Lab 1: Properties of CCD Cameras

Author: 112166935

Co-Authors: 112601517, 112695826

Before we perform any analysis, we must import the necessary Python packages.

In [206…

```
### for array operations
import numpy as np

### for plotting
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
%matplotlib inline

### for operations on FITS images
from astropy.io import fits

### statistics functions needed in this tutorial
from scipy import stats
from scipy.stats import norm
from scipy.stats import mode
import scipy as sp

### for changing directory
import os

### for displaying images (not very good but better than nothing)
from astropy.visualization import astropy_mpl_style
plt.style.use(astropy_mpl_style)
from astropy.visualization import simple_norm

### for curve fitting
from scipy.optimize import curve_fit

### for auto-completing things in loops
import glob

os.chdir('...')   # Replace ... with the path to directory with CCD images
```

4.1 Bias Frames

4.1.1

We open the $0°$ bias frame in Python.

In [207…

```
### Open bias file - May need to change to your directory
```

```
os.chdir('./Imaging CCD')

with fits.open('bias0C.00000000.BIAS.FIT') as hdulist:
    ### Basic Info and Shape of image
    hdulist.info()
    imagedata = hdulist[0].data
    print('Size:', imagedata.shape)
```

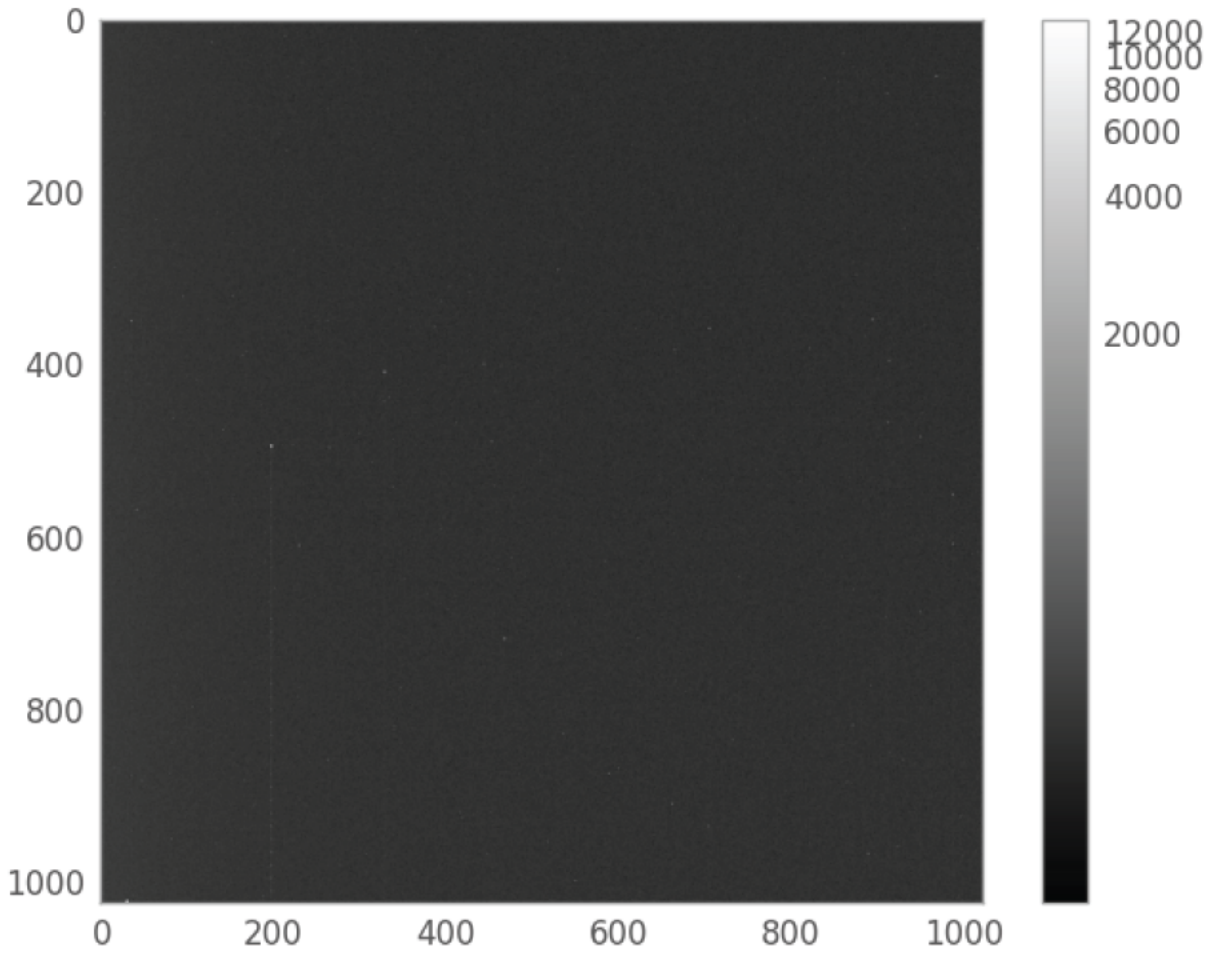
```
Filename: bias0C.00000000.BIAS.FIT
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU      45   (1024, 1024)   int16 (rescales to uint16)
Size: (1024, 1024)
```

We see that the bias frame is a 1024 x 1024 array of integer-valued counts. The bias frame is an image that can be viewed in Python but is ideally viewed in DS9.

We generate the CCD bias frame image with pixel intensity on a log scale.

In [208…
```
### Make a quick image (hard to see hot pixels, so open in ds9 for better look)
plt.figure()
plt.grid(False)
bias_norm=simple_norm(imagedata, 'log')
plt.imshow(imagedata, cmap='gray', norm=bias_norm) #bias_norm is a placeholder for if
plt.colorbar()
```

Out[208]:
```
<matplotlib.colorbar.Colorbar at 0x24d0a151970>
```



It is impractical to visually identify hot pixels, so a histogram would be helpful.

We find the minimum and maximum pixel count values to determine the range of our histogram.

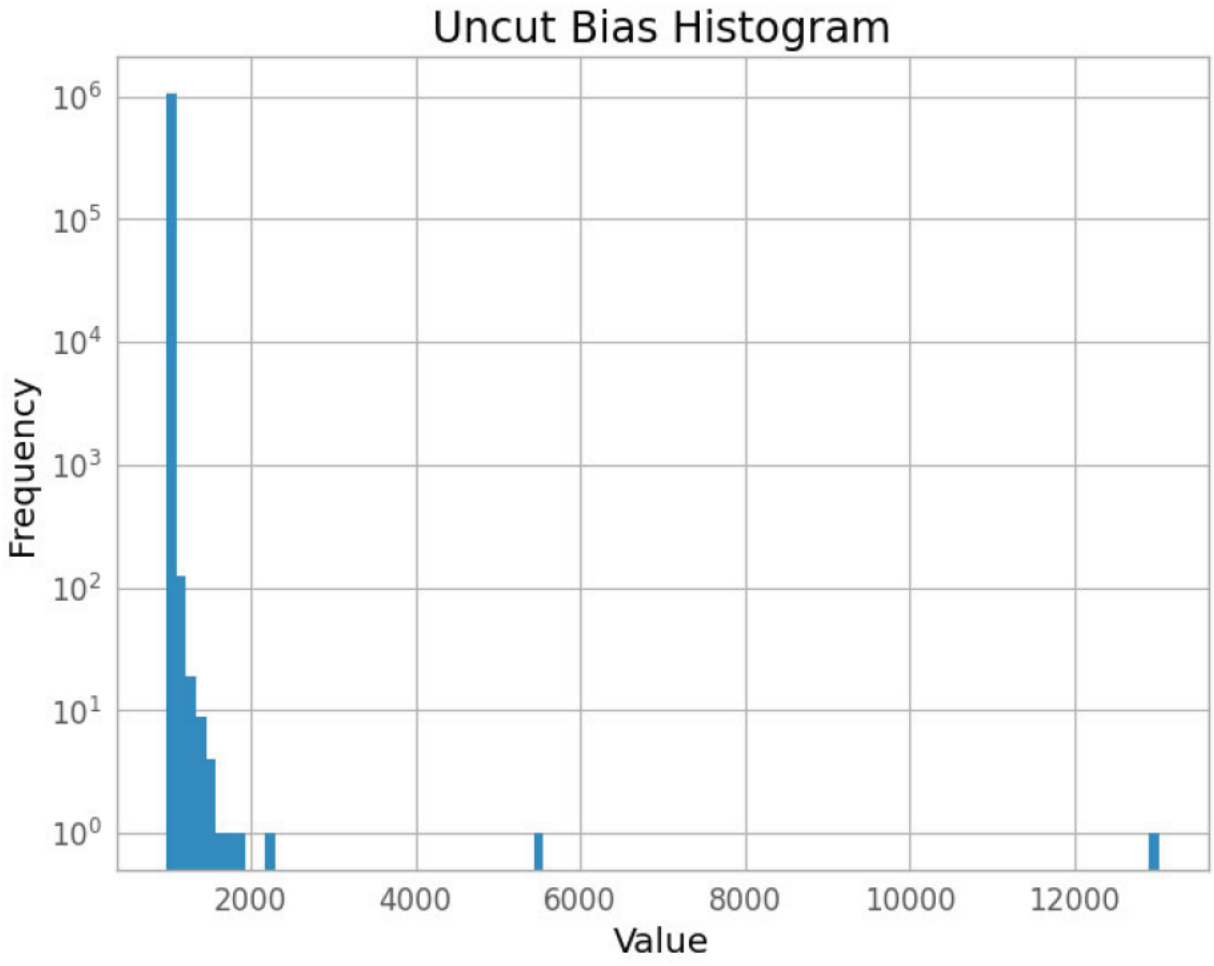
In [209…
```
### Find minmax values to see what we should set our x range to in the histogram

print('Maximum:',np.max(imagedata))
print('Minimum:',np.min(imagedata))
```

```
Maximum: 13024
Minimum: 972
```

We generate our histogram of counts. Our histogram has a log scale y-axis. We need to flatten our image data into a 1D array in order to plot it.

In [210…
```
### Make the histogram
countvalues = imagedata.flatten()
plt.hist(countvalues,bins=100)
plt.yscale('log')
plt.title('Uncut Bias Histogram')
plt.xlabel('Value')
plt.ylabel('Frequency')
```

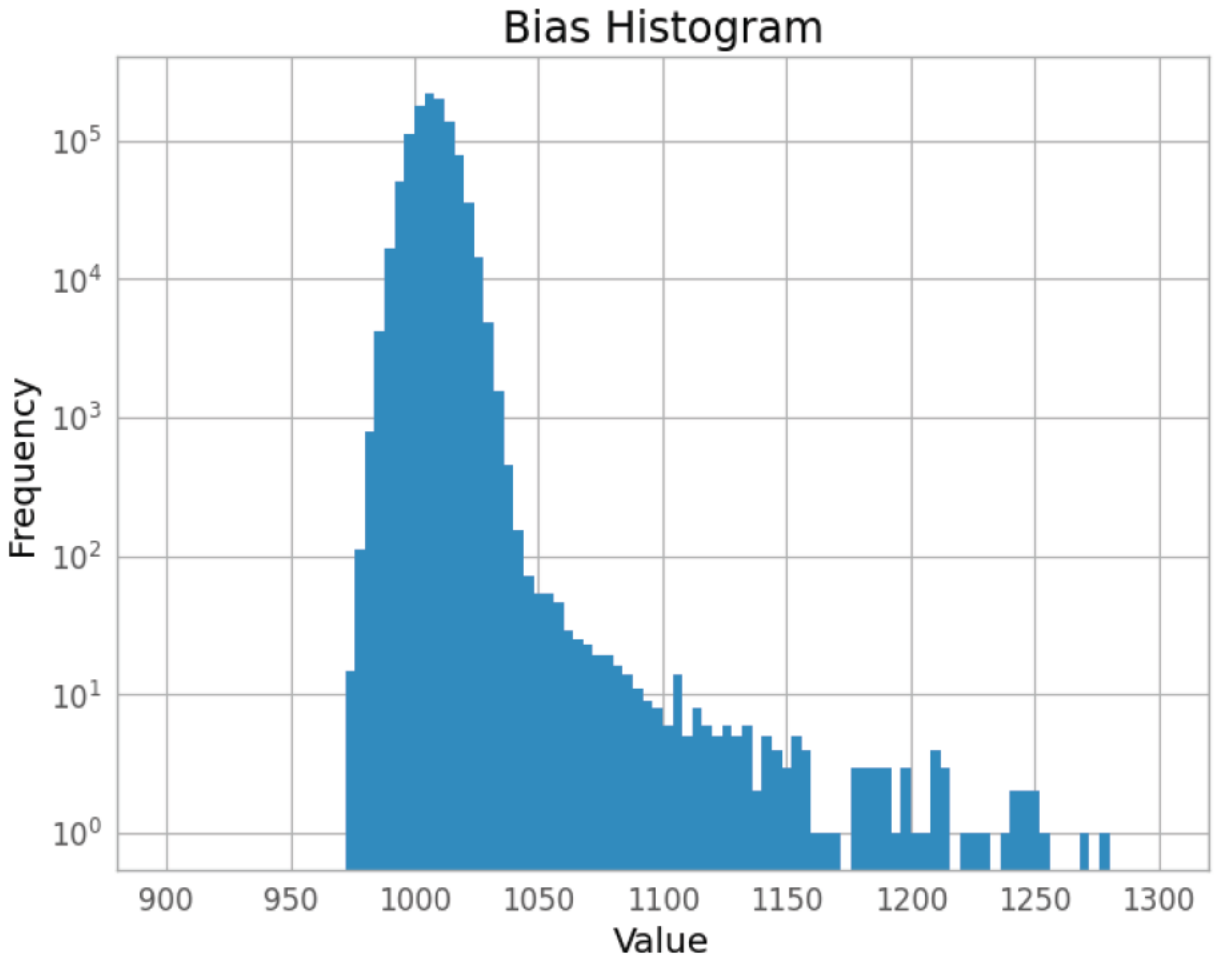
Out[210]: `Text(0, 0.5, 'Frequency')`



We can see several hot pixels in our image. We cut our counts range to [900,1300] to see the counts distribution without the hot pixels.

In [211…
```
### Make our range nicer to see the distribution more clearly
plt.hist(countvalues,bins=100,range=[900,1300])
plt.yscale('log')
plt.title('Bias Histogram')
plt.xlabel('Value')
plt.ylabel('Frequency')
```

Out[211]: Text(0, 0.5, 'Frequency')



We see there are still a few hot pixels in the 1200 and higher range. Gaps begin to appear in the bias histogram above 1150, so we choose 1150 as our upper bound for pixel value.

We calculate our proportion of rejected pixels as follows: First, we find the original number of pixels, which is $1024 \times 1024 = 1048576$. Next, we clip the 1D array of pixel counts such that only values < 1150 are included. This new array has 1048474 pixels. This means that $1048576 - 1048474 = 102$ hot pixels were rejected. Therefore, the proportion of pixels rejected is

$$\frac{\text{hot pixels}}{\text{total pixels}} = \frac{102}{1048576} = 9.727 \times 10^{-5} \approx 10^{-4},$$

or just under $(10^{-2})\% = 0.01\%$. In other words, approximately 0.01% of pixels are hot.

In [212…

```
### Calculate proportion of hot pixels

# Find the original number of pixels
originalsize = np.prod(countvalues.shape)
print('Original number of pixels:', originalsize)

# Clip the original countvalues array into a new array clippedvalues that only has val
clippedvalues = countvalues[countvalues<1125]

# Find the size of clipped values array
clippedsize = np.prod(clippedvalues.shape)
print('Clipped number of pixels:', clippedsize)

print('Hot Pixel Proportion: ', (originalsize-clippedsize)/originalsize)
print(1048576-1048474)
```

```
Original number of pixels: 1048576
Clipped number of pixels: 1048474
Hot Pixel Proportion:  9.72747802734375e-05
102
```

4.1.2

In order to find the read noise (in counts) of the CCD, we find the standard deviation of the bias frame. However, the standard deviation should describe the majority of bias frame without accounting for hot pixels.

Once again, we must cut off the counts values to within a reasonable range that removes hot pixels while preserving the bulk of the data. We take the mean and standard deviation of the clipped values, and use them to find the readout noise.

