Jacopo Fritz, IRyA - UNAM

DataCubes in Astrophysics

Lesson 6

And now the physics!

- Plot, in a log-log scale, the fluxes you have just calculated, with errorbars;
- write down, in the proper units, the modified blackbody equation;
- use it to find a fit to the data: what are the free-parameters?

How to?

Define `wavel`, `fluxes` and `errors`

```
yma = max(ofluxes)*2.
ymi = yma/80.
xma = max(wavel)*1.5
xmi = min(wavel)*0.7
plt.xscale('log')
plt.yscale('log')
plt.plot(wavel, ofluxes, 'ro', color='red')
plt.errorbar(wavel, ofluxes, yerr=errors, fmt='o')
plt.xlabel(r'$\lambda$  [$\mu$ m]',fontsize=16)
plt.ylabel(r'F$_\nu$  [Jy]',fontsize=16)
plt.tick_params(labelsize=16)
plt.xlim(xmi,xma)
plt.ylim(ymi,yma)
plt.show()
```

Modified Black-Body modeling

- We will model the total Infrared emission from M51 in the 5 Herschel bands with a modified black body fitting;
- this will give us the total dust mass, the dust temperature end the dust emissivity;
- we will also need to find uncertainty values on the dust properties derived in this way;
- Finally, we will repeat the same but on a pixel-by-pixel scale, to get the same properties as a function of the position.

Modified Black-Body modeling

Dust emission model

Modified Black-Body emission:

$$F_\nu = k_\nu M_D \frac{B_\nu(\lambda, T)}{D^2}$$

Where:

$$k_\nu = k_{\nu_0} \left(\frac{\nu}{\nu_0} \right)^\beta \qquad B_\nu(\nu, T) = \frac{2h\nu^3}{c^2} \cdot \frac{1}{e^{h\nu/kT} - 1}$$

1. Check (and chose, when possible) the units of each variable;
2. find the values of the constants (in the appropriate units).

Practically…

```
import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt
from matplotlib import rc, rcParams

kpc2cm = 3.08568e21          # conversion factor from Mpc to cm
h_p = 6.62607e-34            # Planck constant in m^2 kg s^-1
k_b = 1.3806485e-23          # Boltzmann constant in m^2 kg s^-2 K^-1
c_l = 2.99792458e8           # speed of light in m s^-1
msol = 1.988e30              # solar mass value in kg
dist = ???                  # Distance do the galaxy in Mac
knu0 = 0.192                 # dust emissivity at 350 µm, in m^2 kg^-1
nu0 = c_l*1e6/350.           # frequency corresponding to 350 micron
```

Practically...

```
def B_nu(tt):
    #in W sr^-1 m^-2 Hz^-1
    return (2.*h_p*nu**3)/c_l**2*1./(np.exp((h_p*nu)/(k_b*tt))-1.)

def mbb_nu(md,tt,bb):
    #in W cm^-2 Hz^-1
    mbb_f = md*msol*knu0*(nu/nu0)**bb*B_nu(tt)/(dist*kpc2cm)**2
    mbb_erg = 1e7*mbb_f    #in erg s^-1 cm^-2 Hz^-1
    return mbb_erg*1e23    #in Jy

def chi2(params):
    md, tt, bb = params
    return np.sum(((ffunc_nu(md,tt,bb)-fluxes)/errors)**2)
```

Practically…

```
wavel = [70., 160., 250., 350., 500.]
ofluxes = [PONGAN SUS FLUJOS AQUI]
errors =  [PONGAN LOS ERRORES AQUI]
nu = np.zeros(5)
nu = c_l*1e6*np.divide(1.,wavel)     #frequency array in Hz
x0 = [???, ???, ???]
limits = [(???,???), (???,???), (???,???)]
fluxes = ofluxes
fit = minimize(chi2, x0, bounds=limits, method='TNC')
print "Chi2 value for this fit: ",fit.fun
print "Best fit dust mass: ",fit.x[0]
print "Best fit dust temperature: ",fit.x[1]
print "Best fit dust emissivity: ",fit.x[2]
```

