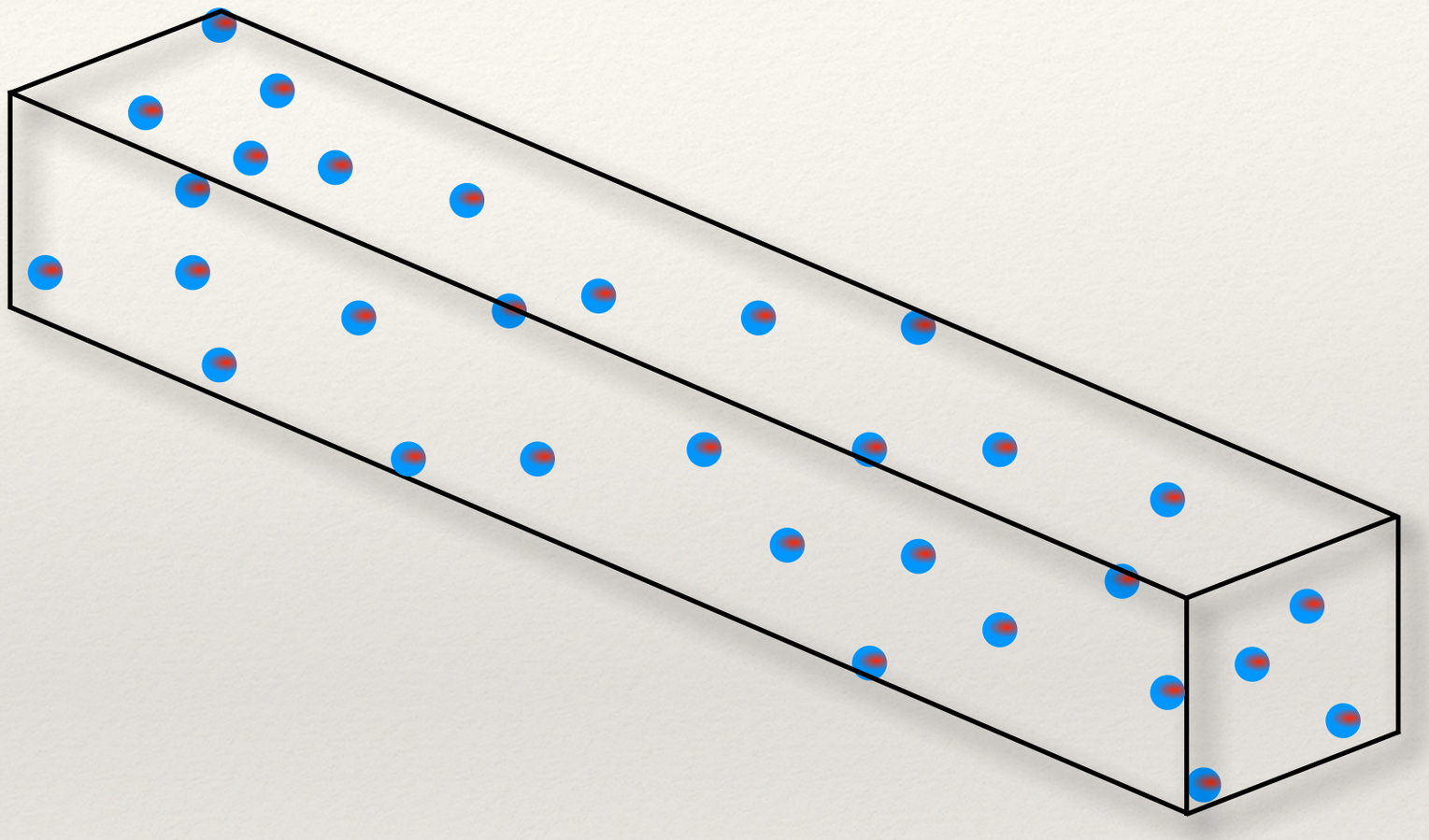
Column density of a gas

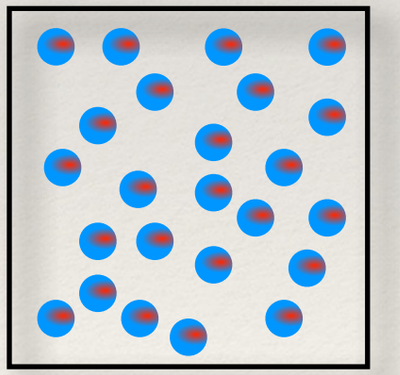


Column density of a gas



Column density of a gas





It is usually measured in cm^{-2}

Atomic hydrogen column density

- From the observed 21 cm line intensity, it is possible to derive the hydrogen column density;
- in the hypothesis of optically thin emission:

