```
pip install astropy
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whe
Requirement already satisfied: astropy in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: pyerfa>=1.7.3 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/di
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: typing-extensions>=3.6.4 in /usr/local/lib/python
```

```
import numpy as np
import math
from scipy import integrate
import matplotlib.pyplot as plt
import matplotlib.colors as colors
import matplotlib.ticker as plticker
import random
import cv2
from astropy.io import fits

%matplotlib inline
```

Part One

```
filestart="d0221_0"
hdul=[] #empty list for the frames
for i in list(range(90,99)):
  if (i>=90) and (i<100):
  # write other loops for 1 through 8 chips
   for j in range(1,9):
     hdul = np.median(fits.open(filestart+str(0)+str(i)+".fits.gz")[j].data[:,:])
     print("The median bias level for chip",j,"in",filestart+str(0)+str(i)+".fits.g
```

```
The median bias level for chip 8 in d0221_0091.fits.gz  is  991.0
The median bias level for chip 1 in d0221_0092.fits.gz  is  993.0
The median bias level for chip 2 in d0221_0092.fits.gz  is  996.0
The median bias level for chip 3 in d0221_0092.fits.gz  is  1019.0
The median bias level for chip 4 in d0221_0092.fits.gz  is  997.0
The median bias level for chip 5 in d0221_0092.fits.gz  is  1006.0
The median bias level for chip 6 in d0221_0092.fits.gz  is  1000.0
The median bias level for chip 7 in d0221_0092.fits.gz  is  1000.0
The median bias level for chip 8 in d0221_0092.fits.gz  is  991.0
The median bias level for chip 1 in d0221_0093.fits.gz  is  993.0
The median bias level for chip 2 in d0221_0093.fits.gz  is  996.0
The median bias level for chip 3 in d0221_0093.fits.gz  is  1019.0
The median bias level for chip 4 in d0221_0093.fits.gz  is  997.0
The median bias level for chip 5 in d0221_0093.fits.gz  is  1006.0
The median bias level for chip 6 in d0221_0093.fits.gz  is  1000.0
The median bias level for chip 7 in d0221_0093.fits.gz  is  1000.0
The median bias level for chip 8 in d0221_0093.fits.gz  is  991.0
The median bias level for chip 1 in d0221_0094.fits.gz  is  994.0
The median bias level for chip 2 in d0221_0094.fits.gz  is  996.0
The median bias level for chip 3 in d0221_0094.fits.gz  is  1019.0
The median bias level for chip 4 in d0221_0094.fits.gz  is  997.0
The median bias level for chip 5 in d0221_0094.fits.gz  is  1006.0
The median bias level for chip 6 in d0221_0094.fits.gz  is  1000.0
The median bias level for chip 7 in d0221_0094.fits.gz  is  1000.0
The median bias level for chip 8 in d0221_0094.fits.gz  is  991.0
The median bias level for chip 1 in d0221_0095.fits.gz  is  993.0
The median bias level for chip 2 in d0221_0095.fits.gz  is  996.0
The median bias level for chip 3 in d0221_0095.fits.gz  is  1019.0
The median bias level for chip 4 in d0221_0095.fits.gz  is  997.0
The median bias level for chip 5 in d0221_0095.fits.gz  is  1006.0
The median bias level for chip 6 in d0221_0095.fits.gz  is  1000.0
The median bias level for chip 7 in d0221_0095.fits.gz  is  1000.0
The median bias level for chip 8 in d0221_0095.fits.gz  is  991.0
The median bias level for chip 1 in d0221_0096.fits.gz  is  994.0
The median bias level for chip 2 in d0221_0096.fits.gz  is  996.0
The median bias level for chip 3 in d0221_0096.fits.gz  is  1019.0
The median bias level for chip 4 in d0221_0096.fits.gz  is  997.0
The median bias level for chip 5 in d0221_0096.fits.gz  is  1006.0
The median bias level for chip 6 in d0221_0096.fits.gz  is  1000.0
The median bias level for chip 7 in d0221_0096.fits.gz  is  999.0
The median bias level for chip 8 in d0221_0096.fits.gz  is  991.0
The median bias level for chip 1 in d0221_0097.fits.gz  is  993.0
The median bias level for chip 2 in d0221_0097.fits.gz  is  996.0
The median bias level for chip 3 in d0221_0097.fits.gz  is  1019.0
The median bias level for chip 4 in d0221_0097.fits.gz  is  997.0
The median bias level for chip 5 in d0221_0097.fits.gz  is  1006.0
The median bias level for chip 6 in d0221_0097.fits.gz  is  1000.0
The median bias level for chip 7 in d0221_0097.fits.gz  is  999.0
The median bias level for chip 8 in d0221_0097.fits.gz  is  991.0
The median bias level for chip 1 in d0221_0098.fits.gz  is  994.0
The median bias level for chip 2 in d0221_0098.fits.gz  is  996.0
The median bias level for chip 3 in d0221_0098.fits.gz  is  1019.0
The median bias level for chip 4 in d0221_0098.fits.gz  is  997.0
The median bias level for chip 5 in d0221_0098.fits.gz  is  1006.0
The median bias level for chip 6 in d0221_0098.fits.gz  is  999.0


The median bias level for chip 7 in d0221_0098.fits.gz  is  999.0
The median bias level for chip 8 in d0221_0098.fits.gz  is  991.0
```

```python
# Chip 1
filestart="d0221_0"
arr1=np.zeros((4096,2140,9)) #empty list of 0 for the array size
for i in list(range(90,99)):
  arr1[:,:,i-90]=fits.open(filestart+str(0)+str(i)+".fits.gz")[1].data[:,:]
  # print("The median bias image for chip 1 for",filestart+str(0)+str(i)+".fits.gz'
x1=np.median(arr1,axis=2)
print(x1)
```

```
[[ 996. 1000. 1001. ...  995.  994.  993.]
 [ 992. 1001. 1001. ...  993.  994.  994.]
 [ 994. 1000. 1000. ...  993.  994.  994.]
 ...
 [ 993. 1001. 1000. ...  992.  993.  993.]
 [ 995. 1000.  999. ...  992.  994.  992.]
 [ 992. 1001. 1000. ...  993.  994.  995.]]
```

```
# to check that we made a 2d array
x1.shape
```

```
(4096, 2140)
```

