Sigma Background estimation

December 11, 2018

In [5]:
```
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()
```

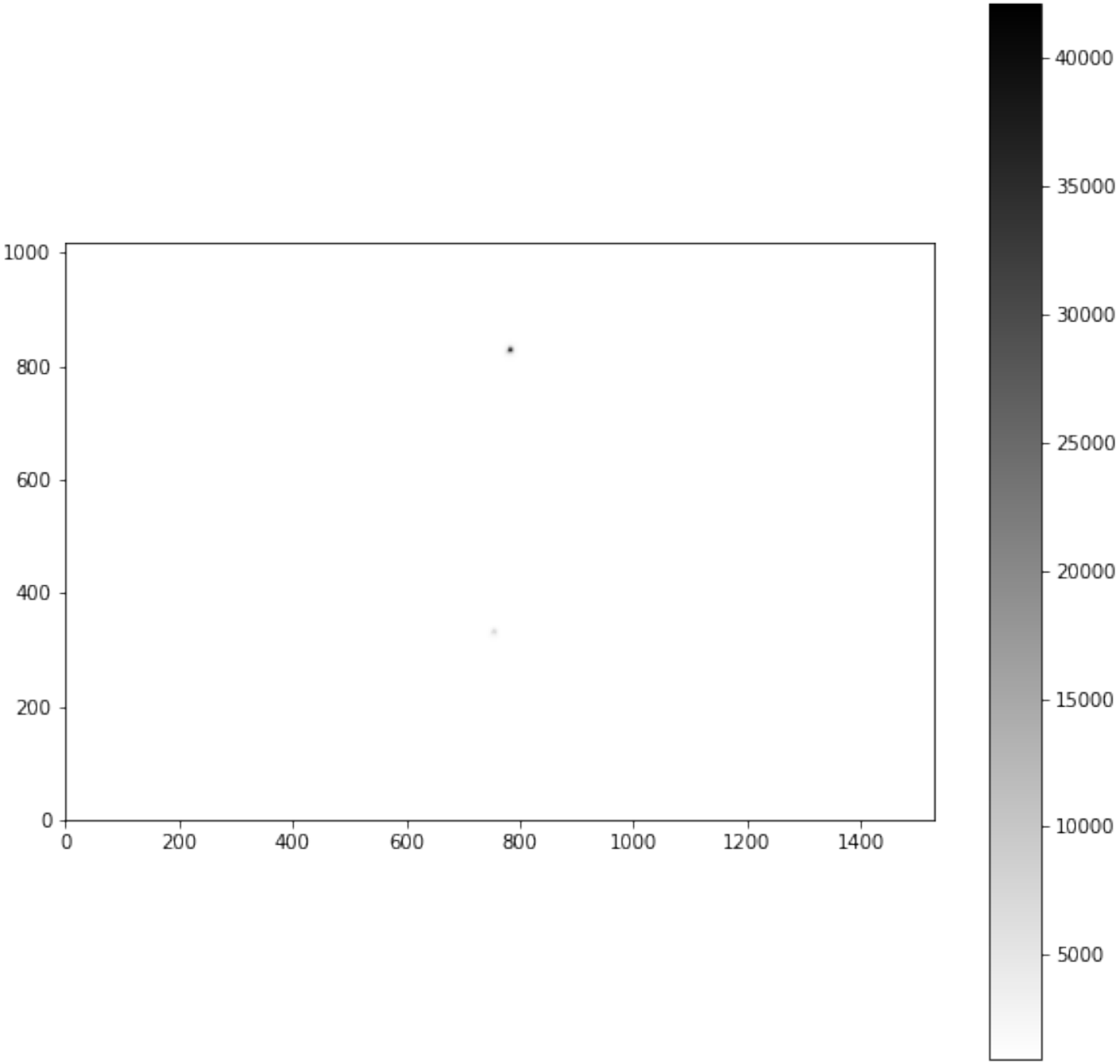
```
Filename: atlas_pleione.fit
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU      38   (1530, 1020)   int16 (rescales to uint16)
```

In [6]:
```
image = hdu_list[0].data #* 2.3 #ADU multiplyed by the gain gives number of p.e.
print(image)
print(image.shape)
#rows 1020, columns 1530
```

```
[[949 941 937 ... 960 945 960]
 [937 936 944 ... 952 949 948]
 [938 944 930 ... 950 948 952]
 ...
 [953 947 936 ... 954 954 961]
 [935 942 948 ... 967 952 947]
 [945 930 935 ... 953 964 958]]
(1020, 1530)
```

In [7]:
```
plt.figure(figsize=(10,10))
plt.imshow(image, cmap='gray_r', origin='lower')
plt.colorbar()
```