Practically…

Instruction from the `plt.py` macro here

```
nwav = 201
llmin = 1.5
llmax = 2.9
dl = (llmax-llmin)/float(nwav-1)
lwav = np.zeros(nwav)
for _i in range(nwav):
   lwav[_i]=llmin+dl*float(_i)
wavel = 10**lwav      # wavelength array in microns
nu = c_l*1e6*np.divide(1.,wavel)
bb_emission = ffunc_nu(mass,temp,beta)
plt.plot(wavel, bb_emission, linestyle='-', color='black')
plt.xlabel(r'$\lambda$  [$\mu$ m]',fontsize=16)
plt.ylabel(r'F$_\nu$  [Jy]’,fontsize=16)
plt.show()
```

Model-Fitting

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{x_i - \mu_i}{\sigma_i} \right)^2$$

- The chi-squared statistic is a measure of the goodness of fit of the model to the data;
- We penalize the statistic according to how many standard deviations each model point lies **from** the data;
- note that this equation assumes that the **data points are independent**.

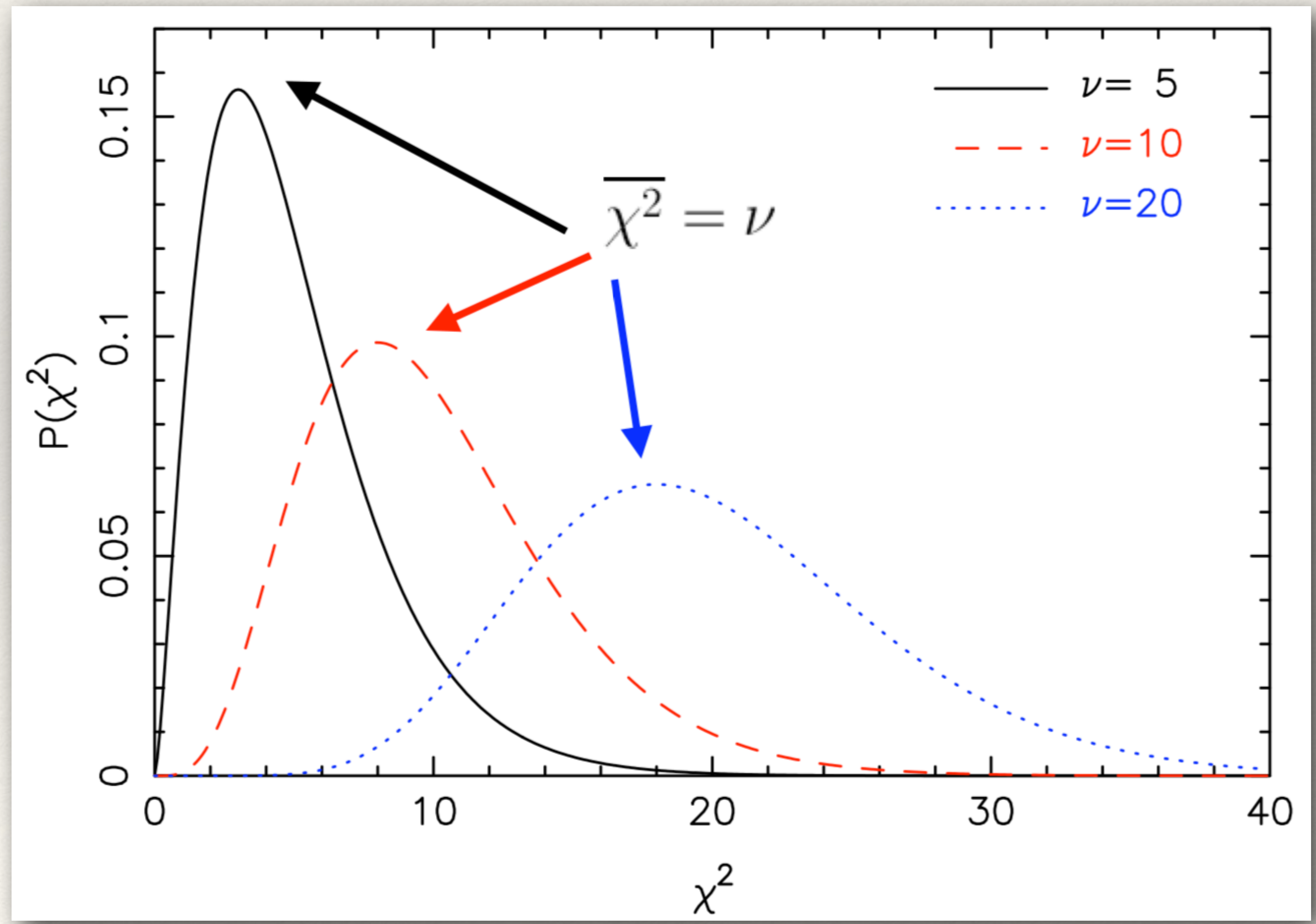
χ^2 probability distribution

$$P(\chi^2) \propto (\chi^2)^{\frac{\nu-2}{2}} \exp(\chi^2/2)$$

- This is the probability distribution that the model is correct (note: assumes the variables are Gaussian-distributed);
- ν is number of degrees of freedom;
- If the model has no free parameters then $\nu=N$;
- If we are fitting a model with p free parameters, we can "force the model to exactly agree with p data points" and $\nu=N-p$.