First, we calculate the normalization constant of the corresponding normal distribution. Using our earlier cutoff range of [900,1300], we calculate the normalization constant as follows:

$$\text{normalization} = \frac{cmax-cmin}{\#\text{ of bins}} \times \#\text{ of counts}$$

In [213…

```
### Calculate some stats, clipping values
cmin=900
cmax=1300
nbins=100
normalization=(cmax-cmin)/nbins*len(countvalues[(countvalues>=cmin) & (countvalues<=cm
```

Next, we clip the histogram values from which we calculate the mean, standard deviation, range, and mode.

In [214…

```
clipmin=cmin
clipmax=1075
clippedvalues = countvalues[(countvalues>=clipmin) & (countvalues<=clipmax)]
```

Next, we calculate the mean, standard deviation, range, and mode of the data. The statistical uncertainty of the mean is given by

$\sigma/\sqrt{\text{clipped } \# \text{ of counts}}$

where σ is the standard deviation of the clipped pixel counts.

In [215…
```
mu=np.mean(clippedvalues)
sig=np.std(clippedvalues)
mode=stats.mode(clippedvalues)[0][0]

print("Range:",clipmax-clipmin)
print("Average:",mu)
print("Stdev:",sig)
print("Mode:",mode)

statuncertainty = sig / (np.prod(clippedvalues.shape))**(1/2)
print("Stat uncertainty of mean:",statuncertainty)
```

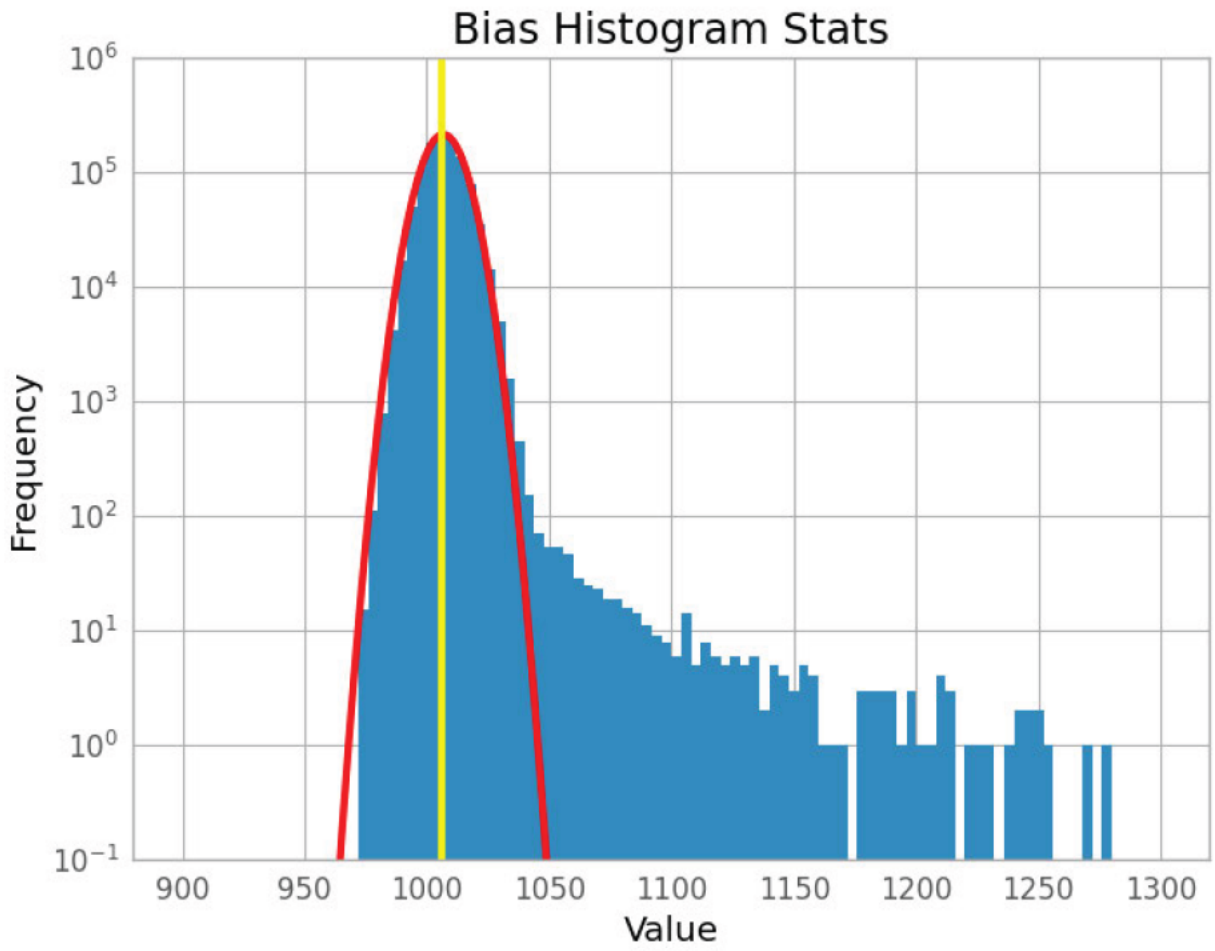
```
Range: 175
Average: 1006.7291346123579
Stdev: 7.817399627143881
Mode: 1006
Stat uncertainty of mean: 0.00763499851380355
```

```
[redacted] FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode=stats.mode(clippedvalues)[0][0]
```

We plot the histogram and a superimposed normal distribution with the same mean, standard deviation, and our calculated normalization constant.

In [216…
```
### Plot these stats on graph
xarray=np.linspace(cmin,cmax,nbins*10)
yarray=normalization*norm.pdf(xarray,loc=mu, scale=sig)

plt.hist(countvalues,range=[cmin,cmax], bins=nbins);
plt.yscale('log')
plt.ylim([0.1,1e6])
plt.plot(xarray,yarray,color="red",linewidth=3.0)
plt.axvline(x=mode,linewidth=3.0,color="yellow")
plt.title('Bias Histogram Stats')
plt.xlabel('Value')
plt.ylabel('Frequency')
```

Out[216]:
```
Text(0, 0.5, 'Frequency')
```



We read the gain of the CCD from the image header. We calculate the readout noise as follows:

Readout noise = $\sigma \times$ gain

In [217…
```
### Find out the gain
header = hdulist[0].header
gain = header['EGAIN']
print('Gain:',gain)
print('Readout Noise:',sig*gain, 'e-')
```

```
Gain: 2.06
Readout Noise: 16.103843231916397 e-
```

Therefore, our calculated readout noise from our experimental bias frame is $16.1e^-$.

The data sheet for the STL-1001E gives a readout noise of $14.8e^-$. Therefore, our readout noise is close to the accepted value. This is consistent with the manufacturer's description of $14.8e^-$ readout noise occuring in "typical" conditions; our readout noise is greater than the manufacturer's description by approximately $1e^-$.

4.2 Dark Frames

4.2.1

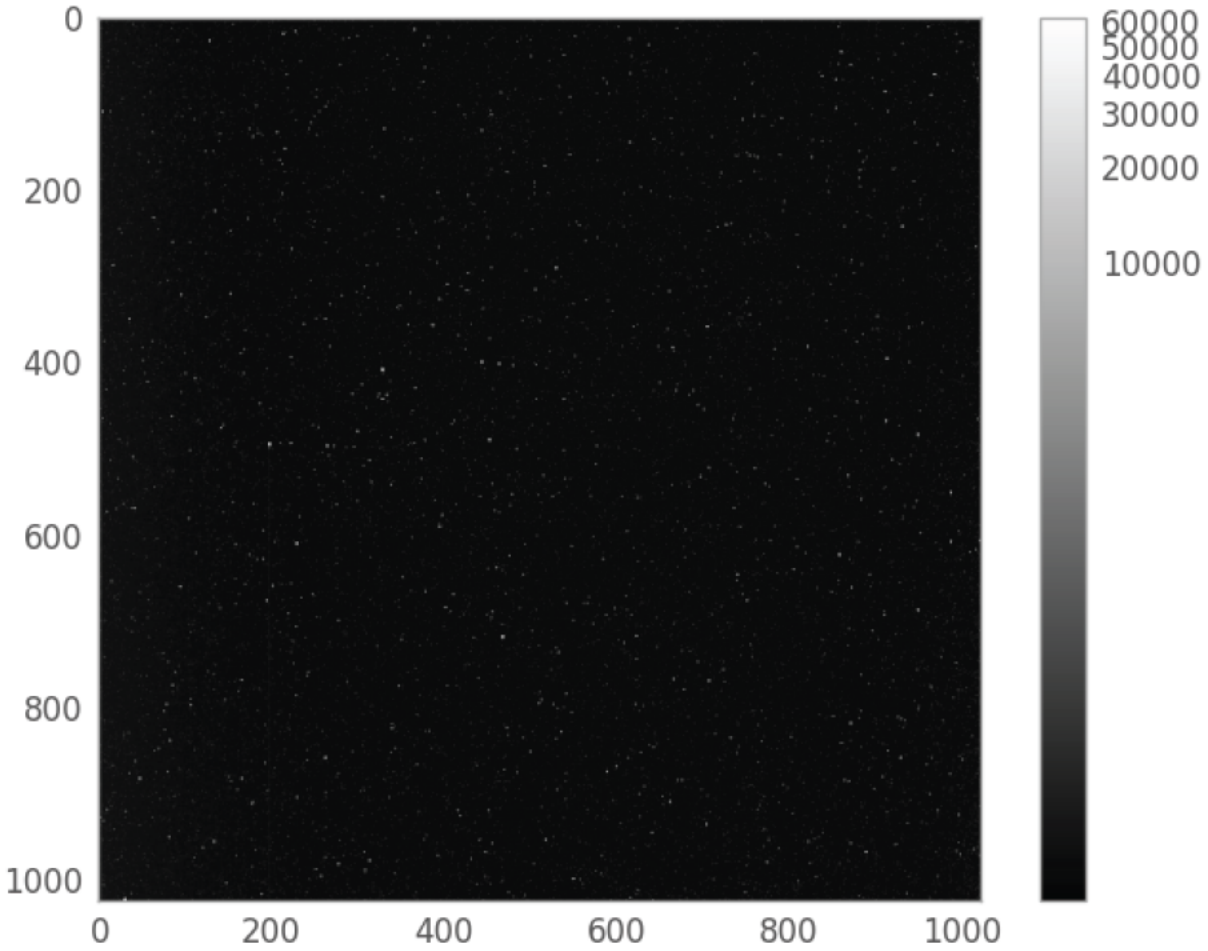
We make a master dark frame by combining 10 dark frames at 30s exposure time and taking the median. We write this master dark to a file output.

In [218…
```
### Open all dark frames for 30 s exposure
hdudark0 = fits.open('dark0C30s.00000000.DARK.FIT')
hdudark1 = fits.open('dark0C30s.00000001.DARK.FIT')
hdudark2 = fits.open('dark0C30s.00000002.DARK.FIT')
hdudark3 = fits.open('dark0C30s.00000003.DARK.FIT')
hdudark4 = fits.open('dark0C30s.00000004.DARK.FIT')
hdudark5 = fits.open('dark0C30s.00000005.DARK.FIT')
hdudark6 = fits.open('dark0C30s.00000006.DARK.FIT')
hdudark7 = fits.open('dark0C30s.00000007.DARK.FIT')
hdudark8 = fits.open('dark0C30s.00000008.DARK.FIT')
hdudark9 = fits.open('dark0C30s.00000009.DARK.FIT')

imagedata0 = hdudark0[0].data
imagedata1 = hdudark1[0].data
imagedata2 = hdudark2[0].data
imagedata3 = hdudark3[0].data
imagedata4 = hdudark4[0].data
imagedata5 = hdudark5[0].data
imagedata6 = hdudark6[0].data
imagedata7 = hdudark7[0].data
imagedata8 = hdudark8[0].data
imagedata9 = hdudark9[0].data

### Make a median of all the darks
allimages=[imagedata0,imagedata1,imagedata2,imagedata3,imagedata4,imagedata5,imagedata
median = np.median(allimages, axis=0)

### Write out
masterdark = fits.PrimaryHDU(median)
masterdark.writeto('masterdark.fits', overwrite='overwrite')
```

We show a quick image of the master dark frame.

In [219…
```
### Make a quick picture of the median
plt.figure()
plt.grid(False)
bias_norm=simple_norm(median, 'log')
plt.imshow(median, cmap='gray', norm=bias_norm)
plt.colorbar()
```

Out[219]: <matplotlib.colorbar.Colorbar at 0x24d0c150af0>



ds9 is a more preferable way to identify hot pixels in the master dark, but a quick log-scale image plot shows some bright spots that indicate hot pixels.

Since these hot pixels are in the median dark frame, they should also be present in most of the dark frames. Indeed, ds9 shows the hot pixels to remain constant between images, i.e. maintaining the same number and position.

A cursory visual inspection of the dark frames in ds9 does not reveal any hot pixels that are present in some frames but not others. Any hot pixels that are in some images but not others could be due to cosmic rays. Since cosmic rays are an infrequent phenomenon, these pixels would be outliers that are not included in the master dark.

4.2.2

For this part, we take a series of dark frames at 0° at different exposure times: 10s, 15s, 25s, 40s, 60s, 90s, 120s.

We open the fits files and read in the data.

For all exposure times, we assume the systematic uncertainty on the mean counts to be 0.1.

In [220…
```
### Open time series darks
```

```
hdudarkVARY0 = fits.open('dark0C.00000000.DARK.FIT') # 10s
hdudarkVARY1 = fits.open('dark0C.00000001.DARK.FIT') # 15s
hdudarkVARY2 = fits.open('dark0C.00000002.DARK.FIT') # 25s
hdudarkVARY3 = fits.open('dark0C.00000003.DARK.FIT') # 40s
hdudarkVARY4 = fits.open('dark0C.00000004.DARK.FIT') # 60s
hdudarkVARY5 = fits.open('dark0C.00000005.DARK.FIT') # 90s
hdudarkVARY6 = fits.open('dark0C.00000006.DARK.FIT') # 120s

imagedataVARY0 = hdudarkVARY0[0].data
imagedataVARY1 = hdudarkVARY1[0].data
imagedataVARY2 = hdudarkVARY2[0].data
imagedataVARY3 = hdudarkVARY3[0].data
imagedataVARY4 = hdudarkVARY4[0].data
imagedataVARY5 = hdudarkVARY5[0].data
imagedataVARY6 = hdudarkVARY6[0].data

hdudarkVARY0.close()
hdudarkVARY1.close()
hdudarkVARY2.close()
hdudarkVARY3.close()
hdudarkVARY4.close()
hdudarkVARY5.close()
hdudarkVARY6.close()
```

10s

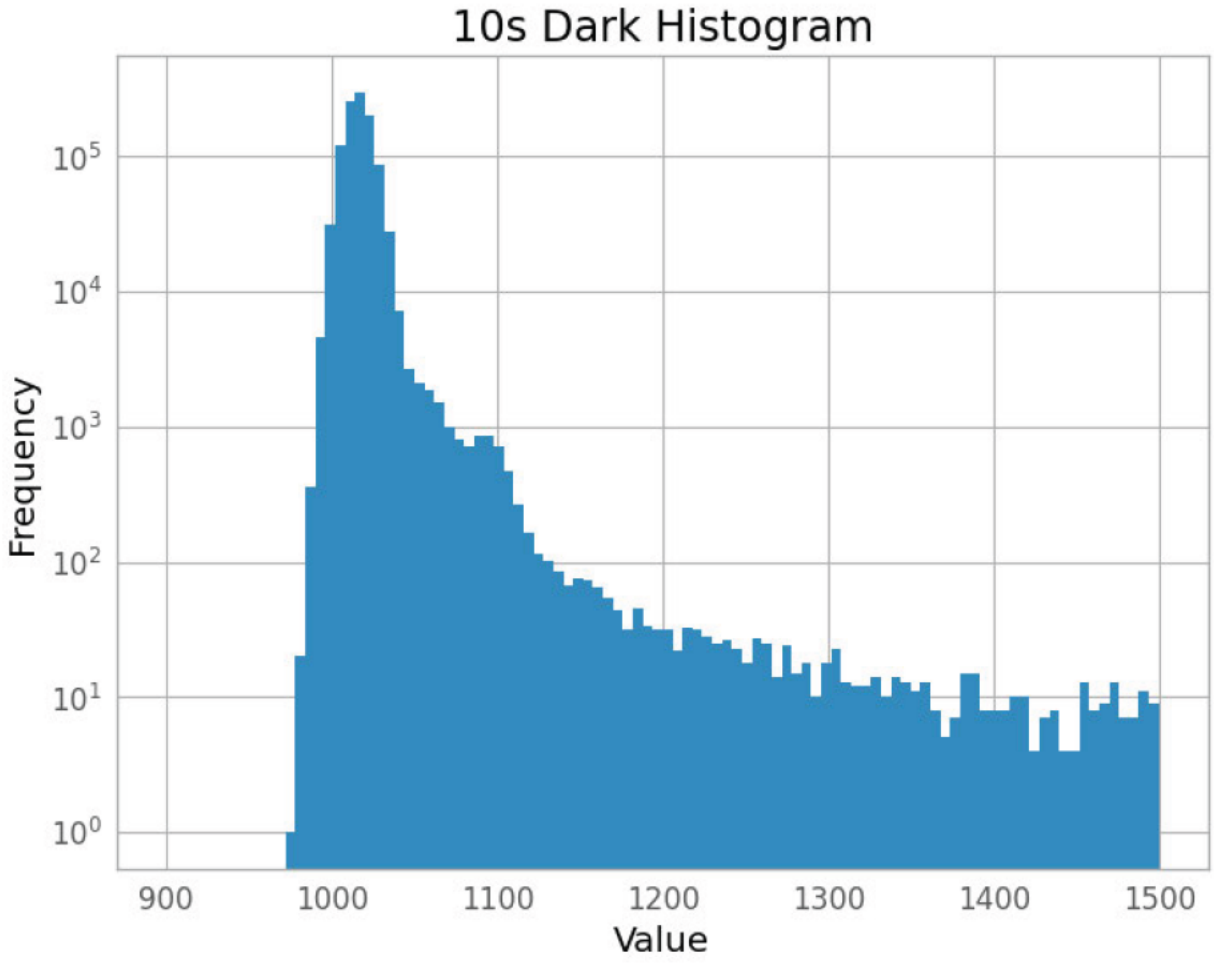
We plot the histogram and display the statistics of the dark frames taken at exposure times. We plot the count values from 900-1500 and take the stats for 900-1300.