Repeat the previous code for all the chips

```
#Chip 2
filestart="d0221_0"
arr2=np.zeros((4096,2140,9)) #empty list of 0 for the array size
for i in list(range(90,99)):
  arr2[:,:,i-90]=fits.open(filestart+str(0)+str(i)+".fits.gz")[2].data[:,:]
  # print("The median bias image for chip 1 for",filestart+str(0)+str(i)+".fits.gz'
x2=np.median(arr2,axis=2)
print(x2)
```

```
[[ 993.  999. 1002. ...  997.  996.  997.]
 [ 996.  998. 1004. ...  999.  997.  997.]
 [ 997.  997. 1004. ...  996.  997.  996.]
 ...
 [ 997.  996. 1002. ...  996.  996.  996.]
 [ 995.  997. 1002. ...  994.  995.  995.]
 [ 996.  997. 1002. ...  996.  995.  996.]]
```

```
#Chip 3
filestart="d0221_0"
arr3=np.zeros((4096,2140,9)) #empty list of 0 for the array size
for i in list(range(90,99)):
  arr3[:,:,i-90]=fits.open(filestart+str(0)+str(i)+".fits.gz")[3].data[:,:]
  # print("The median bias image for chip 1 for",filestart+str(0)+str(i)+".fits.gz'
x3=np.median(arr3,axis=2)
print(x3)
```

```
[[1016. 1020. 1026. ... 1019. 1019. 1018.]
 [1018. 1020. 1026. ... 1018. 1018. 1019.]
 [1020. 1021. 1027. ... 1018. 1018. 1020.]
 ...
 [1017. 1020. 1027. ... 1017. 1020. 1018.]
 [1018. 1021. 1026. ... 1017. 1018. 1018.]
 [1018. 1020. 1027. ... 1019. 1019. 1017.]]
```

```
#Chip 4
filestart="d0221_0"
arr4=np.zeros((4096,2140,9)) #empty list of 0 for the array size
```

```
for i in list(range(90,99)):
  arr4[:,:,i-90]=fits.open(filestart+str(0)+str(i)+".fits.gz")[4].data[:,:]
  # print("The median bias image for chip 1 for",filestart+str(0)+str(i)+".fits.gz'
x4=np.median(arr4,axis=2)
print(x4)
```

```
[[ 994. 1000. 1007. ...  996.  998.  998.]
 [ 998.  999. 1007. ...  997.  998.  996.]
 [ 998. 1000. 1006. ...  998.  998.  998.]
 ...
 [ 997.  998. 1004. ...  997.  997.  997.]
 [ 996.  998. 1006. ...  997.  998.  996.]
 [ 997.  998. 1006. ...  997.  997.  997.]]
```

```
#Chip 5
filestart="d0221_0"
arr5=np.zeros((4096,2140,9)) #empty list of 0 for the array size
for i in list(range(90,99)):
  arr5[:,:,i-90]=fits.open(filestart+str(0)+str(i)+".fits.gz")[5].data[:,:]
  # print("The median bias image for chip 1 for",filestart+str(0)+str(i)+".fits.gz'
x5=np.median(arr5,axis=2)
print(x5)
```

```
[[ 996.  993. 1015. ... 1006. 1007. 1006.]
 [1007.  994. 1014. ... 1007. 1006. 1007.]
 [1005.  994. 1014. ... 1005. 1005. 1005.]
 ...
 [1006.  993. 1013. ... 1008. 1007. 1007.]
 [1007.  992. 1014. ... 1004. 1006. 1005.]
 [1006.  993. 1013. ... 1006. 1005. 1007.]]
```

```
#Chip 6
filestart="d0221_0"
arr6=np.zeros((4096,2140,9)) #empty list of 0 for the array size
for i in list(range(90,99)):
  arr6[:,:,i-90]=fits.open(filestart+str(0)+str(i)+".fits.gz")[6].data[:,:]
  # print("The median bias image for chip 1 for",filestart+str(0)+str(i)+".fits.gz'
x6=np.median(arr6,axis=2)
print(x6)
```

```
[[ 996.  998. 1008. ... 1000. 1001. 1000.]
 [1000.  997. 1008. ... 1001. 1001. 1000.]
 [ 999.  997. 1007. ... 1000. 1000.  998.]
 ...
 [1000.  996. 1008. ...  999. 1000. 1000.]
 [ 998.  996. 1007. ...  999. 1001. 1000.]
 [1000.  997. 1007. ...  999. 1001. 1000.]]
```

```
#Chip 7
filestart="d0221_0"
arr7=np.zeros((4096,2140,9)) #empty list of 0 for the array size
for i in list(range(90,99)):
  arr7[:,:,i-90]=fits.open(filestart+str(0)+str(i)+".fits.gz")[7].data[:,:]
  # print("The median bias image for chip 1 for",filestart+str(0)+str(i)+".fits.gz'
x7=np.median(arr7,axis=2)
print(x7)
```

```
[[ 999.  999. 1008. ... 1001. 1000. 1000.]
 [1001. 1000. 1007. ... 1000. 1001. 1000.]
 [1000. 1000. 1007. ...  999. 1000. 1000.]
 ...
 [1000. 1001. 1006. ...  998.  999.  998.]
 [ 998.  999. 1008. ... 1000. 1001.  999.]
 [1000. 1001. 1005. ... 1000.  998. 1000.]]
```

```
#Chip 8
filestart="d0221_0"
arr8=np.zeros((4096,2140,9)) #empty list of 0 for the array size
for i in list(range(90,99)):
  arr8[:,:,i-90]=fits.open(filestart+str(0)+str(i)+".fits.gz")[8].data[:,:]
  # print("The median bias image for chip 1 for",filestart+str(0)+str(i)+".fits.gz'
x8=np.median(arr6,axis=2)
print(x8)
```

```
[[ 996.  998. 1008. ... 1000. 1001. 1000.]
 [1000.  997. 1008. ... 1001. 1001. 1000.]
 [ 999.  997. 1007. ... 1000. 1000.  998.]
 ...
 [1000.  996. 1008. ...  999. 1000. 1000.]
 [ 998.  996. 1007. ...  999. 1001. 1000.]
 [1000.  997. 1007. ...  999. 1001. 1000.]]
```

```
hdr = fits.Header( )
hdr["CODER"] = "M C Cooper"
hdr["COMMENT"] = "Physics is Phundamental!"
empty_primary = fits.PrimaryHDU(header=hdr)
# no longer an image, it is now packed into a fits file
hdu1 = fits.ImageHDU(x1)
hdu2 = fits.ImageHDU(x2)
hdu3 = fits.ImageHDU(x3)
hdu4 = fits.ImageHDU(x4)
hdu5 = fits.ImageHDU(x5)
hdu6 = fits.ImageHDU(x6)
hdu7 = fits.ImageHDU(x7)
hdu8 = fits.ImageHDU(x8)
```

```
hdu_list = fits.HDUList([empty_primary, hdu1, hdu2, hdu3, hdu4, hdu5, hdu6, hdu7, h
hdu_list.writeto("DEIMOS_bias.fits")
```

