Clusters

December 6, 2018

In [3]:
```
import astropy
import matplotlib.pyplot as plt
from astropy.io import fits
import numpy as np
from scipy.stats import norm
import matplotlib.mlab as mlab
import glob
from scipy import ndimage
from scipy.optimize import curve_fit
from astropy.modeling.models import Linear1D

hdu_list = fits.open('atlas_pleione.fit')
hdu_list.info()
image = hdu_list[0].data

print('mean of image:', np.mean(image))
print('median of image:', np.median(image))
print('standard deviation of image along x axis:', np.std(image[0]))
print('standard deviation of image along y axis:', np.std(image[1]))
print('standard deviation of image:', np.std(image))
```

```
Filename: atlas_pleione.fit
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU      38   (1530, 1020)   int16 (rescales to uint16)
mean of image: 941.1868236575676
median of image: 939.0
standard deviation of image along x axis: 6.977445738101556
standard deviation of image along y axis: 6.808289049229859
standard deviation of image: 183.20296936007793
```

In [4]:
```
plt.figure(figsize=(10,10))
# Define fit function.
def gauss(x, A, mu, sigma):
    return (A * np.exp(-1.0 * (x - mu)**2 / (2 * sigma**2)))

# Histogram.
bins = np.linspace(910, 980, 70)
```

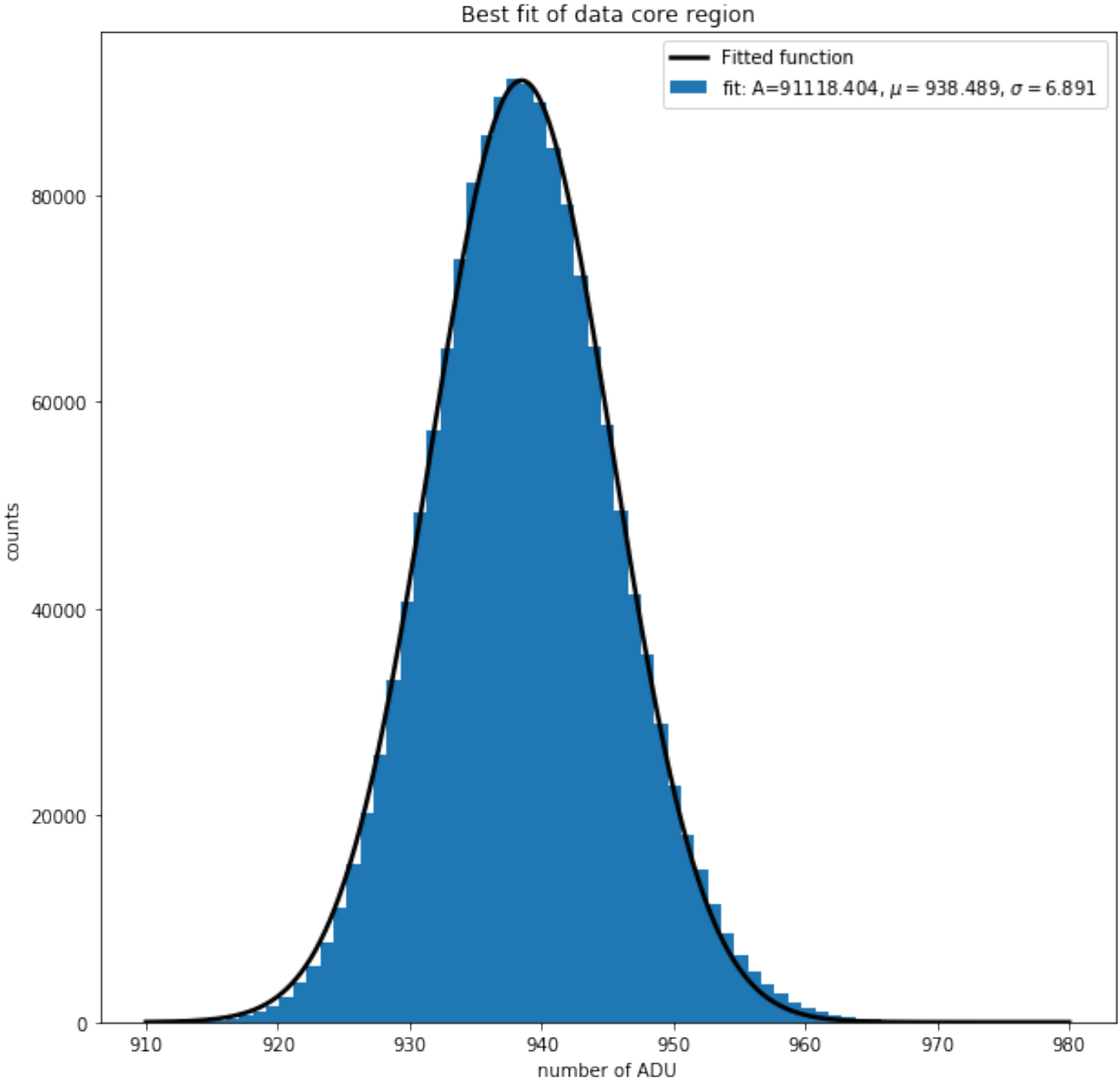
```
    data_entries, bins = np.histogram(image, bins=bins)
    binscenters = np.array([0.5 * (bins[i] + bins[i+1]) for i in range(len(bins)-1)])

    # Fit the function to the histogram data.
    popt, pcov = curve_fit(gauss, xdata=binscenters, ydata=data_entries, p0=[200, 930, 6])
    print(popt)

    # Generate enough x values to make the curves look smooth.
    xspace = np.linspace(910, 980, 10000)

    # Plot the histogram and the fitted function.
    plt.bar(binscenters, data_entries, width=bins[1] - bins[0], label='fit: A=%5.3f, $\mu=%5.3f$, $\si
    plt.plot(xspace, gauss(xspace, *popt), color='k', linewidth=2.5, label=r'Fitted function')
    plt.title("Best fit of data core region")
    plt.xlabel("number of ADU")
    plt.ylabel("counts")
    plt.legend(loc='best')
    plt.show()
```

```
[9.11184040e+04 9.38488963e+02 6.89113580e+00]
```




```
In [5]: n_t = np.median(image) + (5 * np.std(image[1]))
        print(n_t)
```

```
973.0414452461494
```