In [221…

```
### Histogram and stats of 10s dark
countvaluesVARY0 = imagedataVARY0.flatten()

# Generate the plot
plt.hist(countvaluesVARY0,bins=100,range=[900,1500])
plt.yscale('log')
plt.title('10s Dark Histogram')
plt.xlabel('Value')
plt.ylabel('Frequency')

# Calculate the statistics
clipmaxV0=1300
clippedvaluesV0 = countvaluesVARY0[(countvaluesVARY0>=clipmin) & (countvaluesVARY0<=cl

muV0=np.mean(clippedvaluesV0)
sigV0=np.std(clippedvaluesV0)
modeV0=stats.mode(clippedvaluesV0)[0][0]

print("Range:",clipmaxV0-clipmin)
print("Average:",muV0)
print("Stdev:",sigV0)

# Statistical uncertainty on the mean
statuncertainty0 = sigV0 / (np.prod(clippedvaluesV0.shape))**(1/2)
print("Stat uncertainty on mean:",statuncertainty0)
```

```
sysuncertainty = 0.1
print("Sys uncertainty:",sysuncertainty)
```

```
[redacted] FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  modeV0=stats.mode(clippedvaluesV0)[0][0]
Range: 400
Average: 1016.6821552485266
Stdev: 11.919693917776067
Stat uncertainty on mean: 0.011645330353335455
Sys uncertainty: 0.1
```



15s

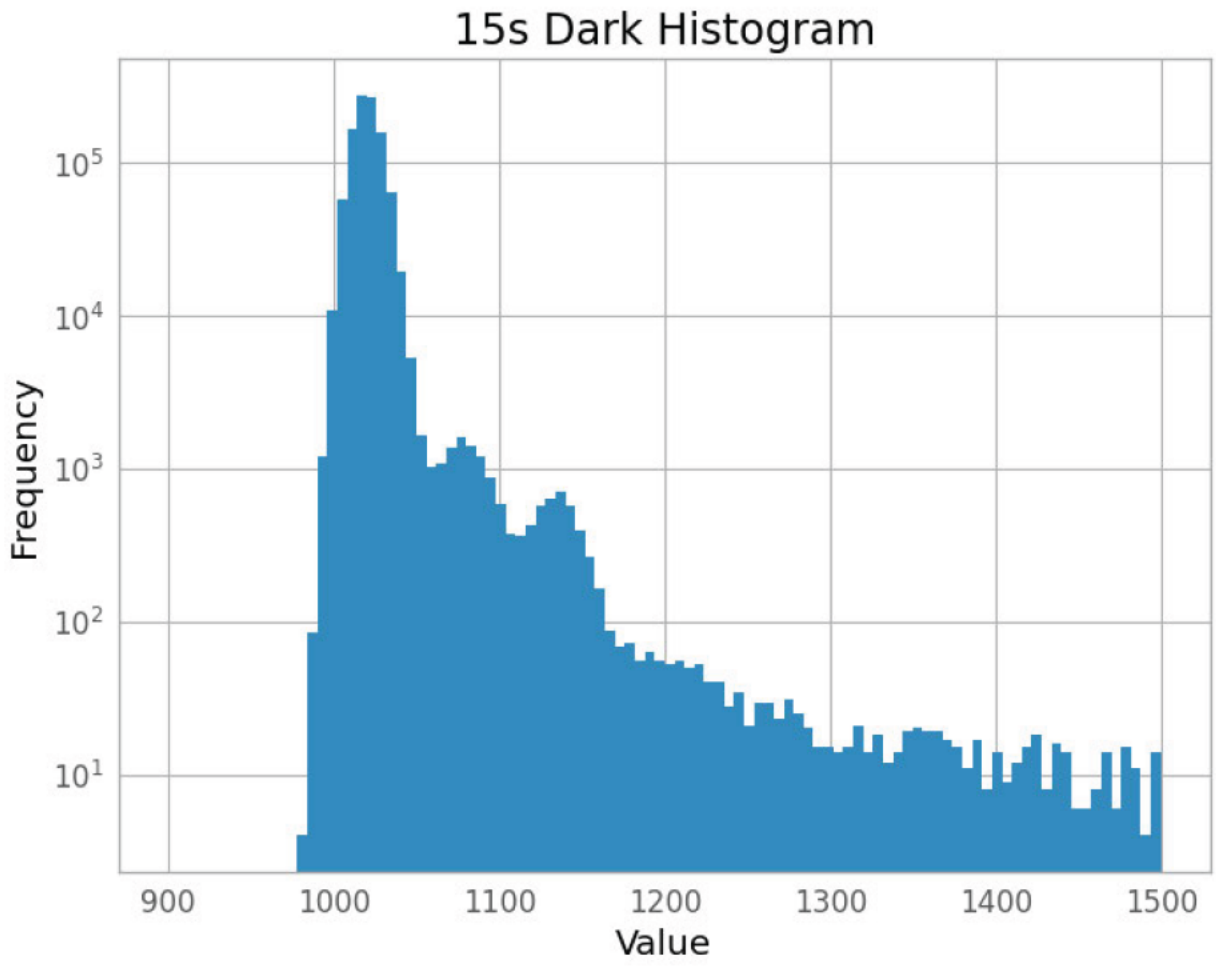
We plot the count values for 900-1500 and take the stats for 900-1300.

In [222…

```
### Histogram and stats of 15s dark
countvaluesVARY1 = imagedataVARY1.flatten()

plt.hist(countvaluesVARY1,bins=100,range=[900,1500])
plt.yscale('log')
plt.title('15s Dark Histogram')
plt.xlabel('Value')
```

```
plt.ylabel('Frequency')

clipmaxV1=1300
clippedvaluesV1 = countvaluesVARY1[(countvaluesVARY1>=clipmin) & (countvaluesVARY1<=cl

muV1=np.mean(clippedvaluesV1)
sigV1=np.std(clippedvaluesV1)
modeV1=stats.mode(clippedvaluesV1)[0][0]

print("Range:",clipmaxV1-clipmin)
print("Average:",muV1)
print("Stdev:",sigV1)

statuncertainty1 = sigV1 / (np.prod(clippedvaluesV1.shape))**(1/2)
print("Stat uncertainty on mean:",statuncertainty1)

sysuncertainty = 0.1
print("Sys uncertainty:",sysuncertainty)
```

```
[redacted] FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  modeV1=stats.mode(clippedvaluesV1)[0][0]
Range: 400
Average: 1021.0141090224415
Stdev: 13.874948862778854
Stat uncertainty on mean: 0.013557721754689637
Sys uncertainty: 0.1
```



25s

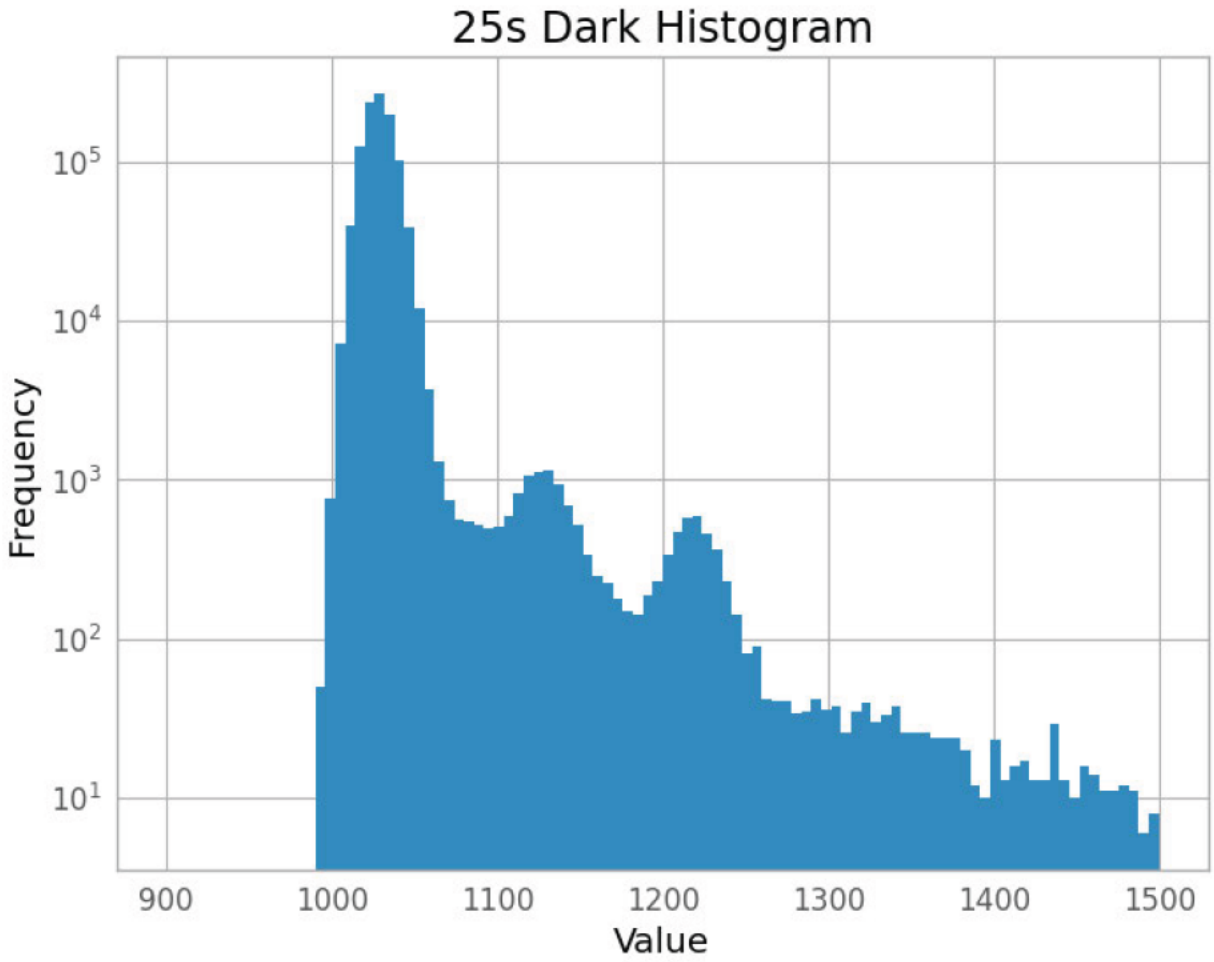
We plot the histogram and display the statistics of the dark frames taken at exposure times. We plot the count values from 900-1500 and take the stats for 900-1300.

In [223…

```
### Histogram and stats of 25s dark
countvaluesVARY2 = imagedataVARY2.flatten()

plt.hist(countvaluesVARY2,bins=100,range=[900,1500])
plt.yscale('log')
plt.title('25s Dark Histogram')
plt.xlabel('Value')
plt.ylabel('Frequency')

clipmaxV2=1300
clippedvaluesV2 = countvaluesVARY2[(countvaluesVARY2>=clipmin) & (countvaluesVARY2<=cl

muV2=np.mean(clippedvaluesV2)
sigV2=np.std(clippedvaluesV2)
modeV2=stats.mode(clippedvaluesV2)[0][0]

print("Range:",clipmaxV2-clipmin)
print("Average:",muV2)
print("Stdev:",sigV2)
```

```
statuncertainty2 = sigV2 / (np.prod(clippedvaluesV2.shape))**(1/2)
print("Stat uncertainty on mean:",statuncertainty2)

sysuncertainty = 0.1
print("Sys uncertainty:",sysuncertainty)
```

```
[redacted] FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  modeV2=stats.mode(clippedvaluesV2)[0][0]
Range: 400
Average: 1029.9901402906703
Stdev: 18.102757064122724
Stat uncertainty on mean: 0.01769444440424348
Sys uncertainty: 0.1
```



40s

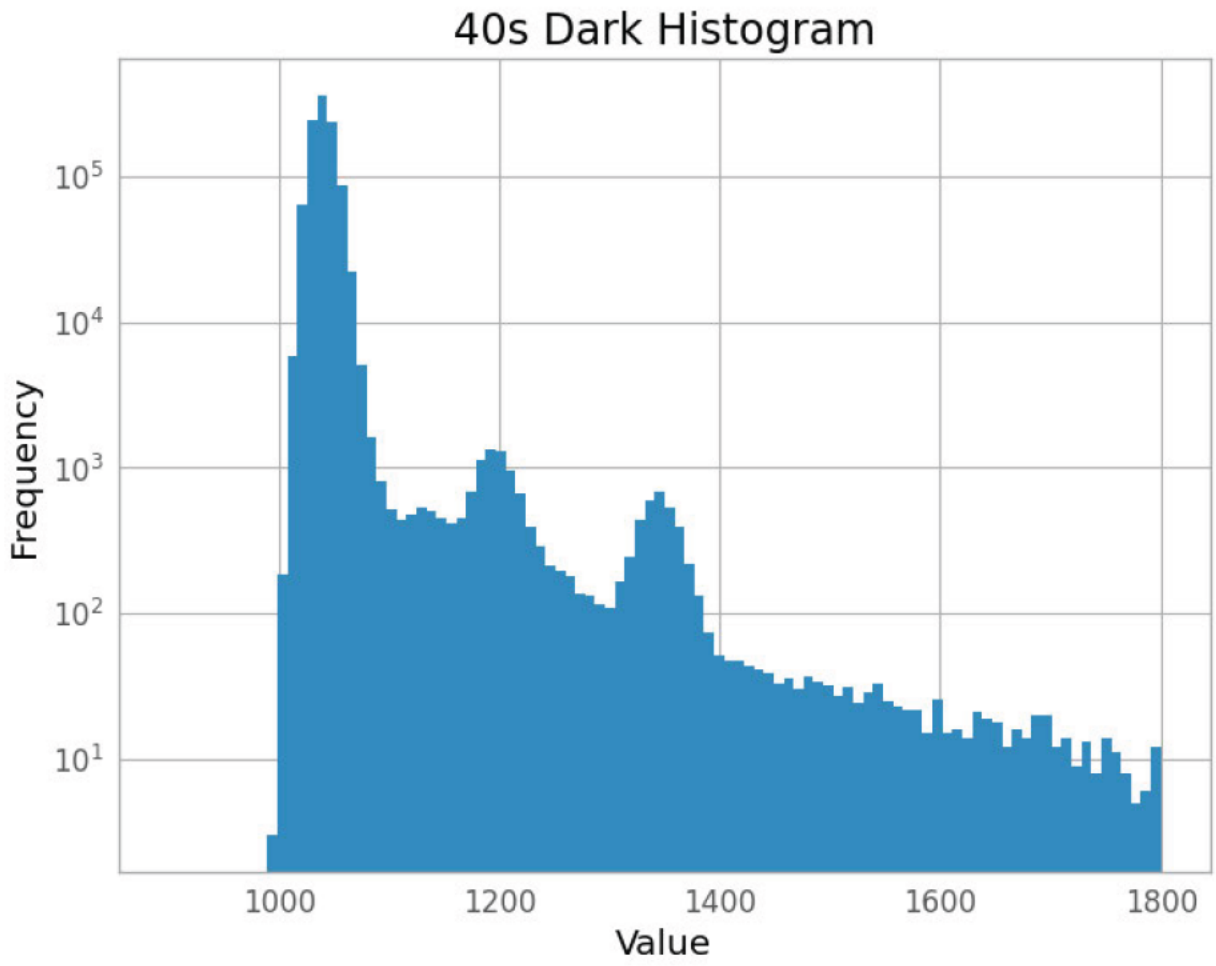
We plot the histogram and display the statistics of the dark frames taken at exposure times. We plot the count values from 900-1800 and take the stats for 900-1450.