Out[7]: <matplotlib.colorbar.Colorbar at 0x1a140cb5f8>

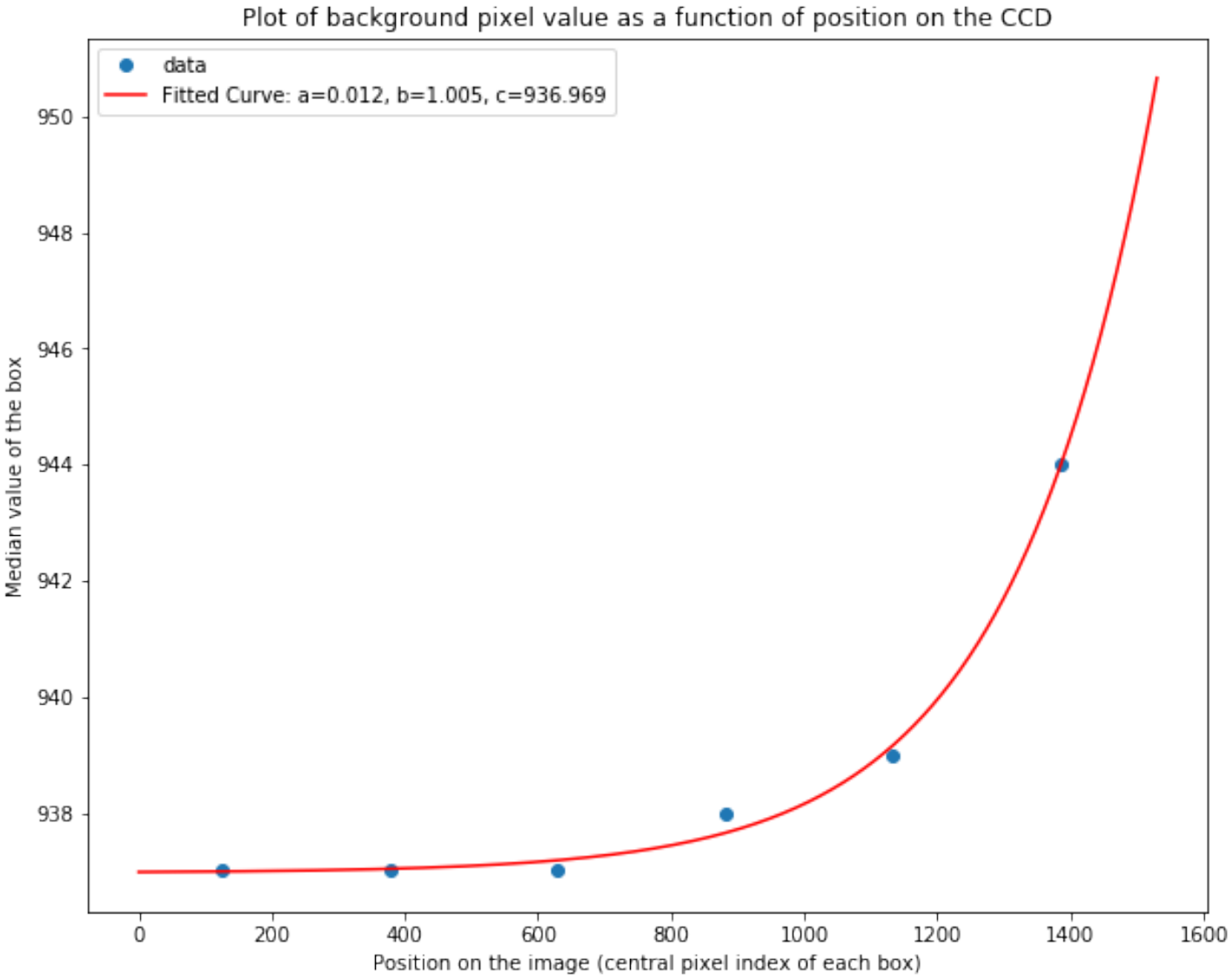


In [287]:
```
#row stays the same, column changes
box1 = image[0:251,0:251]
box2 = image[0:251,252:503]
box3 = image[0:251,504:755]
```

```
        box4 = image[0:251,756:1007]
        box5 = image[0:251,1008:1259]
        box6 = image[0:251,1260:1530]

        l = [np.median(box1), np.median(box2), np.median(box3), np.median(box4), np.median(box5), np.med
        print('median values', l)

        def exp(x, a, b, c):
            return a * (b**x) + c

        x = np.arange(125, 1530, 252)
        y = l
        xspace = np.linspace(0, 1530, 1000)

        popt, pcov = curve_fit(exp, x, y, p0=(0.5,1.0,900))

        plt.figure(figsize=(10,8))
        plt.plot(x, y, 'o', label='data')
        plt.plot(xspace, exp(xspace, *popt), 'r-', label='Fitted Curve: a=%5.3f, b=%5.3f, c=%5.3f' % tup
        plt.title('Plot of background pixel value as a function of position on the CCD')
        plt.ylabel('Median value of the box')
        plt.xlabel('Position on the image (central pixel index of each box)')
        plt.legend()
```

```
median values [937.0, 937.0, 937.0, 938.0, 939.0, 944.0]
```

```
Out[287]: <matplotlib.legend.Legend at 0x1a2029ec18>
```