In [6]:
```
n_tgauss = popt[1] + (5 * popt[2])
print(n_tgauss)
```

```
972.9446417954057
```

In [24]:
```
black=0
white=2**(16)

# floodfill algorithm
def Cluster(image):
    xsize,ysize=image.shape
    b_ij = np.std(image)
    # set threshold to be standard deviation of the whole image minus a multiple of the sigma's c
    # if pixel value is above n_t it is assigned 1, if below, assigned 0
    n_t = b_ij - (5*popt[1])
    list_of_clusters = [] #list of clusters of pixels (stars)
    for x in range(xsize):
        for y in range(ysize):
            single_cluster=[] #list of pixels in star
            if n_t[x,y]==1: #if pixel value is assigned 1 then fill function is called to start f
                fill(n_t,xsize,ysize,x,y,single_cluster) #call fill to begin counting
                list_of_clusters.append(single_cluster) #add each cluster found to list of all cl
                number_of_stars=len(list_of_clusters) #number of stars in file
    return list_of_clusters
    print(list_of_clusters)


#recursive algorithm: stack
def fill(image,xsize,ysize,x,y,clust):
    #print (x,y,image[x,y])
    if image[x,y] == 0:
        return #if pixel value is zero then ignore
    else:
        image[x,y]=0 #if pixel value isn't zero then we set it to zero
        clust.append([x,y])
    if x>0: fill(image,xsize,ysize,x-1,y,clust)
    if x<(xsize-1): fill(image,xsize,ysize,x+1,y,clust)
    if y>0: fill(image,xsize,ysize,x,y-1,clust)
    if y<(ysize-1): fill(image,xsize,ysize,x,y+1,clust)
```