In [224…

```
### Histogram and stats of 40s dark
countvaluesVARY3 = imagedataVARY3.flatten()
```

```
plt.hist(countvaluesVARY3,bins=100,range=[900,1800])
plt.yscale('log')
plt.title('40s Dark Histogram')
plt.xlabel('Value')
plt.ylabel('Frequency')

clipmaxV3=1450
clippedvaluesV3 = countvaluesVARY3[(countvaluesVARY3>=clipmin) & (countvaluesVARY3<=cl

muV3=np.mean(clippedvaluesV3)
sigV3=np.std(clippedvaluesV3)
modeV3=stats.mode(clippedvaluesV3)[0][0]

print("Range:",clipmaxV3-clipmin)
print("Average:",muV3)
print("Stdev:",sigV3)

statuncertainty3 = sigV3 / (np.prod(clippedvaluesV3.shape))**(1/2)
print("Stat uncertainty on mean:",statuncertainty3)

sysuncertainty = 0.1
print("Sys uncertainty:",sysuncertainty)
```

```
[redacted] FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  modeV3=stats.mode(clippedvaluesV3)[0][0]
Range: 550
Average: 1042.7662504681755
Stdev: 26.527917582439596
Stat uncertainty on mean: 0.025930415437322673
Sys uncertainty: 0.1
```



60s

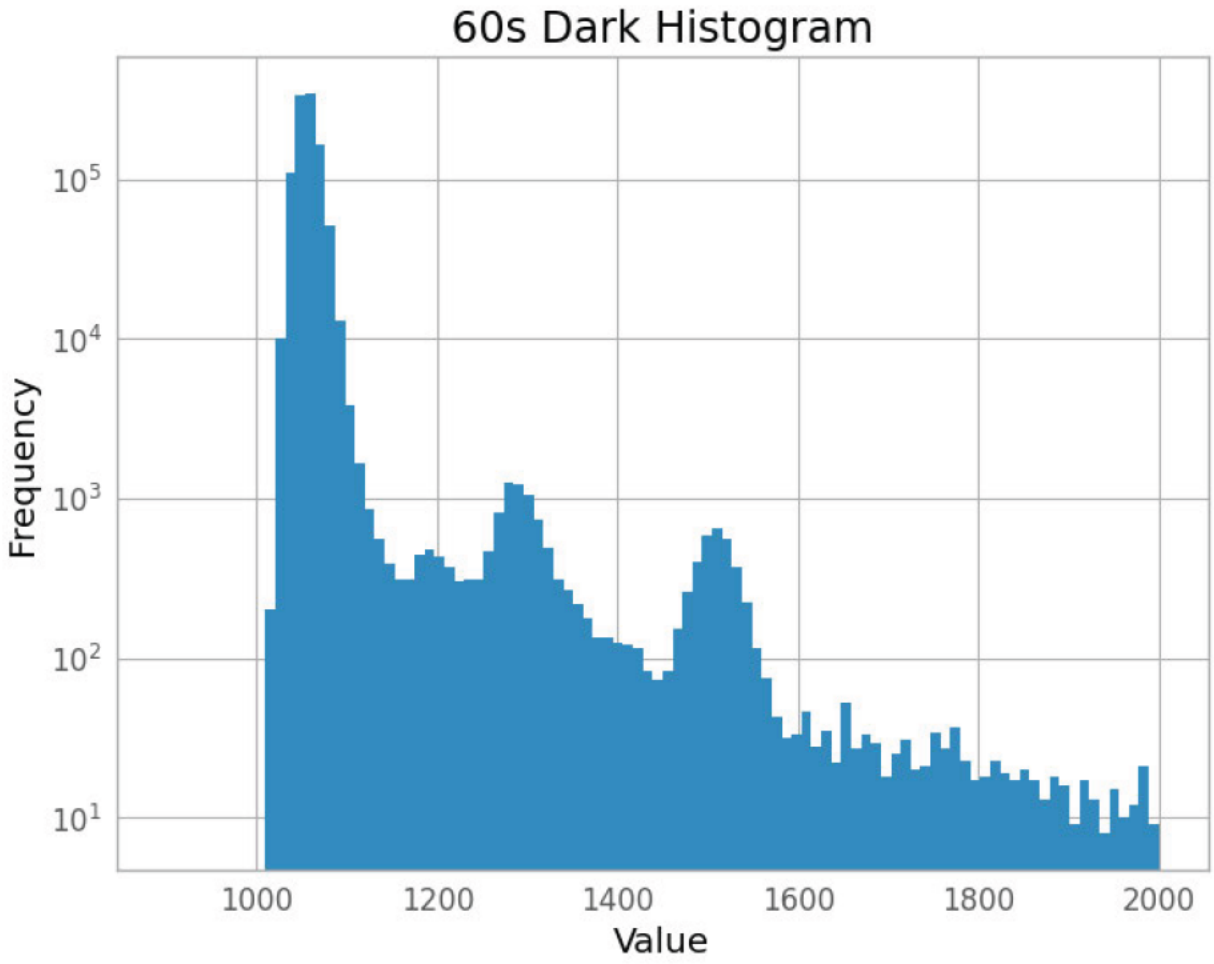
We plot the histogram and display the statistics of the dark frames taken at exposure times. We plot the count values from 900-2000 and take the stats for 900-1600.

In [225…

```
### Histogram and stats of 60s dark
countvaluesVARY4 = imagedataVARY4.flatten()

plt.hist(countvaluesVARY4,bins=100,range=[900,2000])
plt.yscale('log')
plt.title('60s Dark Histogram')
plt.xlabel('Value')
plt.ylabel('Frequency')

clipmaxV4=1600
clippedvaluesV4 = countvaluesVARY4[(countvaluesVARY4>=clipmin) & (countvaluesVARY4<=cl

muV4=np.mean(clippedvaluesV4)
sigV4=np.std(clippedvaluesV4)
modeV4=stats.mode(clippedvaluesV4)[0][0]

print("Range:",clipmaxV4-clipmin)
print("Average:",muV4)
print("Stdev:",sigV4)
```

```
statuncertainty4 = sigV4 / (np.prod(clippedvaluesV4.shape))**(1/2)
print("Stat uncertainty on mean:",statuncertainty4)

sysuncertainty = 0.1
print("Sys uncertainty:",sysuncertainty)
```

```
[redacted] FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  modeV4=stats.mode(clippedvaluesV4)[0][0]
Range: 700
Average: 1060.3274048151252
Stdev: 37.5571170167699
Stat uncertainty on mean: 0.03671479427907159
Sys uncertainty: 0.1
```



90s

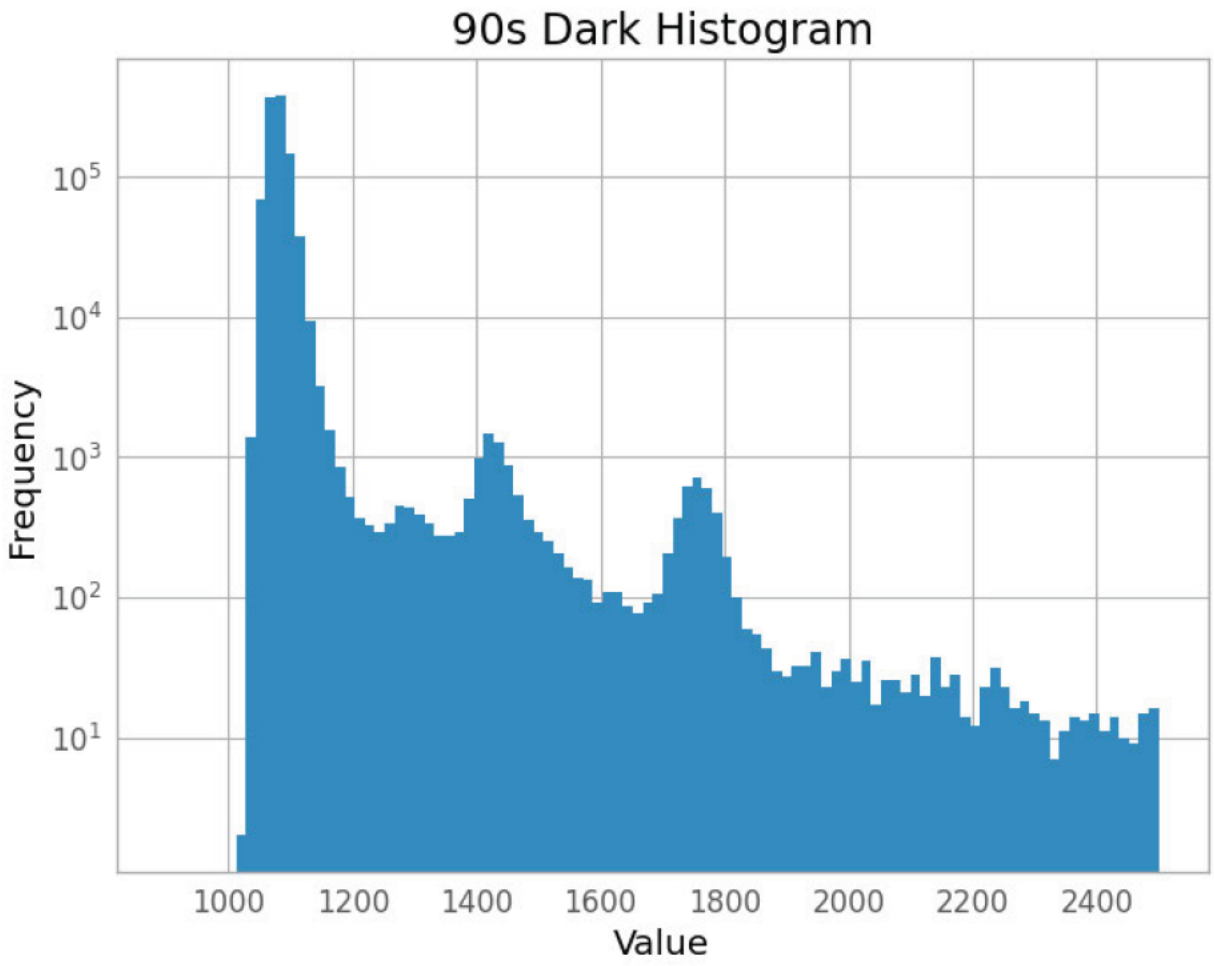
We plot the histogram and display the statistics of the dark frames taken at exposure times. We plot the count values from 900-2500 and take the stats for 900-1850.

In [226…

```
### Histogram and stats of 90s dark
countvaluesVARY5    imagedataVARY5.flatten()
```

```
plt.hist(countvaluesVARY5,bins=100,range=[900,2500])
plt.yscale('log')
plt.title('90s Dark Histogram')
plt.xlabel('Value')
plt.ylabel('Frequency')

clipmaxV5=1850
clippedvaluesV5 = countvaluesVARY5[(countvaluesVARY5>=clipmin) & (countvaluesVARY5<=cl

muV5 np.mean(clippedvaluesV5)
sigV5=np.std(clippedvaluesV5)
modeV5=stats.mode(clippedvaluesV5)[0][0]

print("Range:",clipmaxV5 clipmin)
print("Average:",muV5)
print("Stdev:",sigV5)

statuncertainty5   sigV5 / (np.prod(clippedvaluesV5.shape))**(1/2)
print("Stat uncertainty on mean:",statuncertainty5)

sysuncertainty = 0.1
print("Sys uncertainty:",sysuncertainty)
```

```
[redacted] FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  modeV5=stats.mode(clippedvaluesV5)[0][0]
Range: 950
Average: 1085.6002018477866
Stdev: 54.73571341750148
Stat uncertainty on mean: 0.053510107280516556
Sys uncertainty: 0.1
```



120s

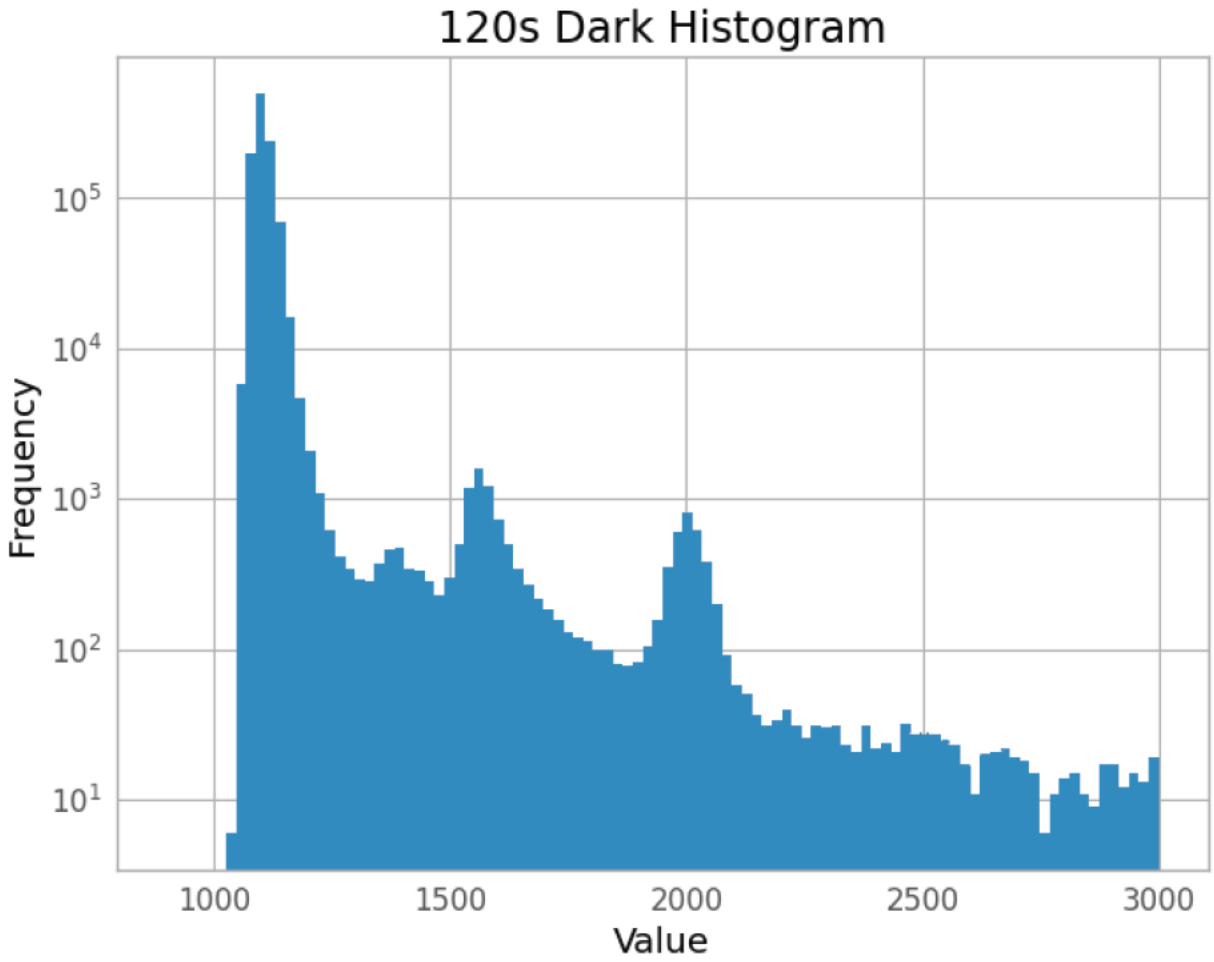
We plot the histogram and display the statistics of the dark frames taken at exposure times. We plot the count values from 900-3000 and take the stats for 900-2250.