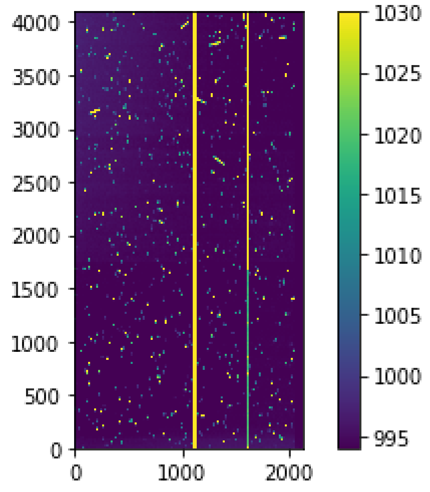
```
var1=fits.open("DEIMOS_bias.fits")
var1.info()
```

```
Filename: DEIMOS_bias.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU       4   ()      
  1                1 ImageHDU         7   (2140, 4096)   float64   
  2                1 ImageHDU         7   (2140, 4096)   float64   
  3                1 ImageHDU         7   (2140, 4096)   float64   
  4                1 ImageHDU         7   (2140, 4096)   float64   
  5                1 ImageHDU         7   (2140, 4096)   float64   
  6                1 ImageHDU         7   (2140, 4096)   float64   
  7                1 ImageHDU         7   (2140, 4096)   float64   
  8                1 ImageHDU         7   (2140, 4096)   float64   
```