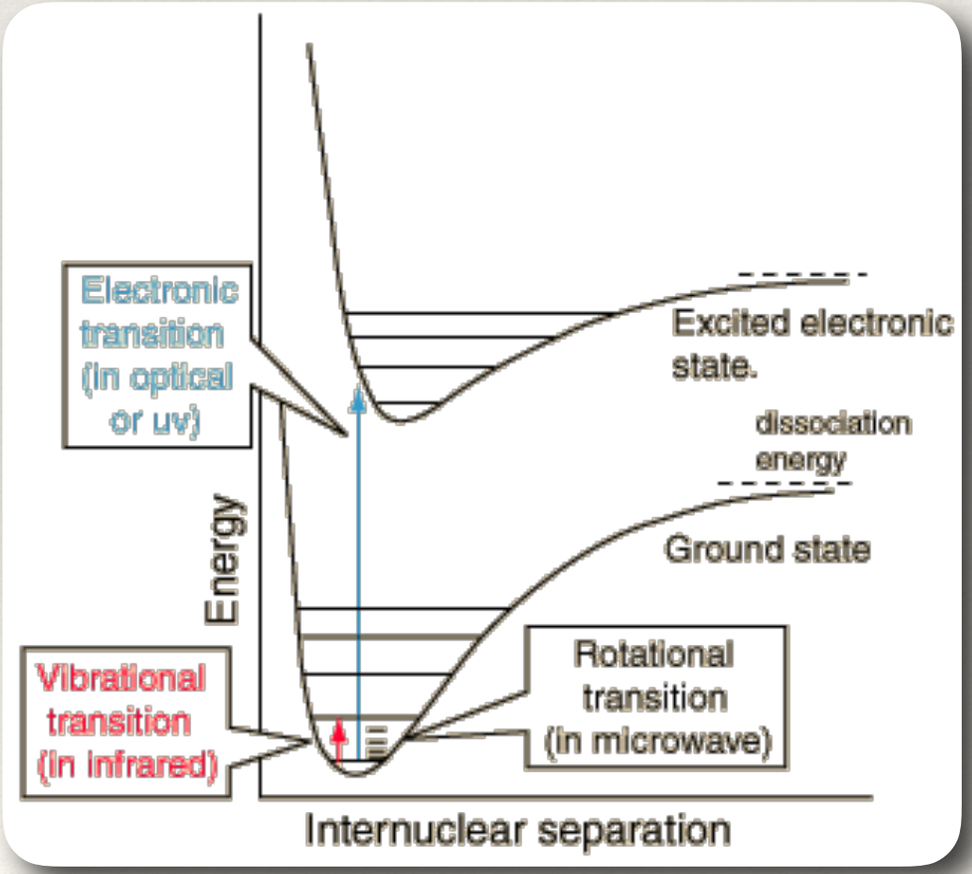
Brightness temperature

$$\left(\frac{N_{HI}}{\mathrm{cm}^{-2}}\right) \simeq 1.82 \times 10^{18} \underbrace{\int_{line} \left[\frac{T_b(v)}{\mathrm{K}}\right] d\left(\frac{v}{\mathrm{km} \cdot \mathrm{s}^{-1}}\right)}_{\text{Total line flux}} \qquad (1)$$

Where: $N_{HI} = \int_{los} n_{HI}(s) ds$ is the column density

Molecular gas

- Molecular emission lines are produced by vibration (IR) and rotation (mm/cm) of molecules;
- vibrational transitions require more energy and are more difficult to excite.



Molecular gas

- H_2 is the most abundant molecule in the ISM, but is symmetric and does not have a dipole moment;
- therefore, it does not emit detectable spectral lines at radio frequencies;
- H_2 is excited by the quadrupole moment;
- the $J = 2 \rightarrow 0$ transition requires an energy of ~ 500 K (FIR).