In [227…

```
### Histogram and stats of 120s dark
countvaluesVARY6 = imagedataVARY6.flatten()

plt.hist(countvaluesVARY6,bins=100,range=[900,3000])
plt.yscale('log')
plt.title('120s Dark Histogram')
plt.xlabel('Value')
plt.ylabel('Frequency')

clipmaxV6=2250
clippedvaluesV6   countvaluesVARY6[(countvaluesVARY6> clipmin) & (countvaluesVARY6< c

muV6=np.mean(clippedvaluesV6)
sigV6=np.std(clippedvaluesV6)
modeV6 stats.mode(clippedvaluesV6)[0][0]

print("Range:",clipmaxV6-clipmin)
print("Average:",muV6)
print("Stdev:",sigV6)
```

```
statuncertainty6 = sigV6 / (np.prod(clippedvaluesV6.shape))**(1/2)
print("Stat uncertainty on mean:",statuncertainty6)

sysuncertainty = 0.1
print("Sys uncertainty:",sysuncertainty)
```

```
[redacted] FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  modeV6=stats.mode(clippedvaluesV6)[0][0]
Range: 1350
Average: 1112.094888599895
Stdev: 73.82203223107447
Stat uncertainty on mean: 0.07216181667336674
Sys uncertainty: 0.1
```



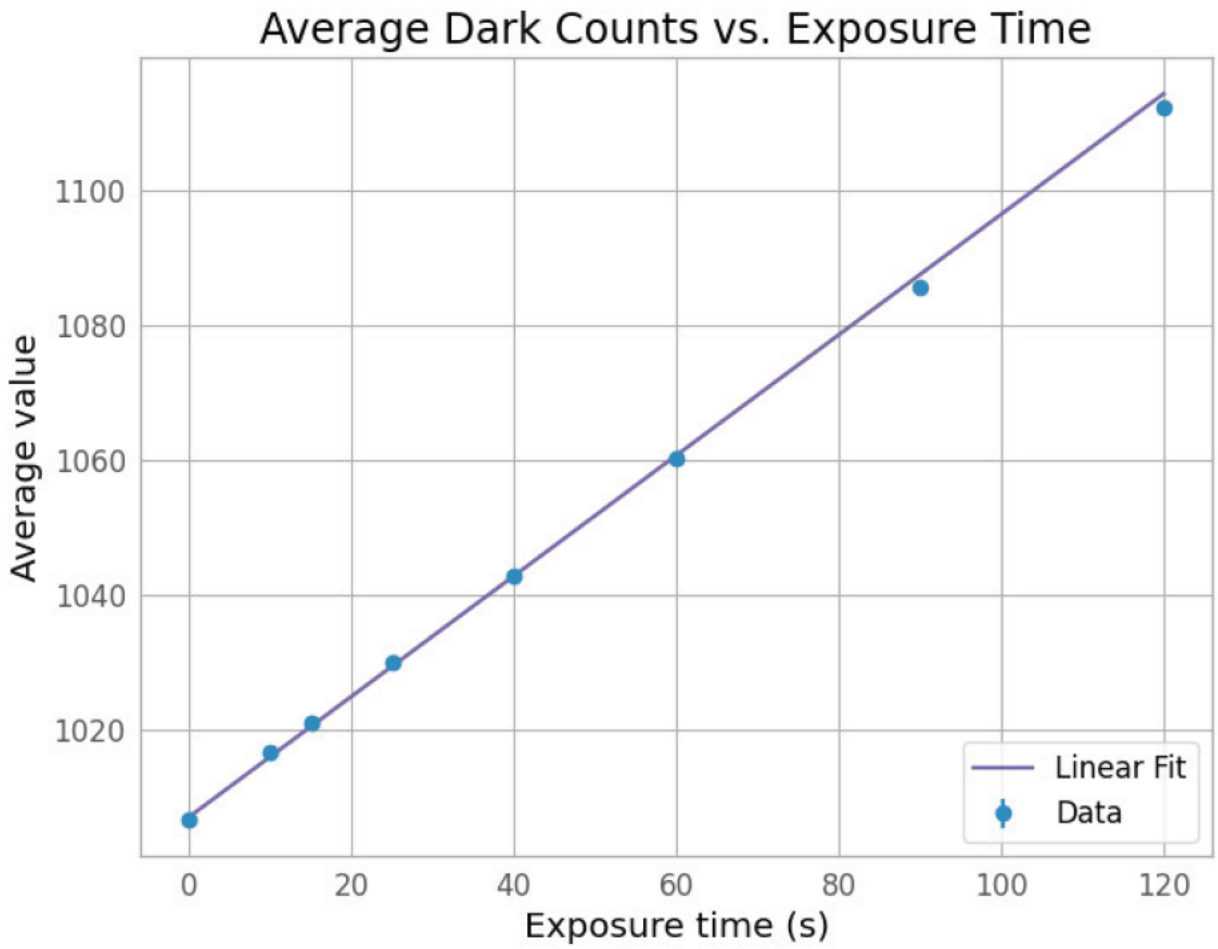
We plot the mean counts vs exposure time, with error bars for the statistical uncertainties on the mean counts. We fit the best-fit linear regression model using the scipy curve_fit function.

In [228…

```
### Plot best fit linear regression

# Create lists of exposure times, mean counts, and error on the means
exptime= np.array([0, 10,15,25,40,60,90,120])
mudata = np.array([mu, muV0, muV1, muV2, muV3, muV4, muV5, muV6])
```

```
sigdata= np.array([statuncertainty, statuncertainty0, statuncertainty1, statuncertaint

# Plot the data points with error bars
plt.errorbar(exptime, mudata, yerr=sigdata,fmt='o')
plt.title('Average Dark Counts vs. Exposure Time')
plt.xlabel('Exposure time (s)')
plt.ylabel('Average value')

# Fit a linear function y = a+b*x to the model
def func(x,a,b):
    return a+b*x

# Extract the optimal fit parameters from the curve_fit() function applied to the data
best vals, covar   curve fit(func, exptime, mudata, sigma sigdata)
a = best_vals[0]
b = best_vals[1]

# Plot the best fit line
x = np.linspace(0, 120, num=100)
yfit = func(x,a,b)
plt.plot(x,yfit,label="fit")

plt.legend(["Linear Fit", "Data"], loc ="lower right")

# Display the best fit parameters
print(f'a   {a:.0f} +/  {covar[0,0]:.0f}')
print(f'b = {b:.3f} +/- {covar[1,1]:.3f}')
```

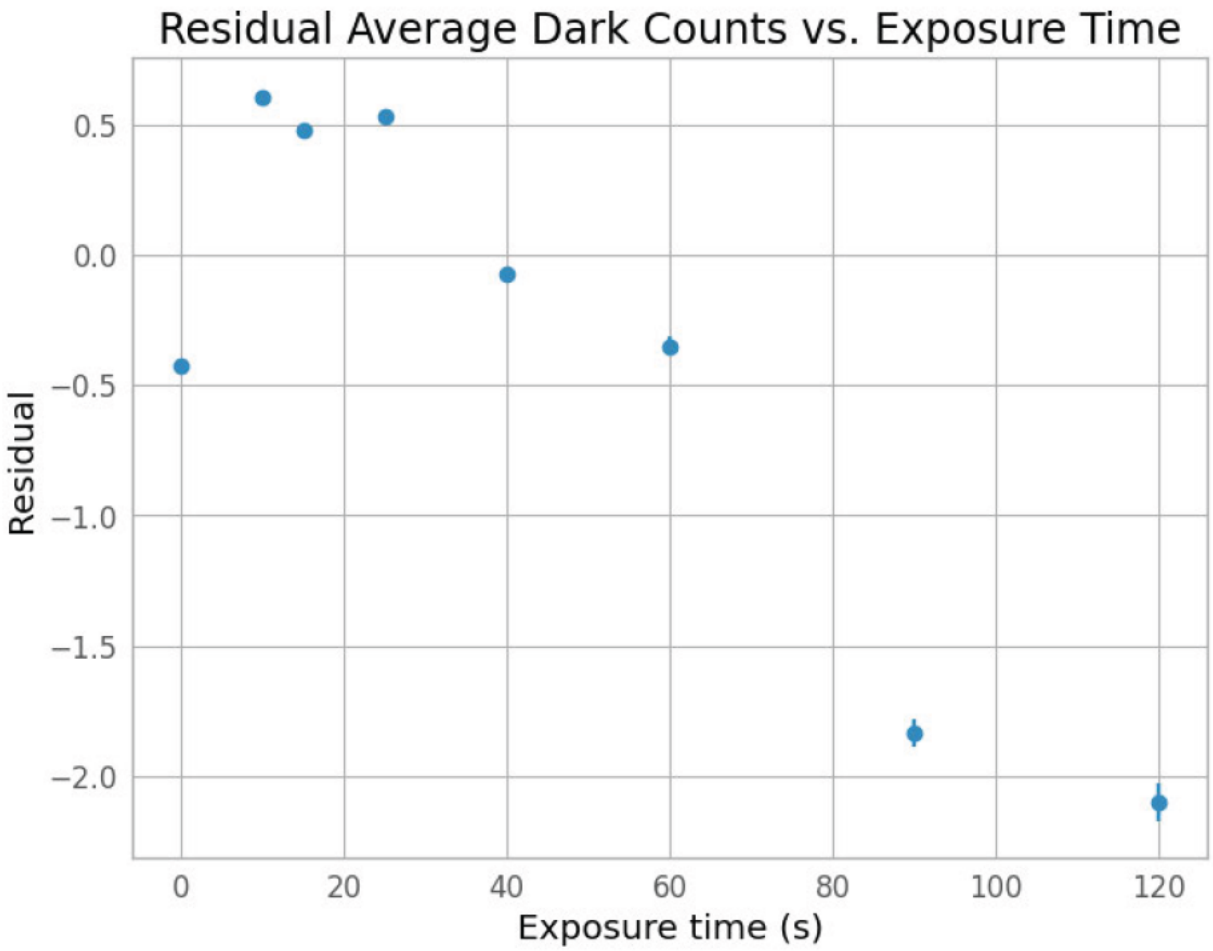
```
a = 1007 +/- 0
b = 0.892 +/- 0.000
```



We plot the residuals:

In [229…
```
plt.errorbar(exptime, mudata-func(exptime,a,b), yerr=sigdata,fmt='o')
plt.title('Residual Average Dark Counts vs. Exposure Time')
plt.xlabel('Exposure time (s)')
plt.ylabel('Residual')
```

Out[229]:
```
Text(0, 0.5, 'Residual')
```



Based on the residual plot, the best-fit line appears to be a good description of the data.

We calculate the dark current (e-/p/s) as the product of the regression slope b and the gain from the image headers. In other words,

$$\text{dark current} = b \times \text{gain}$$

$$\text{dark current} = 0.892 s^{-1} \times 2.06 e^{-}/p$$

In [230…
```
### Determine the dark current from gain
print('Dark current:',b*gain, 'e-/p/s')
```

```
Dark current: 1.837532054738467 e-/p/s
```

Therefore, the dark current is calculated as 1.838 e-/p/s. This is much lower than the 9 e-/p/s as described in the CCD specifications. However, we repeat our calculations for $10°$.

$10°$ bias data

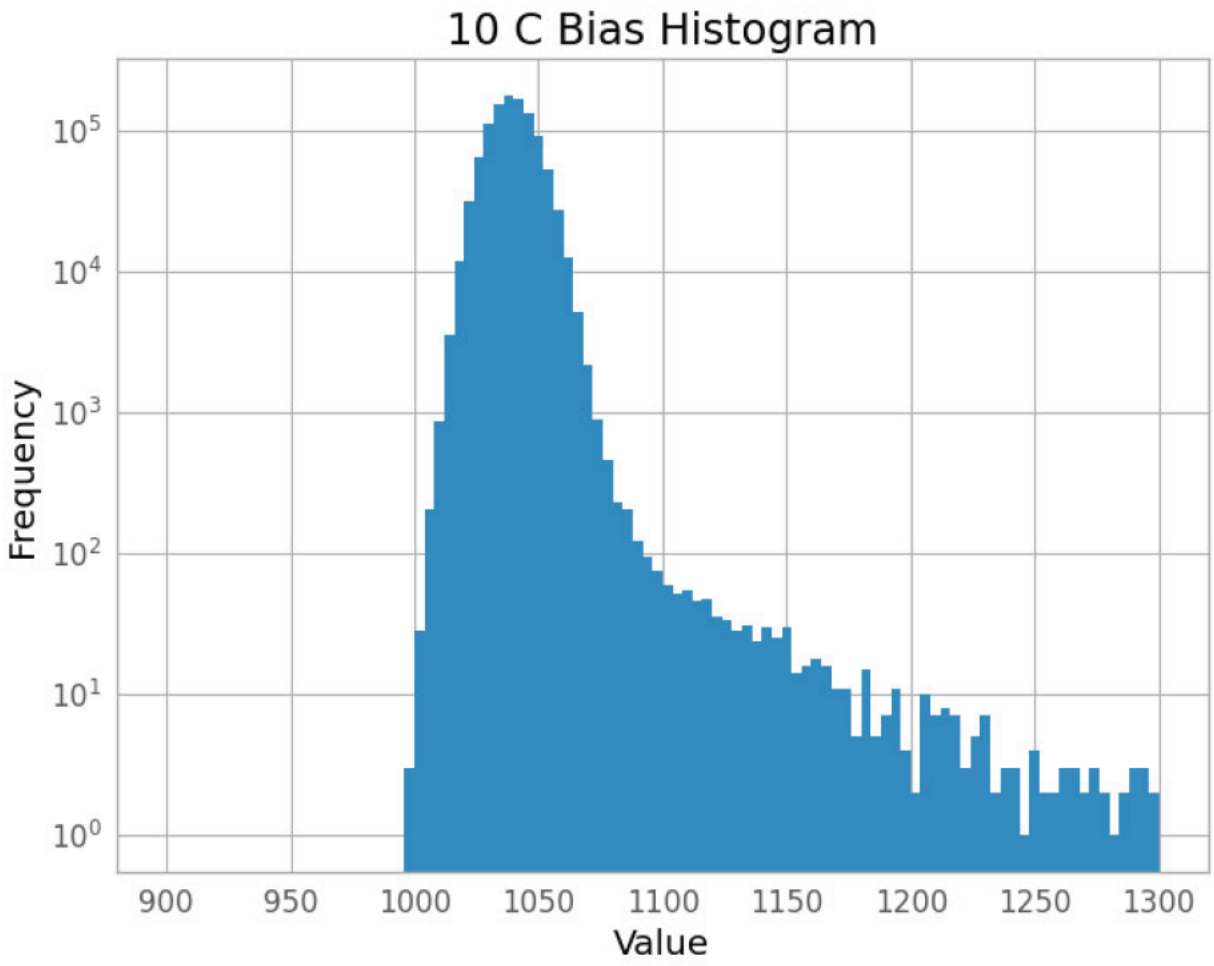
We plot a histogram of the data in the range [900,1300] and take statistics for the cliped range [900,1200]. We display the statistics, once again assuming a systematic uncertainty of 0.1.