χ^2 probability distribution

$$P(\chi^2) \propto (\chi^2)^{\frac{\nu-2}{2}} \exp(\chi^2/2)$$



χ^2 probability distribution

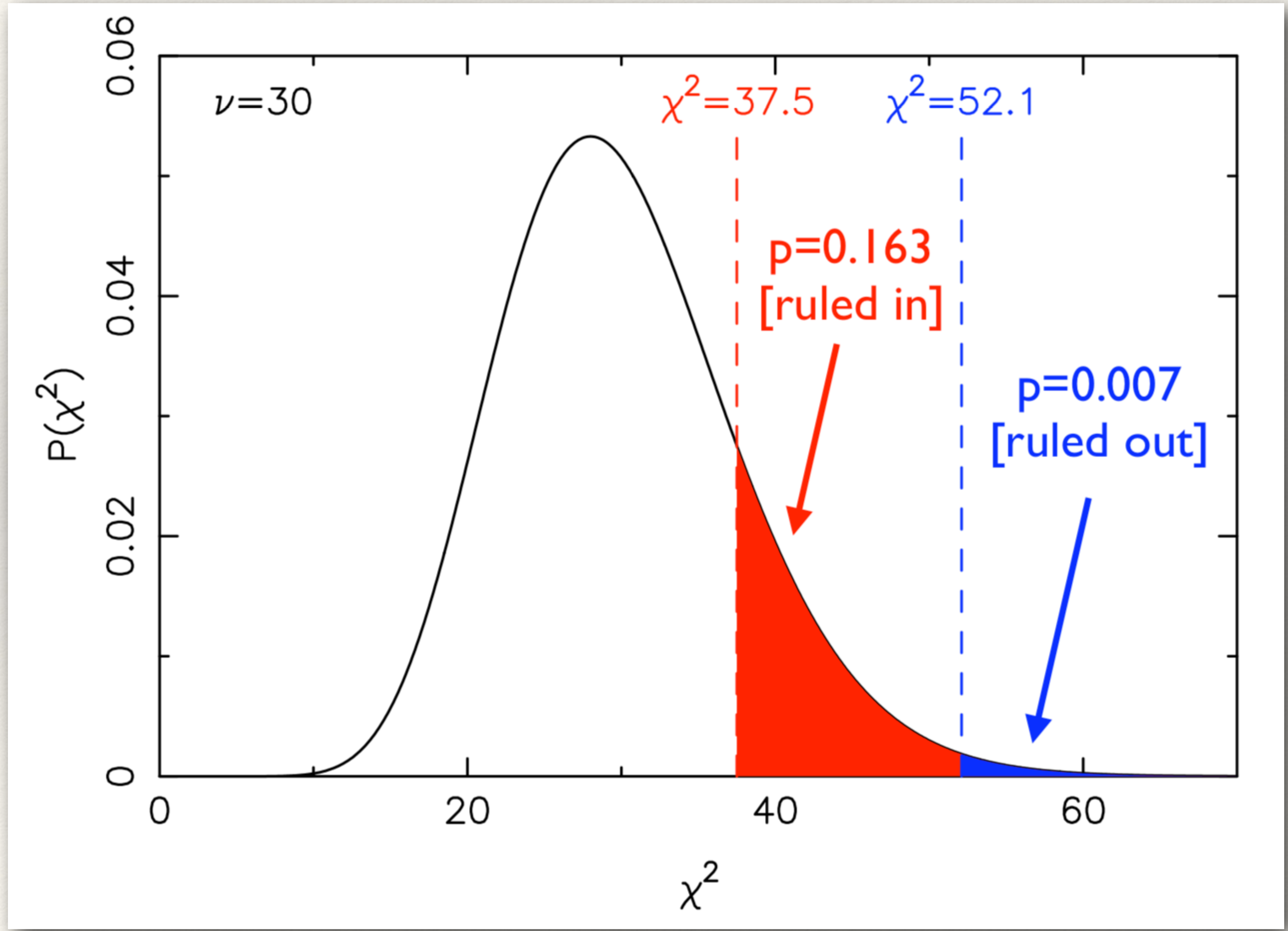
$$P(\chi^2) \propto (\chi^2)^{\frac{\nu-2}{2}} \exp(\chi^2/2)$$