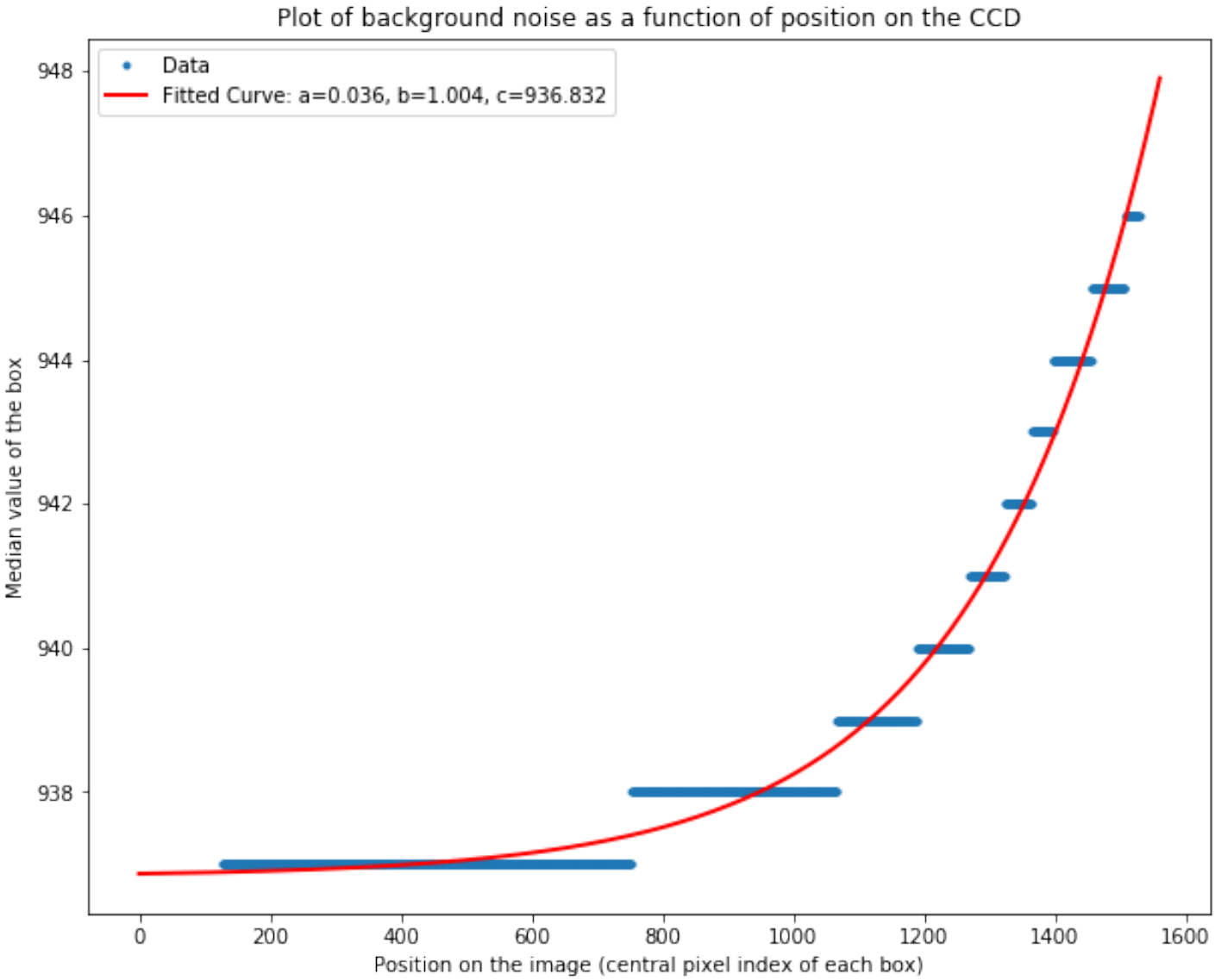


In [288]:
```
xdir = image[0:255]
ydir = image.shape[1]

boxes = []  #i now have a list (of length 1530) of boxes with dimension 255x255
boxes_median = [] #i now have a list (of length 1530) of medians for each box found previously
central_pixels = np.arange(127, 1530, 1) #i want the central pixel number, length is 1403


# loop over the image, box by box, moving by one pixel at time in the x direction
for j in range(0, ydir):
    box = image[0:255, j:j+255]
    boxes.append(box)
```

```
            medians = np.median(box)
            boxes_median.append(medians)

        #discard those arrays that have a number of columns less than 255 by removing values greater tha
        boxes = boxes[0:1403]
        boxes_median = boxes_median[0:1403]


        #best fit function to the data
        def exp(x, a, b, c):
            return a * (b**x) + c

        x = central_pixels
        y = boxes_median
        xspace = np.linspace(0, 1560, 1000)

        popt, pcov = curve_fit(exp, x, y, p0=(0.5,1.0,900))
        print('parameters:',popt)
        print('pcov:',pcov)


        plt.figure(figsize=(10,8))
        plt.plot(x, y, '.', label='Data')
        plt.plot(xspace, exp(xspace, *popt), 'r-',linewidth=2, label='Fitted Curve: a=%5.3f, b=%5.3f, c=
        plt.title('Plot of background noise as a function of position on the CCD')
        plt.ylabel('Median value of the box')
        plt.xlabel('Position on the image (central pixel index of each box)')
        plt.legend()
```

```
parameters: [3.59770919e-02 1.00367928e+00 9.36832248e+02]
pcov: [[ 2.26441439e-06 -4.26177916e-08 -1.75842137e-05]
 [-4.26177916e-08  8.07052229e-10  3.19011894e-07]
 [-1.75842137e-05  3.19011894e-07  2.28225965e-04]]
```

```
Out[288]: <matplotlib.legend.Legend at 0x1a2030b6d8>
```