In [231…
```
### Repeat for the 10 C Bias

hduhotbias = fits.open('bias10C.00000000.BIAS.FIT')
```

```
imagedatahotflat = hduhotbias[0].data
countvalueshotflat   imagedatahotflat.flatten()
hduhotbias.close()

plt.hist(countvalueshotflat,bins=100,range=[900,1300])
plt.yscale('log')
plt.title('10 C Bias Histogram')
plt.xlabel('Value')
plt.ylabel('Frequency')

clipmaxhotflat=1200
clippedvalueshotflat = countvalueshotflat[(countvalueshotflat>=clipmin) & (countvalues

muhotflat np.mean(clippedvalueshotflat)
sighotflat=np.std(clippedvalueshotflat)
modehotflat=stats.mode(clippedvalueshotflat)[0][0]

print("Range:",clipmaxhotflat clipmin)
print("Mode:",modehotflat)
print("Average:",muhotflat)
print("Stdev:",sighotflat)

statuncertaintyhotflat = sighotflat / (np.prod(clippedvalueshotflat.shape))**(1/2)
print("Stat uncertainty on mean:",statuncertaintyhotflat)

sysuncertainty   0.1
print("Sys uncertainty:",sysuncertainty)
```

```
Range: 300
Mode: 1038
Average: 1039.1447414357199
Stdev: 9.871356427972067
Stat uncertainty on mean: 0.009640961965502393
Sys uncertainty: 0.1
```

```
[redacted] FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  modehotflat=stats.mode(clippedvalueshotflat)[0][0]
```



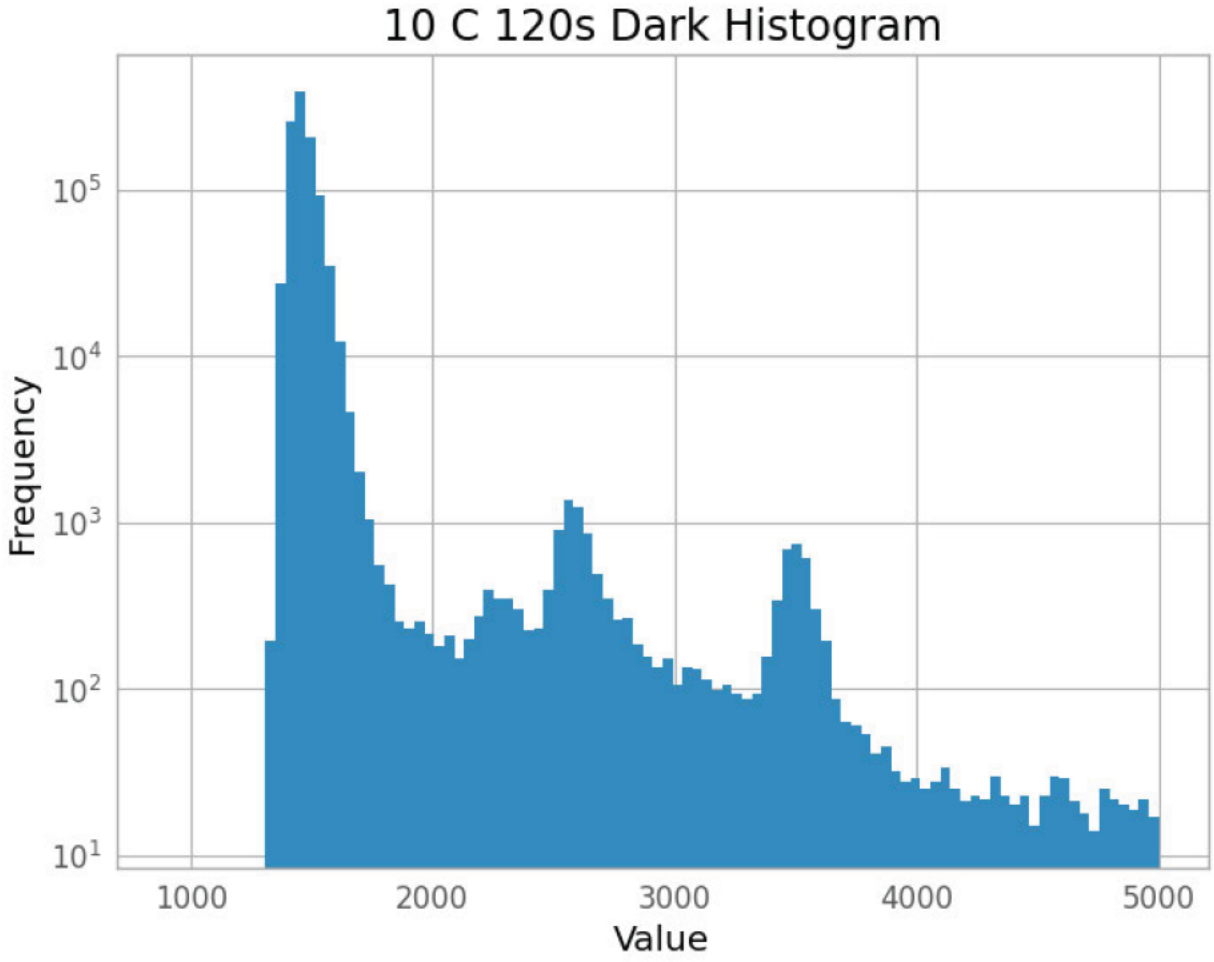
We generate a histogram and statistics for the 10° 120s dark exposure. We plot for the range [900,5000] and take statistics for the range [900,4000].

In [232…

```
### Repeat for the 10 C 120 s exposure
hdudarkhot = fits.open('dark10C120s.00000000.DARK.FIT') # 10 c
imagedatahot   hdudarkhot[0].data
countvalueshot = imagedatahot.flatten()
hdudarkhot.close()

plt.hist(countvalueshot,bins 100,range [900,5000])
plt.yscale('log')
plt.title('10 C 120s Dark Histogram')
plt.xlabel('Value')
plt.ylabel('Frequency')

clipmaxhot=4000
clippedvalueshot = countvalueshot[(countvalueshot>=clipmin) & (countvalueshot<=clipmax

muhot=np.mean(clippedvalueshot)
sighot=np.std(clippedvalueshot)
modehot=stats.mode(clippedvalueshot)[0][0]

print("Range:",clipmaxhot-clipmin)
print("Mode",modehot)
print("Average:",muhot)
print("Stdev:",sighot)
```

```
statuncertaintyhot   sighot / (np.prod(clippedvalueshot.shape))**(1/2)
print("Stat uncertainty on mean:",statuncertaintyhot)

sysuncertainty = 0.1
print("Sys uncertainty:",sysuncertainty)
```

```
[redacted] FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  modehot=stats.mode(clippedvalueshot)[0][0]
Range: 3100
Mode 1442
Average: 1482.8069075718754
Stdev: 173.45222887850622
Stat uncertainty on mean: 0.16956629377977522
Sys uncertainty: 0.1
```



We calculate the dark current as follows:

- We subtract the mean of the bias frame from the mean of the dark frame to calculate the mean pixel count contributed solely by the dark current.

- We divide the mean dark-only count by the exposure time of 120s, yielding a value in units of s^{-1}.
- We multiply this value by the gain of 2.06 e-/p.

In [233…
```
print('Estimated Dark Current:',(muhot-muhotflat)/120*gain,'e-/p/s')
```

```
Estimated Dark Current: 7.61620051867067 e-/p/s
```

The dark current for 10C is 7.616 e-/p/s. It is significantly higher for 10C than for 0C, as expected. 7.62 e-/p/s is closer to the 9 e-/p/s dark current in the STL-1001 specifications. However, the specifications give the dar kcurrent at 0C rather than 10C.

The dark current is higher at a higher temperature because thermal energy of electrons is higher. With a small band gamp in the silicon semiconductor of which the CCD is composed, electrons are more likely to jump into the conduction band and hence produce current. Therefore, as temperature increases, so does dark curent, which is consistent with our observations.

4.3 Imaging Flat Fields

4.3.1

We generate our master flat field by taking the median of several flat fields and taking the median, then dividing the median flat field by its mode.

In [234…
```
### Make the normalized flat
list_of_flats = sorted(glob.glob('flat.0000*.FIT'))
flatsdataA = [fits.open(filename)[0].data for filename in list_of_flats]

### Create a median of all flats, then divide that median by the mode of that median i
medianflat = np.median(flatsdataA,axis=0)
medianflat_norm = medianflat/stats.mode(medianflat.flatten())[0][0]

### Write out the master flat
normflat = fits.PrimaryHDU(medianflat_norm)
normflat.writeto('normflat.fits', overwrite=True)
```

```
[redacted] FutureWarning: Unli
ke other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode
` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will cha
nge: the default value of `keepdims` will become False, the `axis` over which the sta
tistic is taken will be eliminated, and the value None will no longer be accepted. Se
t `keepdims` to True or False to avoid this warning.
  medianflat_norm = medianflat/stats.mode(medianflat.flatten())[0][0]
```

4.3.2

On average, the center receives more counts per pixel than the edges, and is hence more sensitive. The "donuts" in the flat field are caused by light interfering with dust particles.

The brightest central pixels of the master flat have an average of 1.03 counts. The boundary-region pixels have an average of 0.96 counts. Therefore, the dimmest regions get

$0.96/1.03 = 0.932$,

or 93.2% of the counts of the brightest regions.

4.3.3

The star will appear brighter in the center of the image than at the edges of the image. Therefore, the star will be biased to have a lower magnitude in the center of the image than at the edges. This is because the center pixels are more sensitive than the edge pixels.

A star placed in the center of the image, then to one of the corner pixels, would have its "observed" magnitude increase.

Pixels are of uniform size, so counts ratio can be considered equivalent to flux ratio for the magnitude calculation. Namely,

$\frac{f_1}{f_2} = \frac{\text{counts}_1}{\text{counts}_2}$

We calculate the increase in magnitude as follows, using a flu value of $f_1 = 1.03$ for the center and a flux value of $f_2 = 0.96$ for the corner. Using the magnitude equation we calculate that:

$m_1 - m_2 = -2.5 \log\left(\frac{f_1}{f_2}\right)$

$m_1 - m_2 = -2.5 \log\left(\frac{1.03}{0.96}\right)$

$m_1 - m_2 = -2.5 \log(1.0729)$

$m_1 - m_2 = -0.1759$

$m_2 - m_1 = 0.1759$

where m_2 is the magnitude of the corner and m_1 is the magnitude of the center. Therefore, the observed magnitude of the star increases by 0.1759 when moved from the center to the corner of the image.

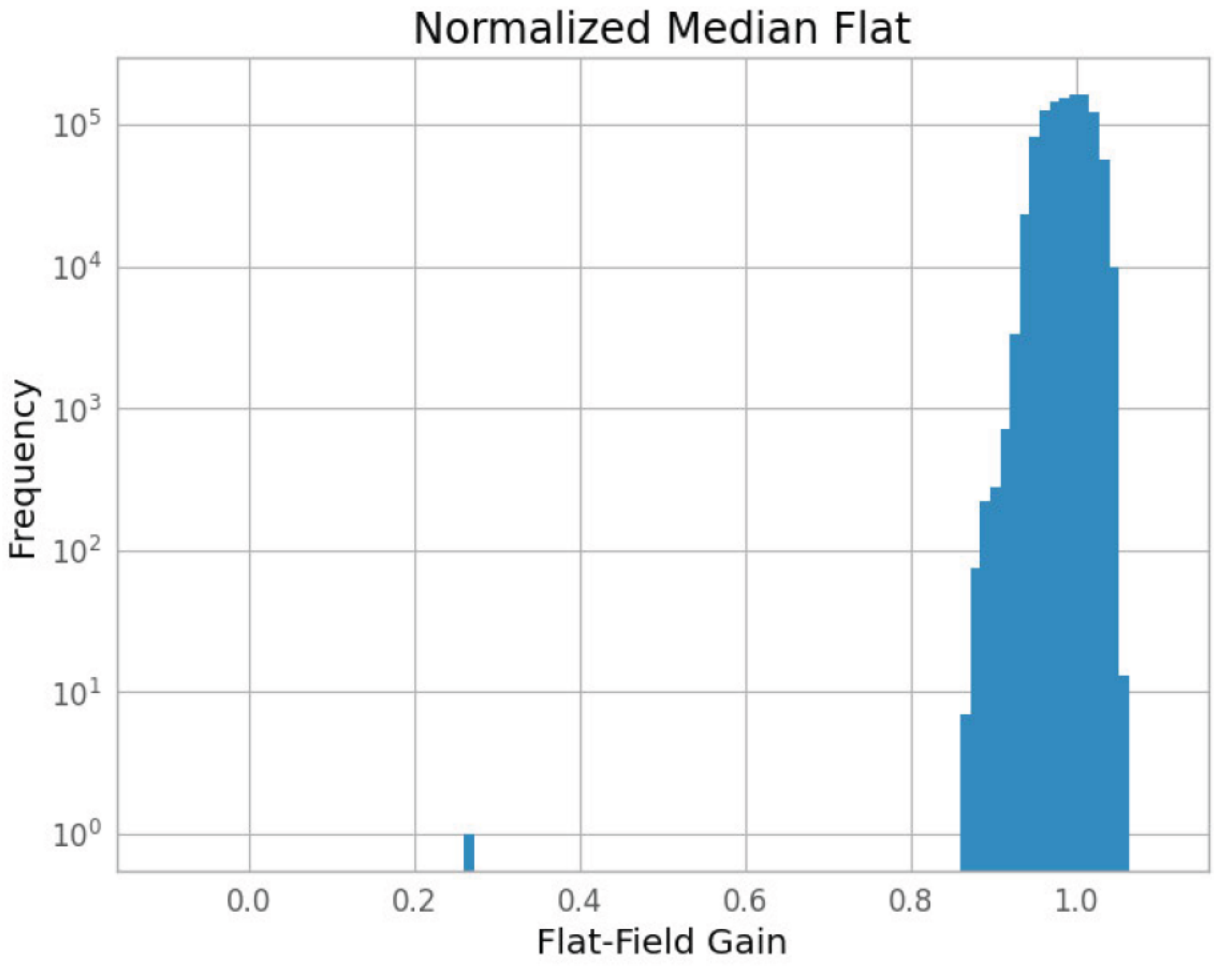
4.3.4

We plot a histogram of the master flat values.

In [235…
```
### Histogram of the normalized median flat.
normflat_cntvals=medianflat_norm.flatten()
```

```
plt.hist(normflat_cntvals,bins=100,range=[-0.1,1.1]); # Range has been checked, this i
plt.yscale("log")
plt.title('Normalized Median Flat')
plt.xlabel('Flat-Field Gain')
plt.ylabel('Frequency')
```

```
Out[235]: Text(0, 0.5, 'Frequency')
```



We can see 1 dead pixel. It has significantly less sensitivity than the other pixels.

4.3.5

If we forgot to take flat fields on the night of observation, we cannot retake them at a later date. The distribution of dust particles on the CCD largely contributes to the flat field. It is exceedingly unlikely to replicate the same dust particle distribution between different observing nights, so the flat field corresponding to one night cannot be taken at a later date if forgotten.

4.4 Spectrographic Calibration

4.4.1

The bottom slit of the spectroscopic flat-field is the 50μ slit. This is because it has the thickest spectral lines.

4.4.2

We open the spectroscopic flat field.

In [236…
```
### Open the spectroscopy flat
os.chdir('../Spectroscopy CCD')

hduspecflat = fits.open('flat.00000000.FIT')

imagespecflat = hduspecflat[0].data

print(imagespecflat)

hduspecflat.close()
```

```
[[ 187  200  216 ...  169  158  214]
 [ 201  158  200 ...  247  209  226]
 [ 204  206  181 ...  193  215  210]
 ...
 [5005 4816 4734 ... 6571 6955 6796]
 [4834 4843 4745 ... 6554 6916 6700]
 [4690 4754 4777 ... 6783 6788 6707]]
```

We observe that the flat-field image is in the format [[row0] [row1] . . . [row255]].

Checking in ds9, we note that 0,0 corresponds to the bottom-left pixel, and that the x- and y-values increase as one moves up or right in the image.