Part Two

```
img_c8 = fits.open("d0221_0099.fits.gz")[8].data
```

```
plt.imshow(img_c8, origin='lower', vmin=994,vmax=1030,cmap='viridis')
cb = plt.colorbar()
plt.show()
```



Now, plot the distribution of counts per pixel for pixels on chip 8 with values within plus or minus 50 of the median pixel value (plot the distribution of counts for only the pixels within plus or minus 50 of the median pixel value). Overplot a vertical line to denote the median pixel value.

```
m = np.median(img_c8)
print(m)
```

```
994.0
```

```
plus = m + 50
minus = m - 50
```

```
print(minus,plus)
# Pixels within plus or minus 50 of the median pixel value: 944 - 1044
```

```
944.0 1044.0
```

```
np.where((img_c8 > 944) & (img_c8 < 1044))
# This code finds the slice of the image
```

```
(array([   0,    0,    0, ..., 4095, 4095, 4095]),
 array([   0,    1,    2, ..., 2137, 2138, 2139]))
```

```
xloc = np.where((img_c8 > 944) & (img_c8 < 1044))[1]
```

```
xloc
```

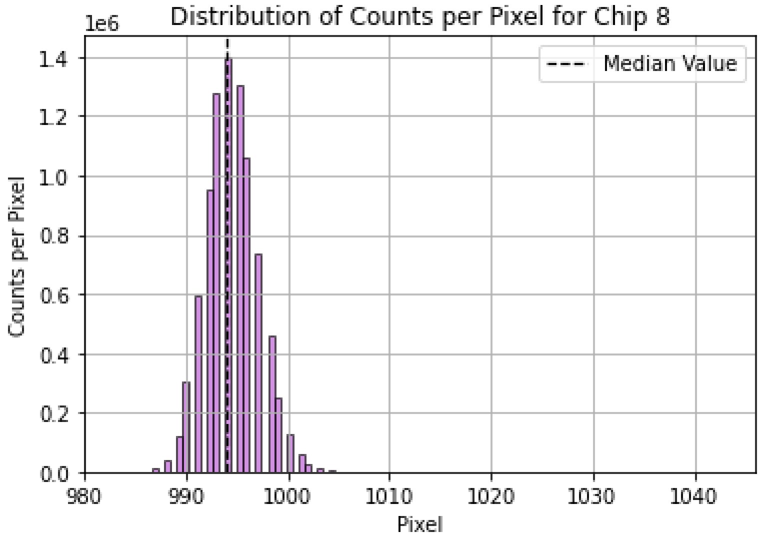
```
array([   0,    1,    2, ..., 2137, 2138, 2139])
```

```
yloc = np.where((img_c8 > 944) & (img_c8 < 1044))[0]
```

```
smaller_imgc8 = img_c8[np.where((img_c8 > 944) & (img_c8 < 1044))]
```