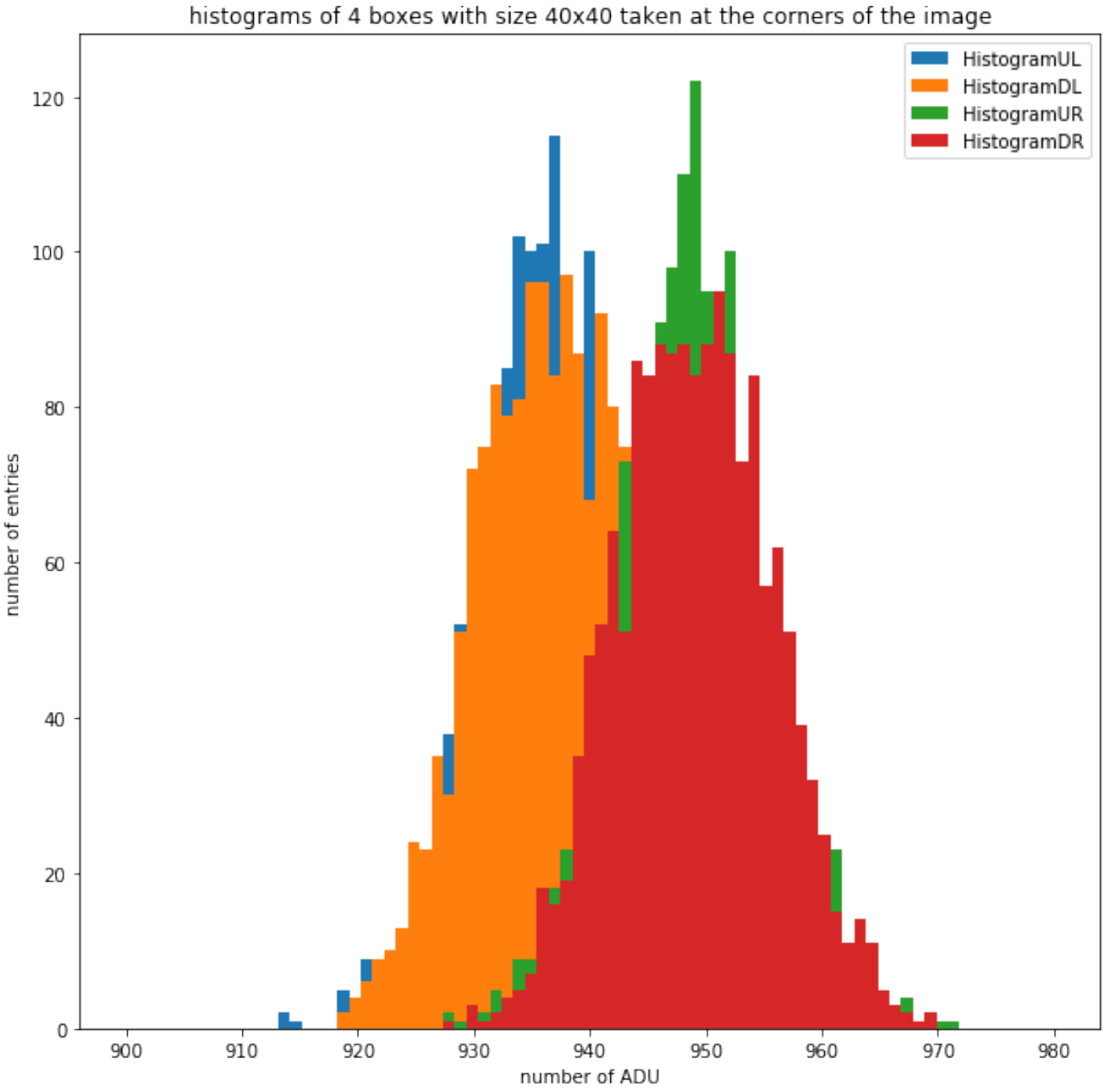


In [276]:
```
#take 4 boxes at the corners of size 40by40
#first argument: row, second: column
boxUL = image[0:40,0:40]
print(boxUL.shape)
print('median UL:', np.median(boxUL))
boxDL = image[980:1020,0:40]
print(boxDL.shape)
print('median DL:', np.median(boxDL))
boxUR = image[0:40,1490:1530]
print(boxUR.shape)
print('median UR:', np.median(boxUR))
boxDR = image[980:1020,1490:1530]
```

```
        print(boxDR.shape)
        print('median DR:', np.median(boxDR))

        l = [np.median(boxUL), np.median(boxDL), np.median(boxUR), np.median(boxDR)]
        print('\n', 'average of pixel value in boxes is the sigma of background:', np.mean(l))
```

```
(40, 40)
median UL: 937.0
(40, 40)
median DL: 937.0
(40, 40)
median UR: 949.0
(40, 40)
median DR: 949.0

 average of pixel value in boxes is the sigma of background: 943.0
```