- Mean : $\overline{\chi^2} = \nu = N - p$
- Variance : $\mathrm{Var}(\chi^2) = 2\nu$
- If the model is correct we expect : $\chi^2 \sim \pm\sqrt{2\nu}$
- Makes (even intuitively) sense because each data point should lie about ~1- σ from the model and hence contribute 1.0 to the chi-squared statistic.

Hypothesis testing with χ^2

- We ask the question : if the model (our hypothesis) is correct, what is the probability that this value of χ^2, or a larger one, could arise by chance?
- this probability is called the *p-value* and may be calculated from the χ^2 distribution;
- If the *p-value* is **not low**, then the data are consistent with being drawn from the model, which is "ruled in";
- If the *p-value* is low, then the data are not consistent with being drawn from the model. The model is "ruled out".

Example



$\nu=30$
$\chi^2=37.5$
$\chi^2=52.1$
p=0.163
[ruled in]
p=0.007
[ruled out]
$P(\chi^2)$
0
0.02
0.04
0.06
0
20
40
60
χ^2

Hypothesis testing with χ^2

- Note that we are assuming the errors in the data are Gaussian and robust;
- if the errors have been under-estimated then an improbably high value of χ^2 can be obtained;
- if the errors have been over-estimated then an improbably low value of χ^2 can be obtained.

Uncertainties derivation

- To calculate uncertainties on the parameters, we will use the “bootstrapping technique:
    1. we will create a random set of observed datapoints, drawn from a gaussian (normal) distribution;
    2. we will fit this new points with the mod-bb model;
    3. doing this for several (hundreds or thousands of) times will allow us to calculate errorbars.

Practically…

Copy the `mod-bb` macro onto another file.

Then…

```
nrun = 1000
chi2_arr = np.zeros(nrun)
dmass_arr = np.zeros(nrun)
dtemp_arr = np.zeros(nrun)
dbeta_arr = np.zeros(nrun)
for _i in range(nrun):
    fluxes = np.random.normal(ofluxes, errors)
    chisq = minimize(chi2,x0,bounds=bounds,method='TNC')
    chi2_arr[_i] = chisq.fun/2.
    dmass_arr[_i] = chisq.x[0]
    dtemp_arr[_i] = chisq.x[1]
    dbeta_arr[_i] = chisq.x[2]
```

Practically…

```
deltachi2 = 9.21 + np.min(chi2_arr)    #at 99% confidence level
accept_chi2 = []
accept_mass = []
accept_temp = []
accept_beta = []
for i in range(nrun):
   if chi2_arr[_i] <= deltachi2:
      accept_chi2.append(chi2_arr[i])
      accept_mass.append(dmass_arr[i])
      accept_temp.append(dtemp_arr[i])
      accept_beta.append(dbeta_arr[i])
```

Practically…

```
mass_best = np.median(accept_mass)
mass_sigm = np.std(accept_mass)
temp_best = np.median(accept_temp)
temp_sigm = np.std(accept_temp)
beta_best = np.median(accept_beta)
beta_sigm = np.std(accept_beta)
nbins = nrun/50
print "Best fit dust mass: ",mass_best," ± ",mass_sigm
print "Best fit dust temperature: ",temp_best," ± ",temp_sigm
print "Best fit dust emissivity: ",beta_best," ± ",beta_sigm
print "Mass with mean value: ",np.mean(mass_fin)
print "Temp with mean value: ",np.mean(temp_fin)
print "Beta with mean value: ",np.mean(beta_fin)
```

Plotting

- We will make histogram plots to show how recurrent are the values of the 3 parameters in our boot-strapping simulation;
- in a fourth plot, we can show the best fit model.

Spatially-Resolved analysis

- We now aim at deriving the properties of the dust as a function of position in the galaxy;
- to do so, we need to extract, for each wavelength, the flux at each position (pixel) in the galaxy;
- to get a physical analysis, the pixels at the various wavelengths, need to have the exact same size and position;
- then, we will reproduce these fluxes with a modified BB model.