Molecular gas

- H_2 is the most abundant molecule in the ISM, but is symmetric and does not have a dipole moment;
- therefore, it does not emit detectable spectral lines at radio frequencies;
- H_2 is excited by the quadrupole moment;
- the $J = 2 \rightarrow 0$ transition requires an energy of ~ 500 K (FIR).

<u>BUT</u>:

- CO is the second most abundant molecule and is commonly used as a proxy to trace the cold molecular gas.

Molecular gas

- To calculate the amount of molecular hydrogen, we calculate the amount of CO, and convert it to molecular hydrogen (which, by far, dominates in mass this gas component);
- for this conversion to apply we assume:
    1. optically thin emission;
    2. typical solar abundances;
- other molecules can be used to trace gas with different physical conditions: e.g. HCN, HCO+ are commonly used as **dense** tracers in galaxies).

How to calculate it?

Conversion factor from Temperature to CO flux
+
Conversion factor from CO to H_2

$$\left[\frac{N_{H_2}}{\mathrm{cm}^{-2}}\right] \simeq 2 \times 10^{20} \underbrace{\int_{line} \left[\frac{T_b^{CO}(v)}{\mathrm{K}}\right] d\left(\frac{v}{\mathrm{km} \cdot \mathrm{s}^{-1}}\right)}_{\text{Total line flux}} \qquad (2)$$

Task 2

- We will analyze VLA and PdB data of the galaxy M51 (Whirpool), and extract their properties;
- we will calculate:
  1. the atomic and molecular gas column density map;
  2. the total atomic and molecular gas mass.

Task 2

Atomic and molecular gas column density
and mass determination

1. Use radio datacubes to determine the flux and velocities of the 21 cm line;
2. create a cube of the line intensity, velocity, and width;
3. create a map of atomic hydrogen column density;
4. calculate the total atomic mass;
5. do the same for the molecular gas from the mm map;
6. Finally, estimate the TOTAL (non-ionized) gas mass.

Let's get started!

- First step is to determine the characteristics of the lines (velocity, sigma, and intensity):
- this is done by fitting one gaussian (or more than one) to the spectra of each pixel;
- hence, you have to **<u>write a code</u>** that does that!
- use an optimizer function to find the three best-fitting values;
- minimize a χ^2 function.

Data Inspection

- Open the datacube with DS9;
- what is the structure of the cube?
- what comes in the header?
- what are the units of the quantities of interest?
- how do the data look like?

Remember we need to have the line's intensity in [K km s^{-1}] to derive a gas column density!

Converting the map's units

- The units of the map are Jy beam^{-1};
- hence, by measuring the line's intensity we get a map that is in units of Jy beam^{-1} km s^{-1};
- to convert it in to a HI column density (using equation 1), we need to have the intensity map in K km s^{-1};
- this is done by inverting the equation:

$$\left[\frac{I_\nu}{\text{mJy beam}^{-1}}\right] = 2.95 \left[\frac{T}{\text{K}}\right] \left[\frac{\theta_A}{\text{arcmin}}\right]^2 \left[\frac{\nu}{\text{GHz}}\right]^2 \qquad (3)$$

Importing modules

```
#!/usr/bin/env python
# -*- coding: utf8 -*-
import numpy as np
from astropy.io import fits
import matplotlib.pyplot as plt
from matplotlib import rc, rcParams
import scipy.optimize
```

Most of the methods of the `optimize` routine need a fairly educated guess of the range of the parameters.
This is why I suggest you to first make one or two fitting trials for one pixel.

Fitting, by χ^2 minimization

We need to find the best parameters of a gaussian function that minimize the difference between the observable (the line) and the model (the gaussian itself).

$$\chi^2 = \sum_{i=1}^{N_{points}} \left(\frac{O_i - M_i}{\sigma_i} \right)^2$$

Normally the *reduced* value is used:

$$\chi_\nu^2 = \frac{\chi^2}{N_{points} - N_{par}}$$

Defining functions

1. How do we write these gaussian and the X^2 functions?
2. What are the variables we will put in?

```
def gaussian(param):
    (v0,I0,sig) = param
    func = I0*prt/sig*np.exp(-0.5*((vel_array-v0)/sig)**2)
    return func

def chi2(param):
    (v0,I0,sig) = param
    chi=np.sum(((gaussian(param)-obs_spec)/cerr)**2)/float(ncont-3)
    return chi
```

Correct units and frequency

- To use eq. (3) and get the map in the desired units we need to know:
    1. the beam size;
    2. the frequency of each channel;
- This is how we get them

Calculating the beam size

- The beam, in this case, is elliptical;
- hence, it is defined by its major and minor axes;
- their values, in degrees, are found within the `HISTORY` keywords;
- these are: `bmaj` and `bmin`.