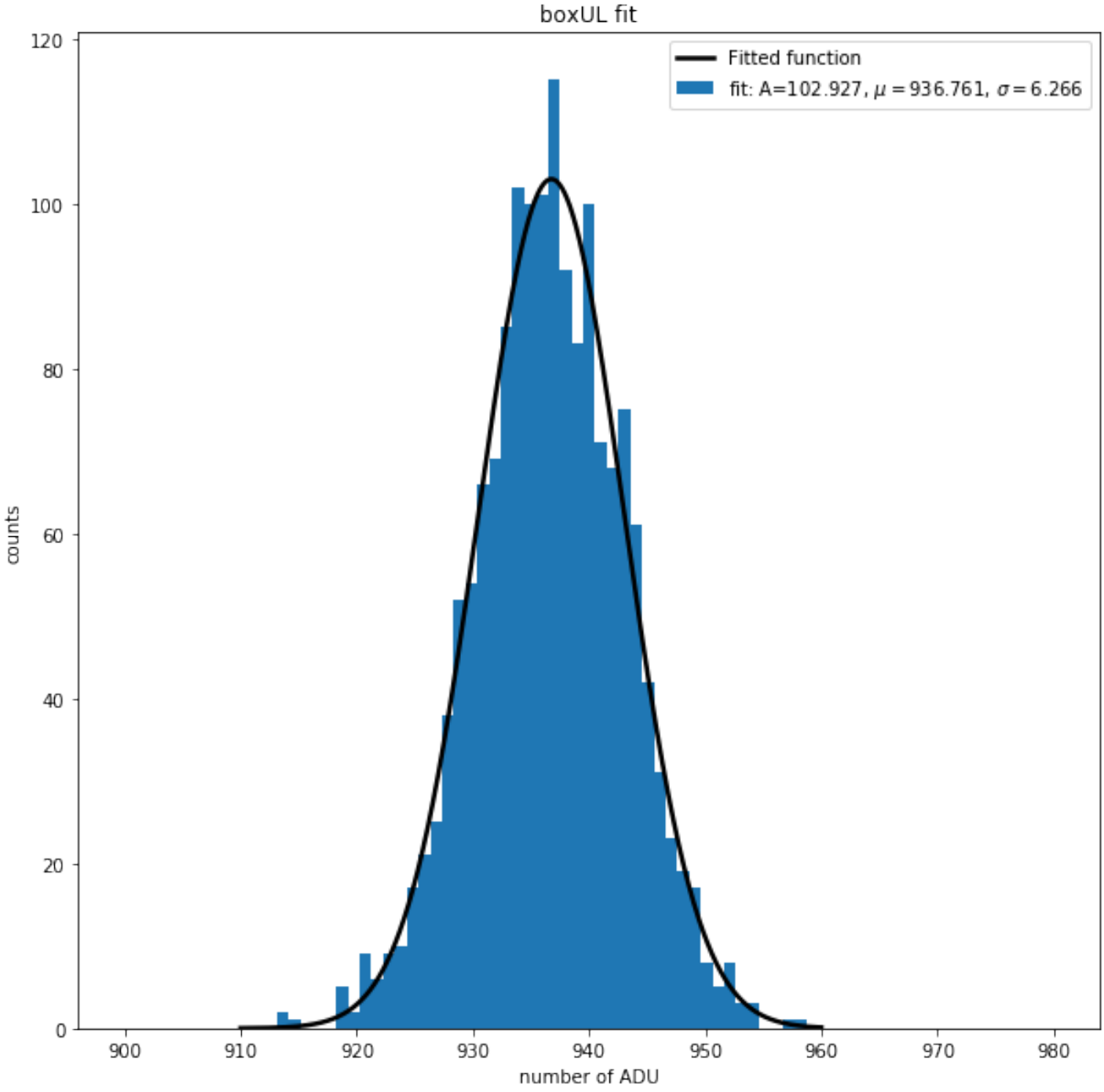
```
In [217]: plt.figure(figsize=(10,10))
          bins = np.linspace(900, 980, 80)
          plt.hist(boxUL.flatten(), bins=bins, label='HistogramUL')
          plt.hist(boxDL.flatten(), bins=bins, label='HistogramDL')
          plt.hist(boxUR.flatten(), bins=bins, label='HistogramUR')
          plt.hist(boxDR.flatten(), bins=bins, label='HistogramDR')

          plt.title('histograms of 4 boxes with size 40x40 taken at the corners of the image')
          plt.xlabel('number of ADU')
          plt.ylabel('number of entries')
          plt.legend()

Out[217]: <matplotlib.legend.Legend at 0x1a2a2c9c50>
```