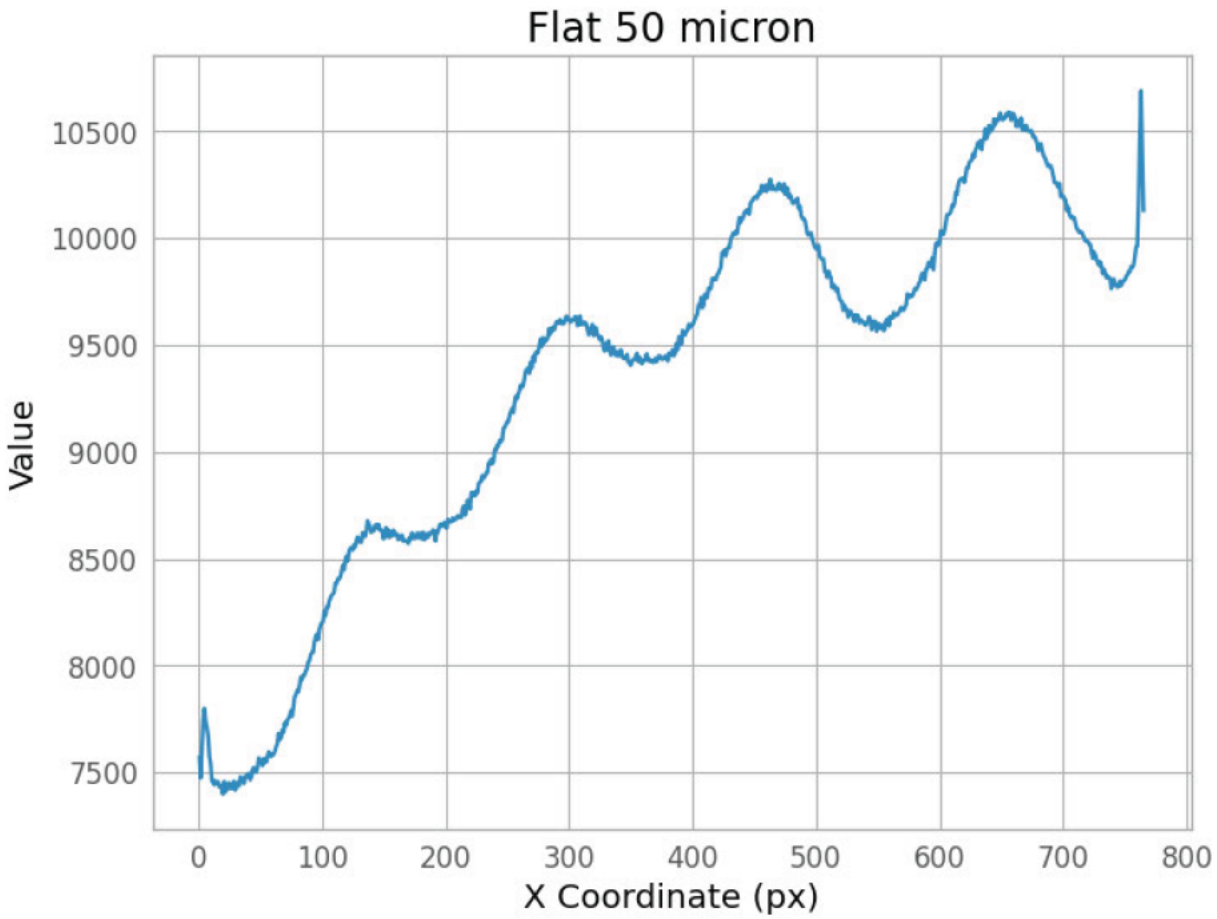
We splice the flat field between rows 23-83 to only include the 50μ slit. We take the median of each column in the 50μ slit section.

In [237…
```
### Data manipulation
spliceimagespecflat = imagespecflat[23:83] # splice to 50 micron rows. Inspect the 3rd

transspliceimagespecflat = np.transpose(spliceimagespecflat) # transpose to get [[colu

# Take the median of each column in the 50 micron segment of the flat-field
medianarray = np.median(transspliceimagespecflat, axis = 1) #median all axis 1 arrays
```

We plot the column median values with respect to x-position.

In [238…
```
### Plot the median of each column of the 50 micron slit
specxvalues = np.arange(765)+1

plt.plot(specxvalues, medianarray)
plt.title('Flat 50 micron')
plt.xlabel('X Coordinate (px)')
plt.ylabel('Value')
```

Out[238]:
```
Text(0, 0.5, 'Value')
```



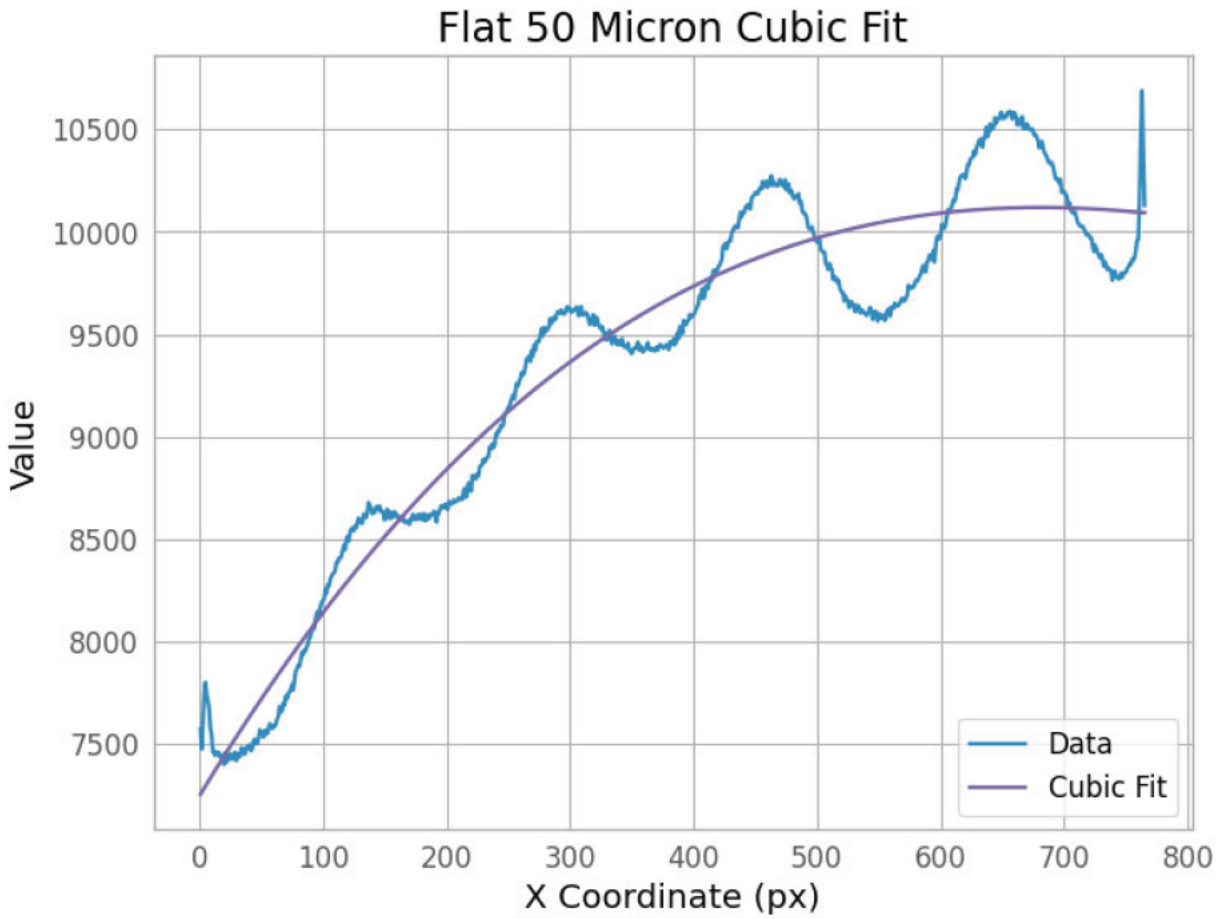
4.4.3

We normalize the flat-field by fitting a polynomial to the 1D spectrum above and then dividing the 2D flat-field by the fitted polynomial.

In [239…
```
### Find a best fit with a cubic fit
polyfit = np.polyfit(np.arange(765)+1, medianarray, 3)
print(polyfit)

### Evaluate the function at integers from 1 to 765 for the plot
ypolyfit_list = [polyfit[0]*x**3 + polyfit[1]*x**2 + polyfit[2]*x + polyfit[3] for x i

### Plot both the data and the best fit
plt.plot(specxvalues,medianarray)   # data
plt.plot(specxvalues,ypolyfit_list) # best fit
plt.title('Flat 50 Micron Cubic Fit')
plt.xlabel('X Coordinate (px)')
plt.ylabel('Value')
plt.legend(["Data", "Cubic Fit"], loc ="lower right")
```

```
[ 3.32421685e-06 -1.07292749e-02  9.98724089e+00  7.24372415e+03]
```

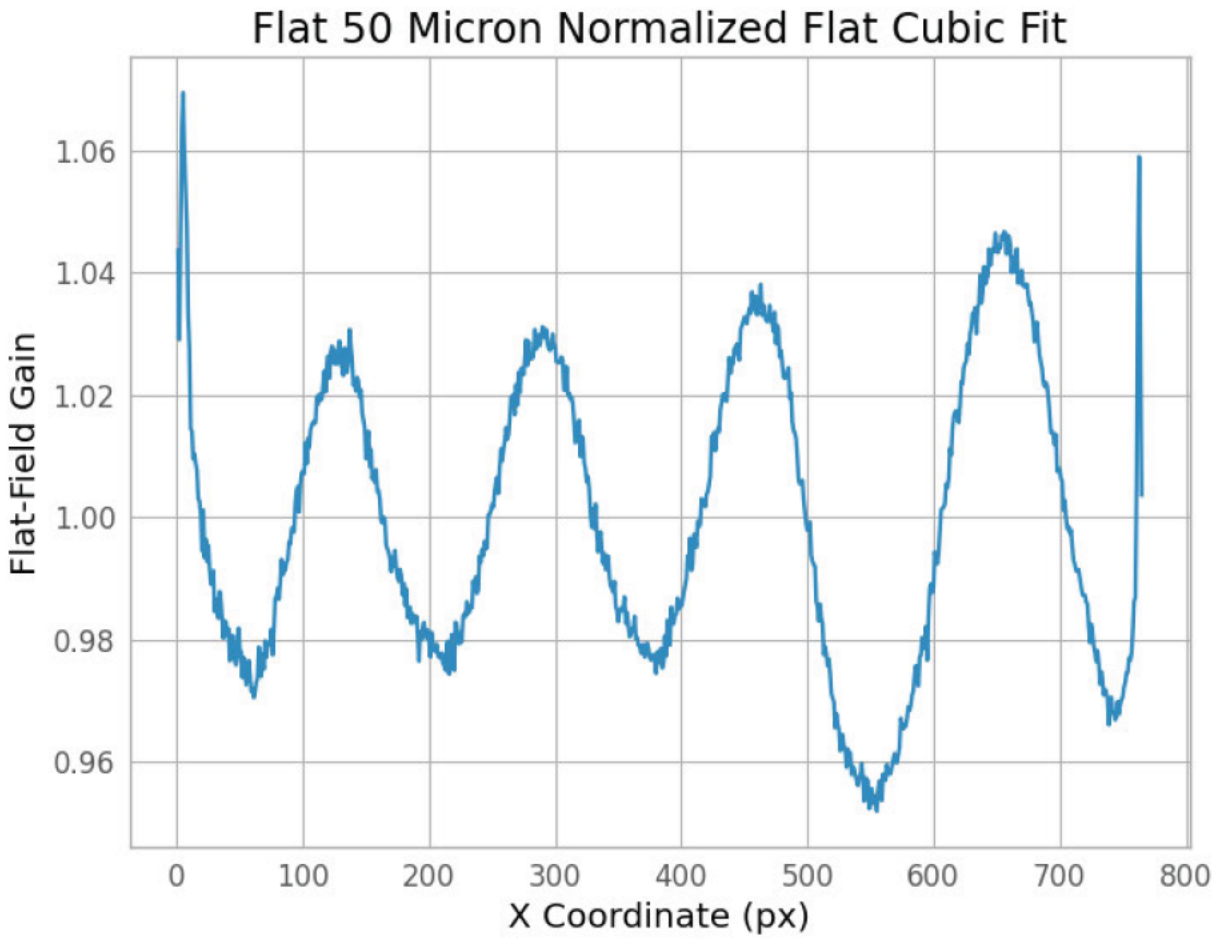
Out[239]:
```
<matplotlib.legend.Legend at 0x24d07254f70>
```



Here is what the 1D version of the normalized flat-field would look like:

In [240…
```
# not needed for report
norm_y_polyfit_list = np.divide(medianarray, ypolyfit_list)

plt.plot(specxvalues,norm_y_polyfit_list)
plt.title('Flat 50 Micron Normalized Flat Cubic Fit')
plt.xlabel('X Coordinate (px)')
plt.ylabel('Flat-Field Gain')
```

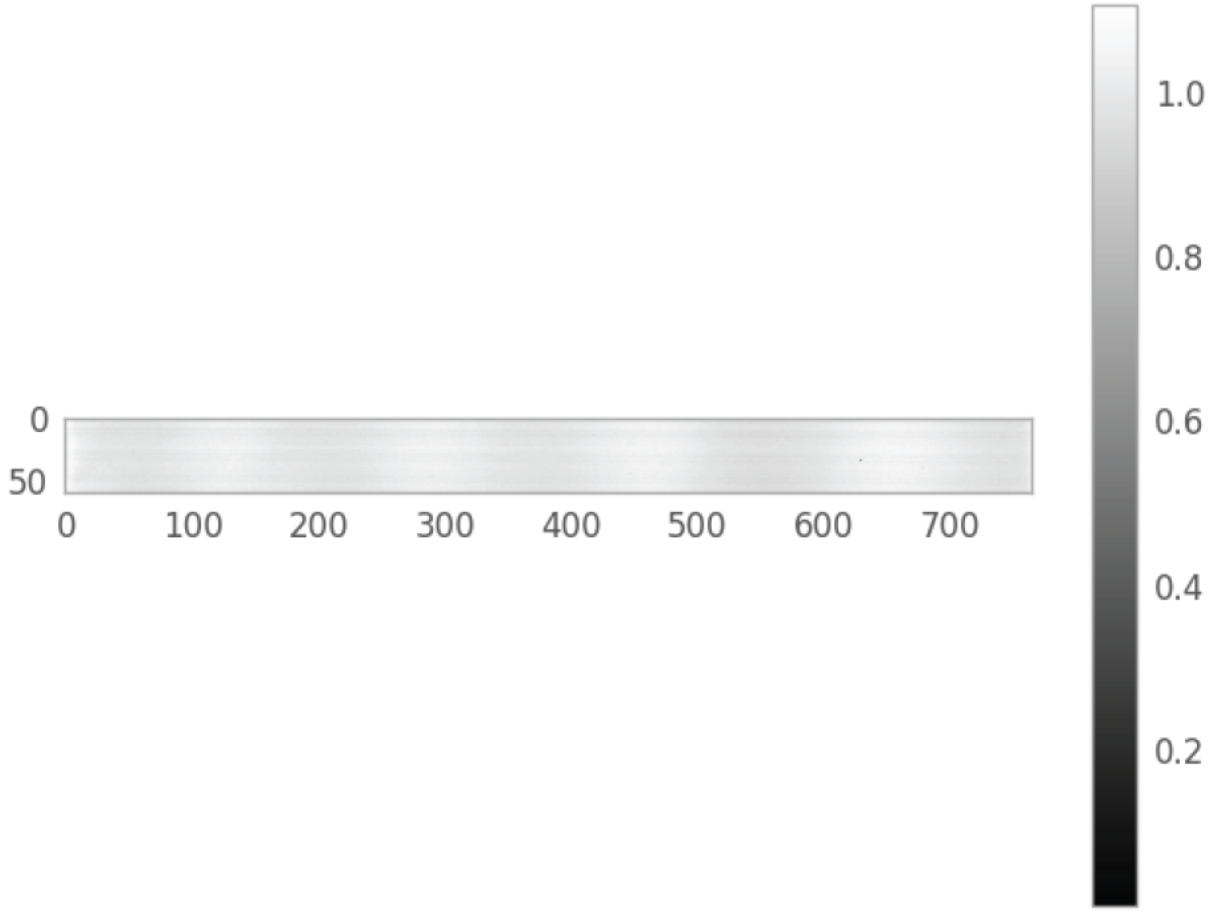
Out[240]:
```
Text(0, 0.5, 'Flat-Field Gain')
```



We normalize our 2D flat field by copying the 1D normalization polynomial into 60 rows and then normalizing over the resulting 2D array.