Then the beam's area is…

Calculating the frequency array

- The third dimension in this cube is velocity (in m s^{-1});
- we need it in frequency (Hz);
- we use the fact that frequency and velocity are connected by redshift:

$$z = \frac{v}{c}; \qquad \text{but:} \qquad \nu_o = \frac{\nu_e}{1+z} \quad \Rightarrow \quad \nu = \frac{\nu_0}{1+\frac{v}{c}}$$

- Where 0 is a reference frequency at the -8th place in the array.

Correct units and frequency

DATA READING

IMAGE PROPERTIES
ACQUISITION

BEAM DEFINITION

Correct units and frequency

```
obsfile = "THINGS_HI_robust.fits"

hdu_data = fits.open(obsfile)

obs = hdu_data[0].data

obs_head = hdu_data[0].header
```

IMAGE PROPERTIES ACQUISITION

BEAM DEFINITION

Correct units and frequency

```
obsfile = "THINGS_HI_robust.fits"

hdu_data = fits.open(obsfile)

obs = hdu_data[0].data

obs_head = hdu_data[0].header

nx = obs_head["NAXIS1"]

ny = obs_head["NAXIS2"]

nv = obs_head["NAXIS3"]

Dv = obs_head["CDELT3"]

vmin = obs_head["CRVAL3"]

pref = obs_head["CRPIX3"]

res_nu = obs_head["RESTFREQ"]

refv = np.abs(obs_head["ALTRPIX"])
```

BEAM DEFINITION

Correct units and frequency

```
obsfile = "THINGS_HI_robust.fits"

hdu_data = fits.open(obsfile)

obs = hdu_data[0].data

obs_head = hdu_data[0].header

nx = obs_head["NAXIS1"]

ny = obs_head["NAXIS2"]

nv = obs_head["NAXIS3"]

Dv = obs_head["CDELT3"]

vmin = obs_head["CRVAL3"]

pref = obs_head["CRPIX3"]

res_nu = obs_head["RESTFREQ"]

refv = np.abs(obs_head["ALTRPIX"])

bmaj =  1.6155e-3

bmin =  1.5437e-3

#in arcminutes plus the 1.442 factor to go from a circle to a gaussian
beam = np.sqrt(bmaj*bmin)*60.*1.442

```

Correct units and frequency

VELOCITY ARRAY

FREQUENCY ARRAY

CUBE DEFINITION

Correct units and frequency

```
velocity = np.zeros(nv)

for x in range(nv):

    velocity[x] = vmin + float(x-pref+1)*Dv

velocity = velocity*1e-3
```

FREQUENCY ARRAY

CUBE DEFINITION

Correct units and frequency

```
velocity = np.zeros(nv)

for x in range(nv):

    velocity[x] = vmin + float(x-pref+1)*Dv

velocity = velocity*1e-3


# here calculates the frequency array
dum_vel = np.zeros(nv+np.int(refv))

for x in range(nv+np.int(refv)):

    dum_vel[x] = vmin + (float(x)-pref+1.-refv)*Dv

dum_vel = dum_vel*1e-3

dum_freq = res_nu/(1.+dum_vel/light)/1e9

freq = dum_freq[refv:]
```

CUBE DEFINITION

Correct units and frequency

```
velocity = np.zeros(nv)

for x in range(nv):

    velocity[x] = vmin + float(x-pref+1)*Dv

velocity = velocity*1e-3


# here calculates the frequency array
dum_vel = np.zeros(nv+np.int(refv))

for x in range(nv+np.int(refv)):

    dum_vel[x] = vmin + (float(x)-pref+1.-refv)*Dv

dum_vel = dum_vel*1e-3

dum_freq = res_nu/(1.+dum_vel/light)/1e9

freq = dum_freq[refv:]

cube = obs[0,:,:,:]

```

Finding the best fit parameters

HOW MANY POINTS TO FIT?