In [218]:
```
plt.figure(figsize=(10,10))

    def fit_function(x, A, mu, sigma):
        return (A * np.exp(-1.0 * (x - mu)**2 / (2 * sigma**2)))
```

```
# Histograms
hdu_list = fits.open('atlas_pleione.fit')
imgdata = hdu_list[0].data

boxUL = image_data[0:40,0:40]

data = boxUL.flatten() #return 1D array
bins = np.linspace(900, 980, 80)

data_entries, bins = np.histogram(data, 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(fit_function, 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, 960, 10000)

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

poptUL = tuple(popt)
```

```
[102.92667463 936.76050701   6.26587862]
```

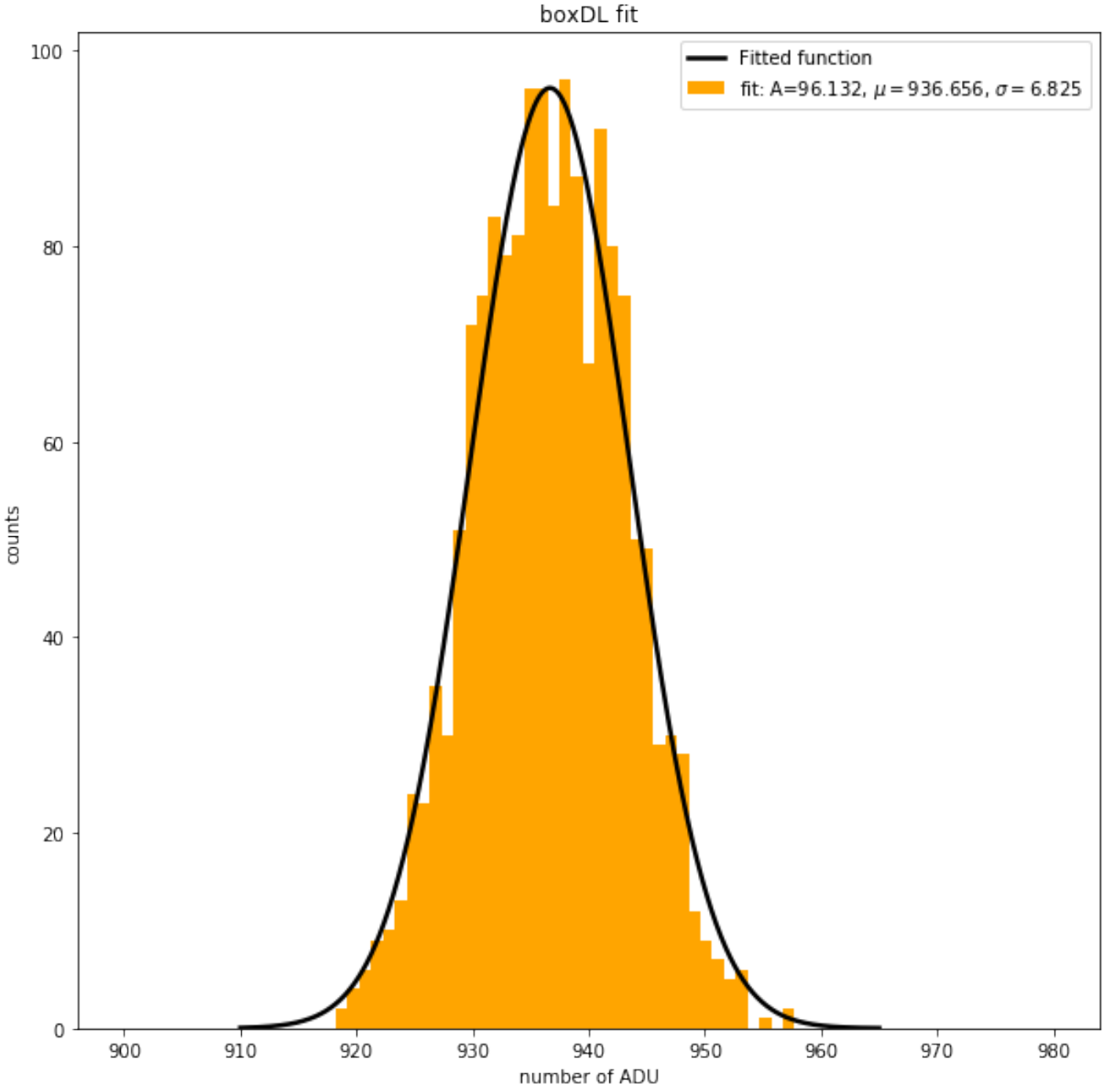


In [219]: plt.figure(figsize=(10,10))

```
def fit_function(x, A, mu, sigma):
    return (A * np.exp(-1.0 * (x - mu)**2 / (2 * sigma**2)))
```

```
hdu_list = fits.open('atlas_pleione.fit')
imgdata = hdu_list[0].data

boxDL = image_data[980:1020,0:40]

data = boxDL.flatten() #return 1D array

bins = np.linspace(900, 980, 80)

data_entries, bins = np.histogram(data, 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(fit_function, xdata=binscenters, ydata=data_entries, p0=[175, 930, 6])
print(popt)

# Generate enough x values to make the curves look smooth.
xspace = np.linspace(910, 965, 10000)

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

poptDL = tuple(popt)
```