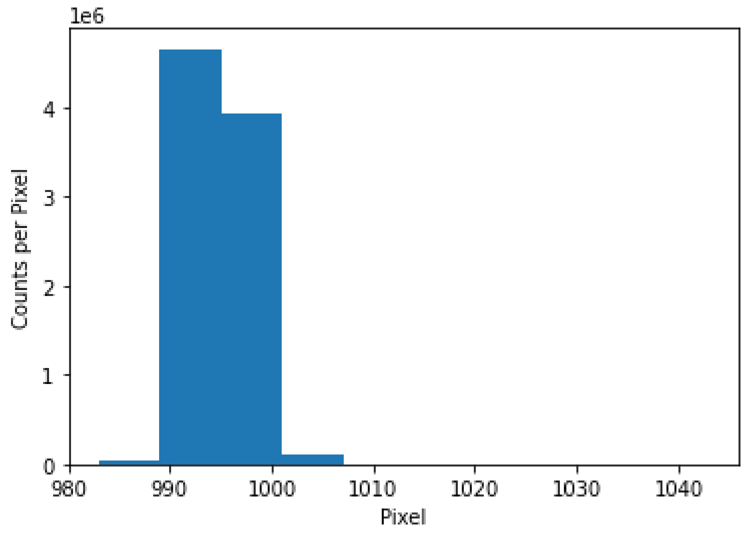
```
# Plotting the histogram with the sliced image.
plt.hist(smaller_imgc8,alpha=0.65,color='mediumorchid',edgecolor='k',bins=100)
plt.title("Distribution of Counts per Pixel for Chip 8")
plt.axvline(m, linestyle='dashed', color='k',linewidth=1.25,label="Median Value")
plt.legend()
plt.grid()
plt.xlabel("Pixel")
plt.ylabel("Counts per Pixel")
```

```
Text(0, 0.5, 'Counts per Pixel')
```



```
# Please disregard the following code; I recreated the same histogram using the
# arrays for x and y location to slice the image. Just wanted to check and see
# if it would produce the same graph for my own understanding.
# Plotting the histogram using the arrays for x and y axes to slice the image.
plt.hist(img_c8[yloc,xloc])
plt.xlabel("Pixel")
plt.ylabel("Counts per Pixel")
```

```
Text(0, 0.5, 'Counts per Pixel')
```



Fraction of the total pixels on chip 8 that have a value within plus or minus 50 of the median pixel value

```
img_c8.shape
```

```
(4096, 2140)
```

```
n = (len(smaller_imgc8)/(4096*2140))*100
print(str(n) + "% of Pixels have a value within plus-or-minus 50 of the median pixe
```

```
99.70534280081776% of Pixels have a value within plus-or-minus 50 of the median 
```

```
small_std = np.std(smaller_imgc8)
print(small_std)
```

```
2.5835420823057285
```

```
np.std(img_c8)
```

```
1367.4290735582474
```

They are significantly different because the total data set being used to calculate the standard deviation for the entire image is much greater than the data set being used to calculate the standard deviation for the sliced image.