WHAT PIXEL TO FIT (AS CHECK) ?

UNIT CONVERSION

CREATING THE SPECTRUM TO BE FITTED and ERROR CALCULATION

MINIMIZATION

GETTING THE RESULTS

Finding the best fit parameters

```
nd = 10     #how many channels before and after the maximum to fit a gaussian?
```

WHAT PIXEL TO FIT (AS CHECK) ?

UNIT CONVERSION

CREATING THE SPECTRUM TO BE FITTED and ERROR CALCULATION

MINIMIZATION

GETTING THE RESULTS

Finding the best fit parameters

```
nd = 10     #how many channels before and after the maximum to fit a gaussian?

_x = 440

_y = 386
```

UNIT CONVERSION

CREATING THE SPECTRUM TO BE FITTED and ERROR CALCULATION

MINIMIZATION

GETTING THE RESULTS

Finding the best fit parameters

```
nd = 10     #how many channels before and after the maximum to fit a gaussian?

_x = 440

_y = 386

spec = cube[:,_y,_x]*1e3/2.95*beam**(-2)*freq**(-2)
```

CREATING THE SPECTRUM TO BE FITTED and ERROR CALCULATION

MINIMIZATION

GETTING THE RESULTS

Finding the best fit parameters

```
nd = 10     #how many channels before and after the maximum to fit a gaussian?

_x = 440

_y = 386

spec = cube[:,_y,_x]*1e3/2.95*beam**(-2)*freq**(-2)

maxi = np.argmax(spec)

vmax = velocity[maxi]

vel_array = velocity[maxi-nd:maxi+nd]

obs_spec = spec[maxi-nd:maxi+nd]

ncont = len(vel_array)

cont = np.concatenate((spec[maxi-2*nd:maxi-nd],spec[maxi+nd:maxi+2*nd]))

cerr = np.std(cont)
```

MINIMIZATION

GETTING THE RESULTS

Finding the best fit parameters

```
nd = 10     #how many channels before and after the maximum to fit a gaussian?

_x = 440

_y = 386

spec = cube[:,_y,_x]*1e3/2.95*beam**(-2)*freq**(-2)

maxi = np.argmax(spec)

vmax = velocity[maxi]

vel_array = velocity[maxi-nd:maxi+nd]

obs_spec = spec[maxi-nd:maxi+nd]

ncont = len(vel_array)

cont = np.concatenate((spec[maxi-2*nd:maxi-nd],spec[maxi+nd:maxi+2*nd]))

cerr = np.std(cont)

x0 = [vmax, 100., 20.]

limits = [(vmax*0.90,vmax*1.05), (0,2e3), (0,50.)]

fit = scipy.optimize.minimize(chi2,x0,method='TNC',bounds=limits)
```

GETTING THE RESULTS

Finding the best fit parameters

```
nd = 10     #how many channels before and after the maximum to fit a gaussian?

_x = 440

_y = 386

spec = cube[:,_y,_x]*1e3/2.95*beam**(-2)*freq**(-2)

maxi = np.argmax(spec)

vmax = velocity[maxi]

vel_array = velocity[maxi-nd:maxi+nd]

obs_spec = spec[maxi-nd:maxi+nd]

ncont = len(vel_array)

cont = np.concatenate((spec[maxi-2*nd:maxi-nd],spec[maxi+nd:maxi+2*nd]))

cerr = np.std(cont)

x0 = [vmax, 100., 20.]

limits = [(vmax*0.90,vmax*1.05), (0,2e3), (0,50.)]

fit = scipy.optimize.minimize(chi2,x0,method='TNC',bounds=limits)

print "Best chi2: ",fit.fun

print "Best velocity ",fit.x[0]

print "Best intensity ",fit.x[1]

print "Best sigma ",fit.x[2]

```

PLOTTING

DEFINING PLOT LIMITS

PLOTTING SPECTRUM & MODEL

SETTING LABELS

PLOTTING...

PLOTTING

```
yma = max(spec)*1.2

ymi = min(spec)*1.2

xma = max(velocity)*1.01

xmi = min(velocity)*0.97

plt.figure(figsize=(15,10))
```

PLOTTING SPECTRUM & MODEL

SETTING LABELS

PLOTTING...