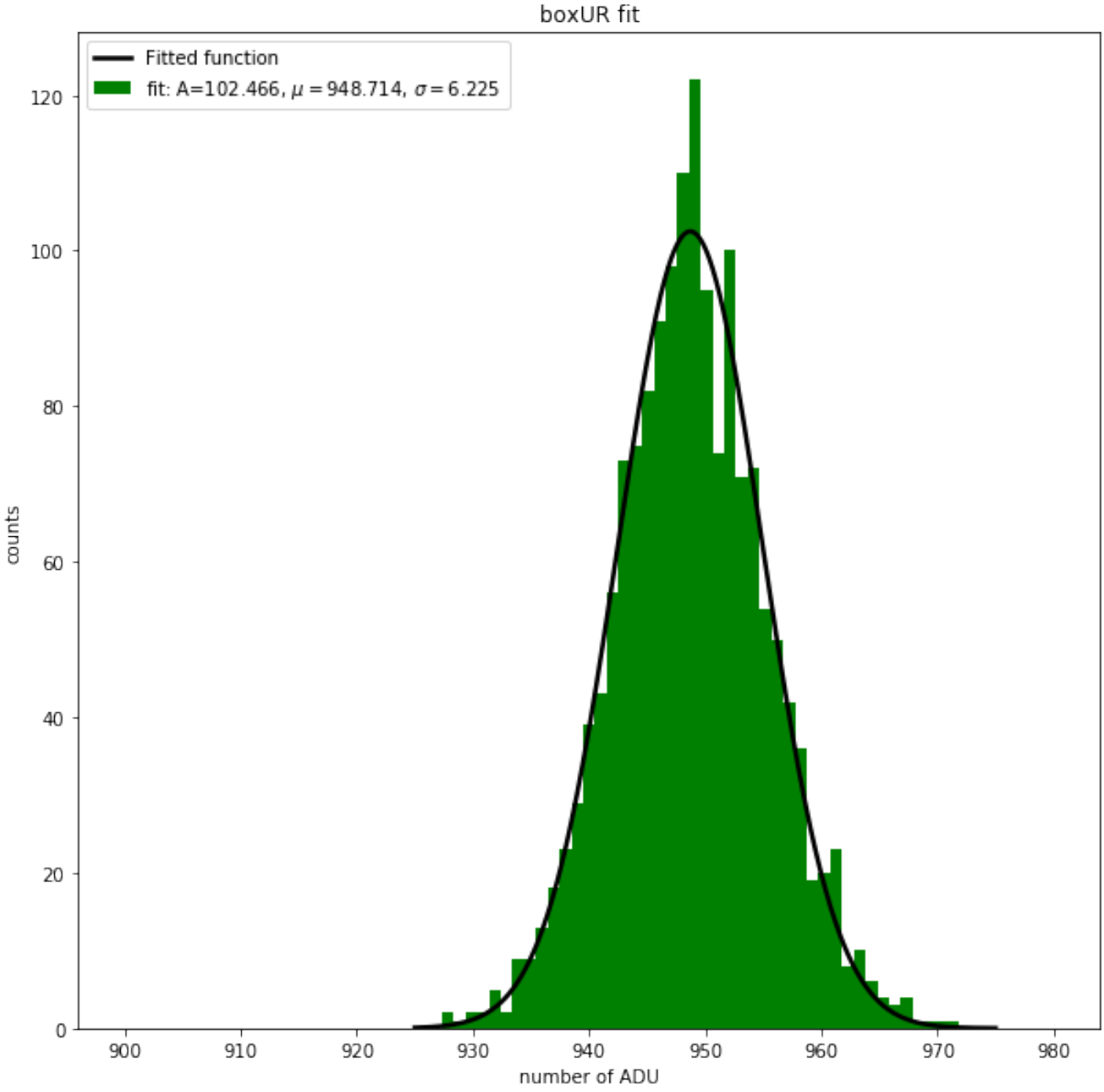
```
[ 96.13220564 936.65644429   6.82539114]
```



In [220]:
```
plt.figure(figsize=(10,10))

def fit_function(x, A, mu, sigma):
    return (A * np.exp(-1.0 * (x - mu)**2 / (2 * sigma**2)))
```

```
hdu_list = fits.open('atlas_pleione.fit')
imgdata = hdu_list[0].data

boxUR = image_data[0:40,1490:1530]

data = boxUR.flatten() #return 1D array


bins = np.linspace(900, 980, 80)
data_entries, bins = np.histogram(data, 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(fit_function, xdata=binscenters, ydata=data_entries, p0=[200, 930, 6])
print(popt)

# Generate enough x values to make the curves look smooth.
xspace = np.linspace(925, 975, 10000)

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

poptUR = tuple(popt)
```