Before doing so, we must take into account an important detail:

Image Smoothing

- Images at different wavelengths have different spatial resolution, hence, different PSFs;
- the pixel size might also be, in general, different for each wavelength;
- for these reasons, the pixels are sampling different regions of the galaxy at different wavelengths;
- we need to have images at the SAME SPATIAL RESOLUTION!

Spatially-Resolved analysis

- Creating images at different wavelengths sampling the same regions is done in 2 steps:
    1. smooth the images such that they have the same spatial resolution (beam) of the image at the poorest resolution;
    2. re-grid (if needed) the pixels of the image such that they are of the same size and with the very same coordinates.

Image Smoothing (Convolution)

- The basic idea for convolution is that you define a filter/kernel, as an example, a 3 x 3 matrix;
- the values of the 9 elements of this kernel are used as a sliding filter over the image you want to process;
- then, the kernel filter is applied to each pixel of your image;
- the kernel, in order to define a new value for each pixel of your image, takes into account the values of the (8) neighboring elements;
- each element of the kernel is multiplied with the corresponding image pixel value, and then those products are summed;
- this sum becomes the new value for the pixel currently being processed.

Practically...

```python
import numpy as np
from astropy.io import fits
from FITS_tools.hcongrid import hcongrid
from astropy.convolution import Gaussian2DKernel
from scipy.signal import convolve as scipy_convolve

wimg = input("Which image are you convolving? (1-4) ")
psf = [5.5,6.7,18.,24.,36.]
kk = 2.*np.pi/(360.*3600.)
fw2sig = 1./(2.*np.sqrt(2.*np.log(2.)))
```

Practically…

```
# pacs @ 70
if wimg == 1:
    img = 'M51_pacs70.fits'
    outfile = 'M51_pacs70_convolved.fits'
    wind = 0
    print ''
    print 'Processing pacs data at 70 microns'
    print ''
# pacs @ 160
if wimg == 2:
    img = 'M51_pacs160.fits'
    outfile = 'M51_pacs160_convolved.fits'
    wind = 0
    print ''
    print 'Processing pacs data at 160 microns'
    print ''
```

Practically…

```
img500 = 'M51_spire500.fits'
hdulist500 = fits.open(img500)
header500 = hdulist500[1].header
pixsz500 = np.abs(header500["CDELT1"])*3600.
hdulist = fits.open(img)
header = hdulist[1].header
flux = hdulist[1].data
nx = header["NAXIS1"]
ny = header["NAXIS2"]
flx_zero = flux.copy()
flx_zero[np.isnan(flux)] = 0.0
pixsz = np.abs(header["CDELT1"])*3600.
```

Practically…

```
if wind < 2:
    flx_sb = flx_zero/(1e6*(kk*pixsz)**2)
else:
    flx_sb = flx_zero
sigma = np.sqrt(psf[4]**2-psf[wind]**2)/pixsz*fw2sig
kernel = Gaussian2DKernel(stddev=sigma,x_size=20,y_size=20)
convolved = convolve(flx_sb, kernel, mode='same')
flxmap = hcongrid(convolved,header,header500)
flx_reg = flxmap*1e6*kk**2*pixsz500**2
hdu_s = fits.PrimaryHDU(flx_reg,header=header500)
hdu_s.writeto(outfile)
```

Checks

- Verify that the WCS are ok (where is the galaxy in the images at various wavelength?);
- control that the pixels “coincide”;
- check that the total flux is roughly what you have measured in the original images.

Spatially-resolved analysis

- Now we have 5 images at the “same resolution” and whose pixels are spatially coincident;
- this means we can construct a “spatially-resolved” SED for each pixel position, using as flux the pixel’s values;
- using this, we can —through a mod-BB modeling— derive the spatially resolved dust properties.