PLOTTING

```
yma = max(spec)*1.2

ymi = min(spec)*1.2

xma = max(velocity)*1.01

xmi = min(velocity)*0.97

plt.figure(figsize=(15,10))

plt.plot(velocity,spec,linestyle='-',color='black')

plt.plot(vel_array,chimodel,linestyle='-',color='blue')
```

SETTING LABELS

PLOTTING...

PLOTTING

```
yma = max(spec)*1.2

ymi = min(spec)*1.2

xma = max(velocity)*1.01

xmi = min(velocity)*0.97

plt.figure(figsize=(15,10))

plt.plot(velocity,spec,linestyle='-',color='black')

plt.plot(vel_array,chimodel,linestyle='-',color='blue')

plt.xlabel(r'Velocity  [m s$^{-1}$]',fontsize=20)

plt.ylabel(r'T  [K]',fontsize=20)

plt.tick_params(labelsize=20)

tstring = 'Pixel coordinates: x=' + str(_x)+' y='+str(_y)

plt.title(tstring)
```

PLOTTING...

PLOTTING

```
yma = max(spec)*1.2

ymi = min(spec)*1.2

xma = max(velocity)*1.01

xmi = min(velocity)*0.97

plt.figure(figsize=(15,10))

plt.plot(velocity,spec,linestyle='-',color='black')

plt.plot(vel_array,chimodel,linestyle='-',color='blue')

plt.xlabel(r'Velocity  [m s$^{-1}$]',fontsize=20)

plt.ylabel(r'T  [K]',fontsize=20)

plt.tick_params(labelsize=20)

tstring = 'Pixel coordinates: x=' + str(_x)+' y='+str(_y)

plt.title(tstring)

plt.xlim(xmi,xma)

plt.ylim(ymi,yma)

fig = plt.gcf()

plt.show()
```

Finding a good range

Try first plotting the model with some trial values, to get the starting point and the parameters limits.

```
trial = (vmax*1.01,8.e2,1.3e1)

model = gaussian_fit(trial)
```

Now loop over all the image

Now that you have found a good parameters' range, you can repeat the operation for all the pixels

```
results = np.zeros([4,ny,nx])
for _y in range(ny):
   for _x in range(nx):
      print "Doing pixel ",_x,_y
```

Writing parameters maps

Now that you are in a loop, you need to save the results for each pixel. Use the matrix results you have defined just before star iterating.

```
results[0,_y,_x] = fit.fun
results[1,_y,_x] = fit.x[0]
results[2,_y,_x] = fit.x[1]
results[3,_y,_x] = fit.x[2]
```

Writing parameters maps

```
outfile = 'fit_HI.fits'
newheader = obs_head.copy()
newheader['NAXIS3'] = 4
del newheader[#:#]
newheader['PLANE00'] = 'chi2'
newheader['PLANE01'] = 'velocity'
newheader['PLANE02'] = 'I0'
newheader['PLANE03'] = 'sigma'
hdu_s = fits.PrimaryHDU(results,header=newheader)
hdu_s.writeto(outfile)
```

Defining a fitting criterion

- Not all the pixel have a real signal;
- hence, some of the detections might just be noise variation;
- how do we choose where we have a real signal?
- set your fitting condition just before doing any operation;
- repeat the fitting again;
- check your results.

Dealing with the CO map

- Open the CO map, check the header, units, and perform a visual inspection;
- what are the differences with respect to the HI map?
- What are the changes you need to make to your fitting routine in order to get the column density map?

The final step

- From the parameters maps, calculate atomic and molecular hydrogen column density maps;
- calculate the total atomic and molecular hydrogen gas mass, by applying equation (1) and (2);
- what do you have to do to get the TOTAL neutral gas mass?

Extra steps?

- Make a quick study of the gas dynamics;
- smooth and re-grid the images so that they have the same resolution and pixel-size, and calculate a total gas column density map.

Summary

- Quick and brief review of DS9 user interface;
- DS9 coordinate dealing;
- emission from atomic and molecular gas (quick resume);
- fitting HI and CO emission lines in a datacube;
- creation of the moment maps;
- quick analysis.