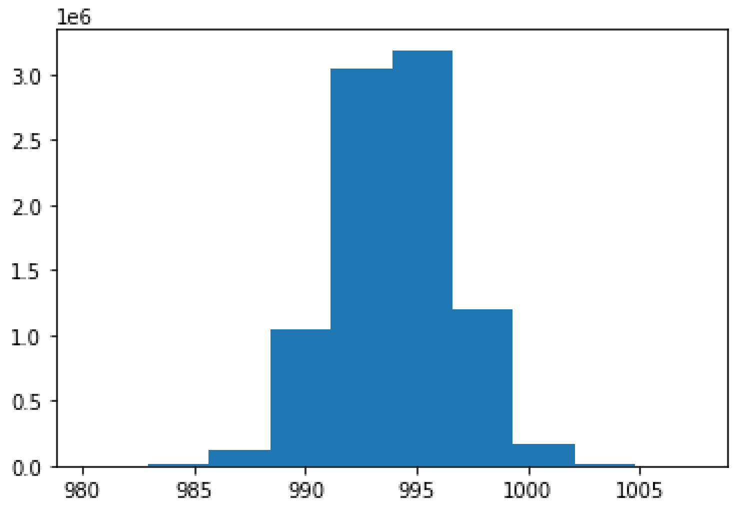
The disparity in the dispersion between the entire image and the sliced image is proportionate to the amount of data being used to calculate each dispersion value.

Overplotting a Gaussion distribution onto the previous histogram:

```
gd = np.random.normal(np.median(smaller_imgc8),small_std,4096*2140)
```

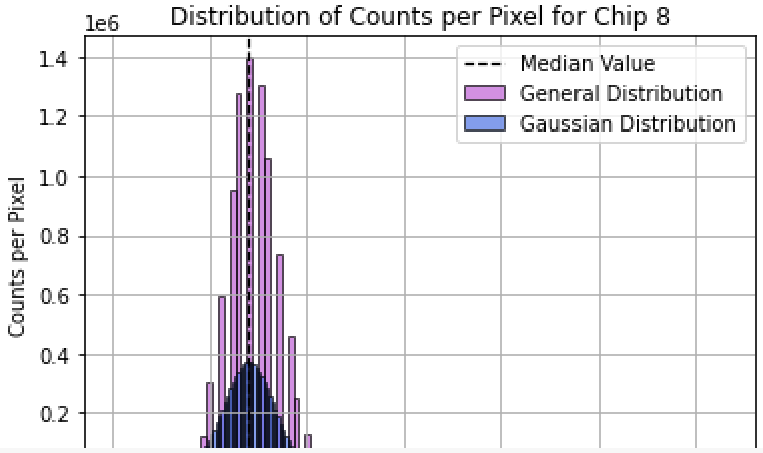
```
plt.hist(gd)
```

```
(array([9.700000e+01, 5.641000e+03, 1.272450e+05, 1.040862e+06,
        3.035414e+06, 3.184546e+06, 1.201983e+06, 1.618270e+05,
        7.692000e+03, 1.330000e+02]),
 array([ 980.20226831,  982.93627834,  985.67028837,  988.4042984 ,
         991.13830844,  993.87231847,  996.6063285 ,  999.34033853,
        1002.07434856, 1004.8083586 , 1007.54236863]),
 <a list of 10 Patch objects>)
```



```
plt.hist(smaller_imgc8,alpha=0.65,color='mediumorchid',edgecolor='k',label="General
plt.hist(gd,alpha=0.65,color="royalblue",edgecolor='k',label="Gaussian Distributior
plt.title("Distribution of Counts per Pixel for Chip 8")
plt.axvline(m, linestyle='dashed', color='k',linewidth=1.25,label="Median Value")
plt.legend()
plt.grid()
plt.xlabel("Pixel Value")
plt.ylabel("Counts per Pixel")
```

```
Text(0, 0.5, 'Counts per Pixel')
```



```
print(small_std)
```

```
2.5835420823057285
```

```
a = np.where(smaller_imgc8 > small_std+m)
# a = np.where(smaller_imgc8 > (5*small_std)+m)
b = np.where(smaller_imgc8 < m-small_std)
a_s = np.size(a)
```

```
b_s = np.size(b)
```

```
np.size(smaller_imgc8)
```

```
8739612
```

```
np.std(gd)
```

```
2.5835936636873065
```

```
w = 100*((a_s+b_s)/8739612)
print(w)
```

```
31.383464162939955
```

```
e = 100-w
print(e)
```

```
68.61653583706004
```