Practically...

```
import numpy as np
from astropy.io import fits
import pyregion
from scipy.optimize import minimize

Mpc2cm = 3.08568e24                         # Mpc to cm
h_p = 6.62607e-34                           # m^2 kg s^-1
k_b = 1.3806485e-23                         # m^2 kg s^-2 K^-1
c_l = 2.99792458e8                          # m s^-1
msol = 1.988e30                             # kg
dist = ???                                  # Mpc
knu0 = 0.192                                # m^2 kg^-1
nu0 = c_l*1e6/350.                          # Hz
const = msol/(dist*Mpc2cm)**2
```

Practically…

```
def B_nu(tt):
    a = (2.*h_p*nu**3)/c_l**2
    arg = h_p*nu/(k_b*tt)
    return a * 1./(np.exp(arg)-1.)          # in W sr^-1 m^-2 Hz^-1

def ffunc_nu(md,tt,bb):
    knu = knu0*(nu/nu0)**bb
    mbb_f = md*const*knu*B_nu(tt)           # in W cm^-2 Hz^-1
    mbb_erg = 1e7*mbb_f                     # in erg s^-1 cm^-2 Hz^-1
    return mbb_erg*1e23                     # in Jy

def chi2(params):
    md, tt, bb = params
    return np.sum(((ffunc_nu(md,tt,bb)-oflux)*check/error)**2)/
          (sum(check)-3.)
```

Practically…

```
wavel = [70., 160., 250., 350., 500.]
nu = np.zeros(5)
nu = c_l*1e6*np.divide(1.,wavel)
img1 = 'M51_pacs70_convolved.fits'
hdulist1 = fits.open(img1)
flux1 = hdulist1[0].data
img2 = 'M51_pacs160_convolved.fits'
hdulist2 = fits.open(img2)
flux2 = hdulist2[0].data
img3 = 'M51_spire250_convolved.fits'
hdulist3 = fits.open(img3)
flux3 = hdulist3[0].data
. . .
. . .
```

Practically…

```
nx = header['NAXIS1']
ny = header['NAXIS2']
bg = np.zeros(5)
sig = np.zeros(5)
size = (ny, nx)
# PACS 70
reg = 'pacs_lowres.reg'
source = pyregion.open(reg)
nsky = len(source) - 1
sky_ap = np.zeros(nsky)
sky_pix = []
pix_ap = np.zeros(nsky)
for k in range(nsky):
    sky = pyregion.open(reg)
    del sky[0:k+1]
    del sky[1:nsky]
    bmask = sky.get_mask(shape=size)
    sflx = 0.0
    pix = 0
    for i in range(nx):
        for n in range(ny):
            if bmask[n,i] == True:
                sflx = sflx + flux1[n,i]
                sky_pix.append(flux1[n,i])
                pix = pix + 1
    sky_ap[k] = sflx
    pix_ap[k] = pix
sig[0] = np.std(sky_pix)
bg[0] =  sum(sky_ap)/sum(pix_ap)
```

Practically…

```
bounds = [(1e1,1e6), (5.,60.), (0.5, 3.0)]
x0 = [1e2, 20., 2.0]
results = np.zeros([4,ny,nx])
for yyy in range(ny):
    for xxx in range(nx):
        oflux = np.zeros(5)
        error = np.zeros(5)
        oflux[0] = flux1[yyy,xxx]-bg[0]
        oflux[1] = flux2[yyy,xxx]-bg[1]
        oflux[2] = flux3[yyy,xxx]-bg[2]
        oflux[3] = flux4[yyy,xxx]-bg[3]
        oflux[4] = flux5[yyy,xxx]-bg[4]
        error[0:2]=oflux[0:2]*0.05
        error[2:5]=oflux[2:5]*0.07
```

Practically…

```python
check = np.zeros(5)
for nnn in range(5):
    if oflux[nnn] >= 3.*sig[nnn]:
        check[nnn] = 1
if sum(check) > 3:
    print 'Fitting pixel ',xxx,yyy
    chisq = minimize(chi2,x0,bounds=bounds,method='TNC')
    results[0,yyy,xxx] = chisq.fun
    results[1,yyy,xxx] = chisq.x[0]
    results[2,yyy,xxx] = chisq.x[1]
    results[3,yyy,xxx] = chisq.x[2]
    print 'Reduced chi2=',chisq.fun
else:
    print 'Skipping...'
    results[:,yyy,xxx] = -999.
```

Practically…

```
outfile = "results_map.fits"
newheader = header.copy()
del newheader[5:24]
del newheader[14:]
newheader['NAXIS'] = 3
newheader.insert(5, ('NAXIS3', 4))
newheader['PLANE0'] = 'chi2'
newheader['PLANE1'] = 'Dust mass [Msol]'
newheader['PLANE2'] = 'Dust temperature [K]'
newheader['PLANE3'] = 'Emissivity index'
hdu_s = fits.PrimaryHDU(results,header=newheader)
hdu_s.writeto(outfile)
```