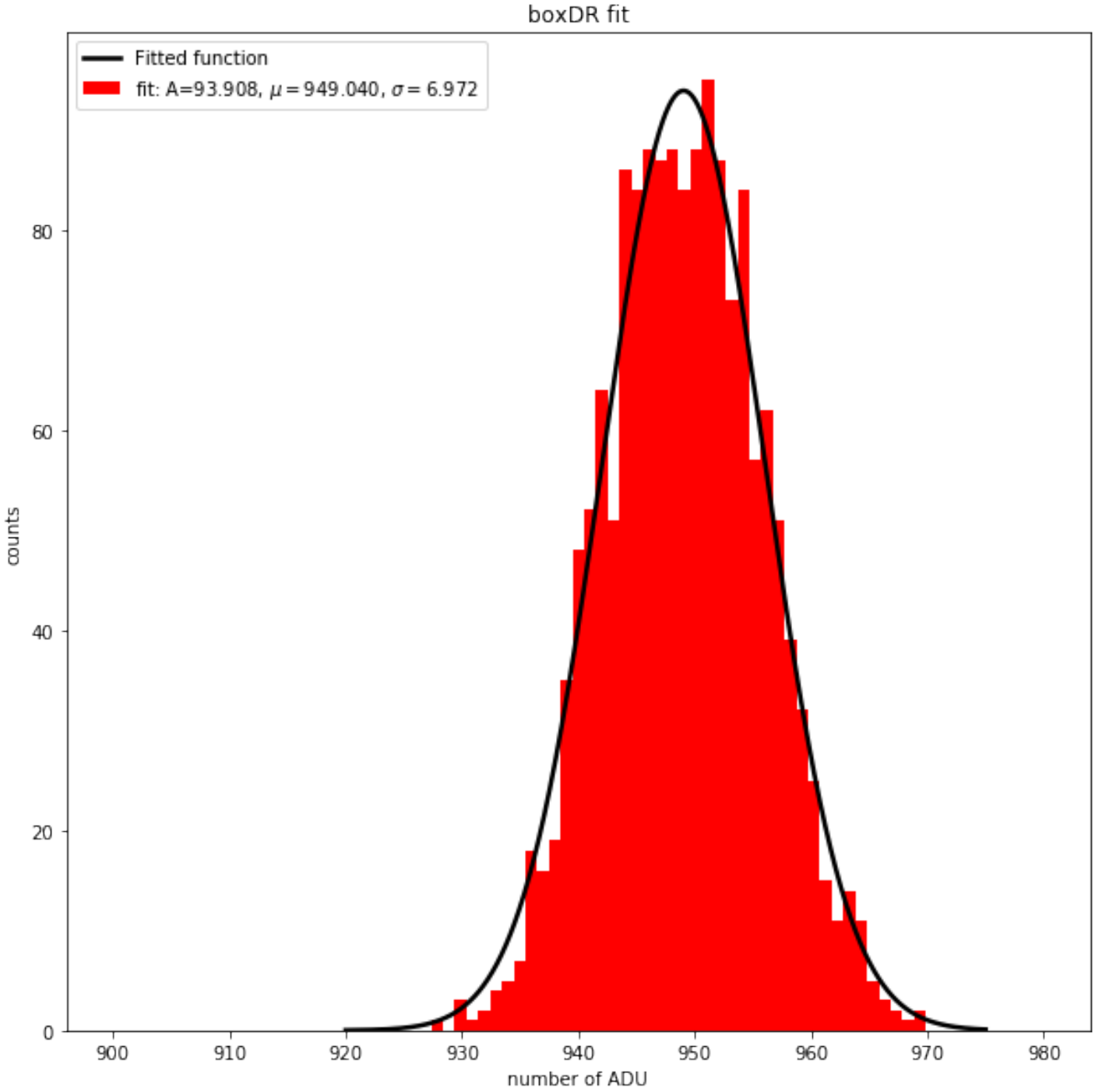
```
[102.46621147 948.71394493   6.22540316]
```




```
In [221]: plt.figure(figsize=(10,10))

          def fit_function(x, A, mu, sigma):
              return (A * np.exp(-1.0 * (x - mu)**2 / (2 * sigma**2)))
```

```
hdu_list = fits.open('atlas_pleione.fit')
imgdata = hdu_list[0].data

boxDR = image_data[980:1020,1490:1530]

data = boxDR.flatten() #return 1D array


bins = np.linspace(900, 980, 80)
data_entries, bins = np.histogram(data, 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(fit_function, xdata=binscenters, ydata=data_entries, p0=[200, 950, 6])
print(popt)

# Generate enough x values to make the curves look smooth.
xspace = np.linspace(920, 975, 10000)

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

poptDR = tuple(popt)
```

```
[ 93.90814027 949.040279     6.97239603]
```



In [222]:
```
#find the average sigma between the 4 boxes

list_of_sigmas = [poptUL[2], poptDL[2], poptUR[2], poptDR[2]]
average_of_sigmas = np.mean(list_of_sigmas)
print('sigma average of background:', average_of_sigmas)
```

```
        list_of_mus = [poptUL[1], poptDL[1], poptUR[1], poptDR[1]]
        average_of_mus = np.mean(list_of_mus)
        print('mu average of background:', average_of_mus)
```

```
sigma average of background: 6.572267235291228
mu average of background: 942.7927938062426
```