About 68.62% of the total pixels on chip 8 have a pixel value that is above or below one standard deviation of the median pixel value. This finding aligns with the general breakdown of standard deviations of a Gaussian distribution; generally, 68.2% of the data being used to generate a

Guassian distribution lies within one standard deviation of the average value of the data set. Therefore, the percentage that I found agrees well with the expectation for a Gaussian distribution.

```
a_ten = np.where(smaller_imgc8 > (10*small_std)+m)
b_ten = np.where(smaller_imgc8 < m-(10*small_std))
a_ten_s = np.size(a_ten)
b_ten_s = np.size(b_ten)

sol = 100*((a_ten_s + b_ten_s)/8739612)
# sol = fraction of data that lies outside of 10 stds
g_sol = 100 - sol
# g_sol = fraction of data that lies within 10 stds
print(str(sol) + "," + str(g_sol))
```

```
0.012323201533431919,99.98767679846657
```

```
a_5 = np.where(smaller_imgc8 > (5*small_std)+m)
b_5 = np.where(smaller_imgc8 < m-(5*small_std))
a_5_s = np.size(a_5)
b_5_s = np.size(b_5)

sol_5 = 100*((a_5_s + b_5_s)/8739612)
# sol_5 = fraction of data that lies outside of 5 stds
g_sol_5 = 100 - sol_5
# g_sol_5 = fraction of data that lies within 5 stds
print(str(sol_5) + "," + str(g_sol_5))
```

```
0.06498000140051985,99.93501999859949
```

```
a_100 = np.where(smaller_imgc8 > (100*small_std)+m)
b_100 = np.where(smaller_imgc8 < m-(100*small_std))
a_100_s = np.size(a_100)
b_100_s = np.size(b_100)

sol_100 = 100*((a_100_s + b_100_s)/8739612)
# sol_100 = fraction of data that lies outside of 100 stds
g_sol_100 = 100 - sol_100
# g_sol_100 = fraction of data that lies within 100 stds
print(str(sol_100) + "," + str(g_sol_100))
```

```
0.0,100.0
```

The fractions of pixels that lie within 5, 10, and 100 standard deviations are very similar to the expected values for a Gaussian distribution.

For chip 8 of frame d0221_0099.fits.gz, make a figure illustrating the location of those pixels with values that are greater than 10 times the standard deviation above the median pixel value. (again, using the standard deviation value determined above)

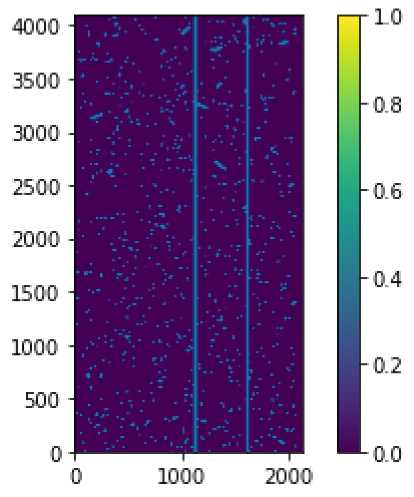
```
a_ten2 = np.array(np.where(img_c8 > (10*small_std)+m))
b_ten2 = np.array(np.where(img_c8 < m-(10*small_std)))
```

```
a_ten2[0] # y positions of imgc8 that satisfy greater than 10 sigma
a_ten2[1] # x positions of imgc8 that satisfy greater than 10 sigma
# Similarly, b_ten2[0] is an array of the y positions that satisfy the condition
# of being below 10 sigma, and b_ten2[1] is an array of the x positions that
# satisfy the condition of being below 10 sigma.
```

```
array([1126, 1127, 1128, ..., 1722, 1723, 1724])
```

```
# The following code plots the x and y values of the positions of pixels that
# are below and above 10 sigma.

plt.imshow(img_c8, origin='lower',cmap='viridis')
plt.scatter(np.array(a_ten2)[1],np.array(a_ten2)[0],s=0.01)
plt.scatter(np.array(b_ten2)[1],np.array(b_ten2)[0],s=0.01)
cb = plt.colorbar()
plt.show()
```


4000
3500
3000
2500
2000
1500
1000
500
0
0
1000
2000
1.0
0.8
0.6
0.4
0.2
0.0