In [241…

```
### Normalized master flat field

# Copy normalization polynomial into 60 rows
norm_y_polyfit_list_2d = np.tile(ypolyfit_list, (60,1)) #[median row] -> [[med row][me

# Divide 2D master flat over the 60-row duplicated normalization polynomial
masterflat50 = spliceimagespecflat/norm_y_polyfit_list_2d #Divide 3rd slit flat by arr

# Generate output file
masterflat50micron = fits.PrimaryHDU(masterflat50)
masterflat50micron.writeto('masterflat50micron.fits', overwrite='overwrite')

plt.figure()
plt.grid(False)
plt.imshow(masterflat50, cmap='gray')
plt.colorbar()
plt.autoscale()
```



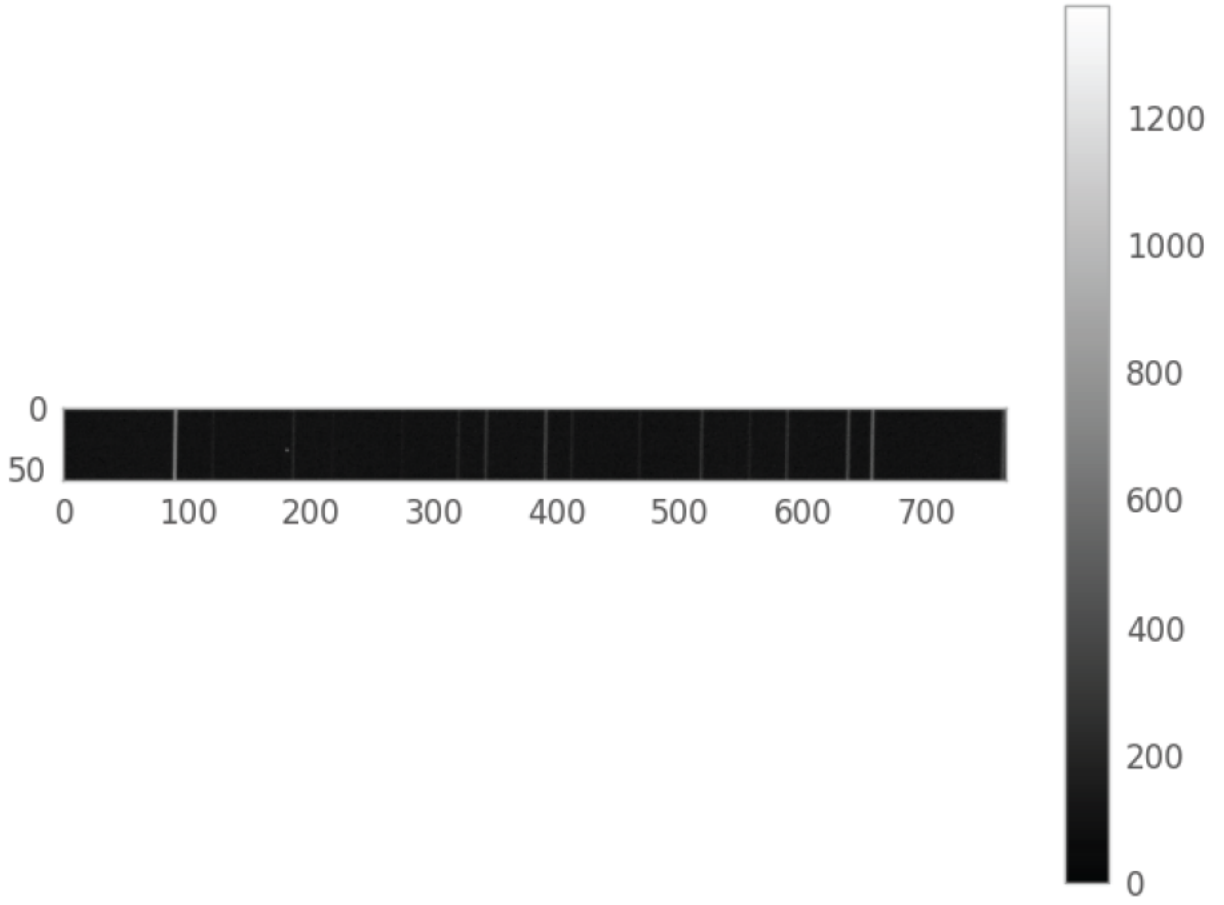
The masterflat removes the gradient from dimmer on left to brighter on right (spectrum) and leaves the wiggles (spectrograph response). We also notice row-to-row sensitivity differences.

4.4.4

We cut out the exact same rows in our science image and normalize it by dividing over the normalized flat field.

In [242…

```
### Calibrate our science images
hduspec0 = fits.open('spec.00000000.FIT')
imagespec0 = hduspec0[0].data
hduspec0.close()

spliceimagespec0 = imagespec0[23:83] #Splice the science image as necessary
calibsci0 = spliceimagespec0/masterflat50 # Divide by our masterflat50

# Generate output science image
calibscience0_50micron = fits.PrimaryHDU(calibsci0)
calibscience0_50micron.writeto('calibscience50micron_0.fits', overwrite='overwrite')

# Plot calibrated science image
plt.figure()
plt.grid(False)
plt.imshow(calibsci0, cmap='gray')
plt.colorbar()
plt.autoscale()
```



4.4.5

We derive the wavelength calibration from the arc lamp spectrum by identifying emission lines and pixel positions (x-coordinates).

The x-coordinates are obtained from visual inspection of the calibrated science image, and the emission line wavelengths are obtained from Lecture 6.

In [243…
```
### Derive the wavelength calibration

### x-coordinates from visual inspection; lambda coordinates from Lecture 6
xcoordarray = [92, 122.5, 187.5, 220, 276, 321, 343, 391.5, 413, 467.5, 518, 557, 587,
lambdaarray = [5852.49, 5881.89, 5944.83, 5975.53, 6030.00, 6074.34, 6096.16, 6143.06,
```

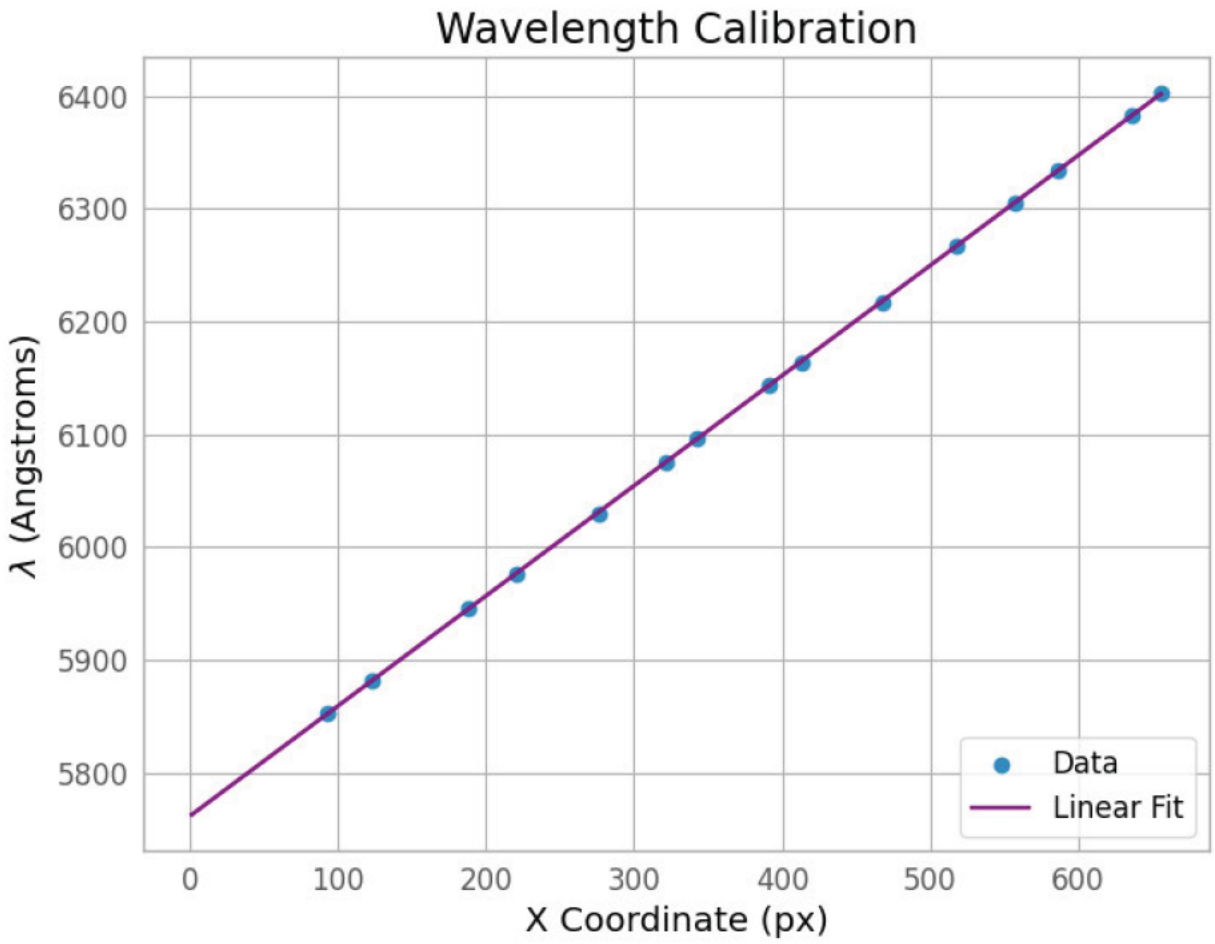
We plot the linear fit for wavelength (Angstroms) vs position (x-coordinate).

In [244…
```
### Linear fit for wavelength vs position
calib_polyfit = np.polyfit(xcoordarray, lambdaarray, 1)
print(calib_polyfit)

### Plot the emission lines and fit
plt.scatter(xcoordarray,lambdaarray)
plt.plot(np.arange(656)+1,[calib_polyfit[0]*x + calib_polyfit[1] for x in range(0+1,65
plt.title('Wavelength Calibration')
plt.xlabel('X Coordinate (px)')
```

```
plt.ylabel('$\lambda$ (Angstroms)')
plt.legend(["Data", "Linear Fit"], loc ="lower right")
```

```
[9.75423614e-01 5.76162454e+03]
```

Out[244]: <matplotlib.legend.Legend at 0x24d109dce20>



With a slope of 0.975, it appears that each pixel corresponds to a change of 0.975 Angstroms. Pixel 1 corresponds to to 5761.62 Angstroms, and the wavelength increases to the right along the pixels.

4.4.6

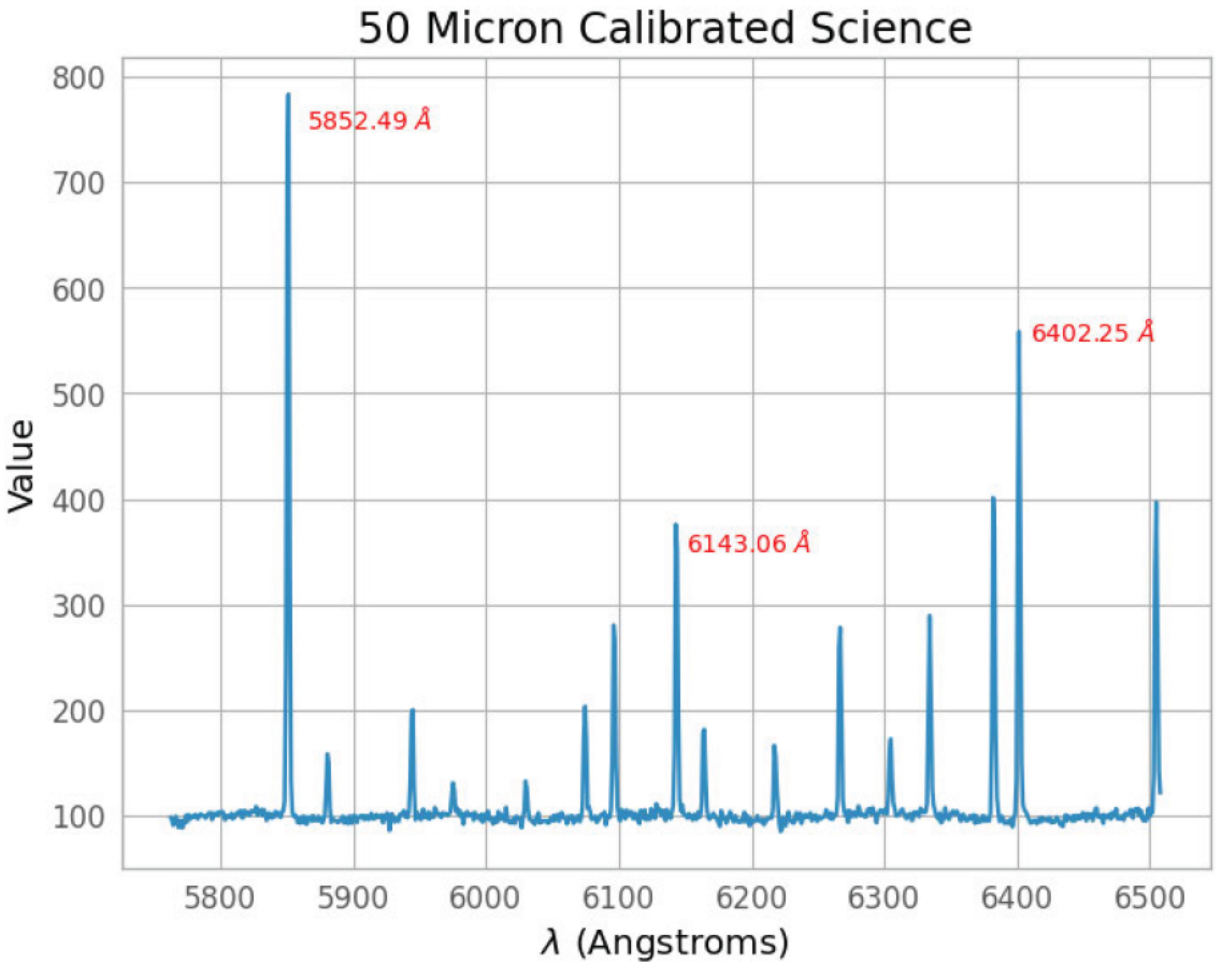
We plot the calibrated emission spectrum and label 3 prominent emission lines.

In [245…

```
### Plot the counts as a function of wavelength

# transpose calibrated science values to get [[col0] [col1] ... [col59]]
transcalibsci0 = np.transpose(calibsci0)

# take the median of every row in the calibrated science image
medianarraysci0 = np.median(transcalibsci0, axis = 1) #median all columns to get [med

# Plot the labels
plt.text(5865, 750, '5852.49 $\AA$', fontsize=10, c = 'red')
plt.text(6150, 350, '6143.06 $\AA$', fontsize=10, c = 'red')
plt.text(6410, 550, '6402.25 $\AA$', fontsize=10, c = 'red')
```

```
# Plot emission lines vs wavelength, where x-axis is corrected for dispersion (pixel r
plt.plot(specxvalues*calib_polyfit[0]+calib_polyfit[1], medianarraysci0) # x axis is c
plt.title('50 Micron Calibrated Science')
plt.xlabel('$\lambda$ (Angstroms)')
plt.ylabel('Value')
```

Out[245]: Text(0, 0.5, 'Value')



4.4.7

We calculate the spectrograph length and dispersion as follows:

- The spectrograph dispersion is 0.975 Angstroms per pixel.
- The spectrograph length is equal to the width of the image (765 pixels) multiplied by the dispersion (0.975 Angstroms/pixel), or 746.21 Angstroms.

In [246…

```
### Determine spectrograph length and dispersion
print("Spectrograph length: ", calib_polyfit[0]*765, 'Angstroms')
print("Spectrograph dispersion: ", calib_polyfit[0], 'Angstroms')
```

```
Spectrograph length:  746.1990644435748 Angstroms
Spectrograph dispersion:  0.9754236136517317 Angstroms
```

4.5 Calibration strategies

There are 3 types of calibration exposures: bias frames, dark fields, and flat fields.

- The bias frame shows the readout noise of each pixel. Dark frames already account for this.
- Dark frames are taken over the same time interval as each science exposure but without light exposure, and show the dark current of each pixel. The dark frames are combined into a median master dark frame that is then subtracted from the science images to remove dark current and bias current.
- Flat fields are taken over the same time interval as each science exposure but exposed to a uniform light source. Flat fields reveal that pixels have varying sensitivities based on position and dust particles on the CCD. Pixels in the center of the image tend to be more sensitive than pixels on the boundary. After subtracting the master dark, the science image is then divided by the master flat to compensate for differing pixel sensitivities.

All calibration exposures have to be taken with the same telescope and instrument setup as the observations. Changes in focus also have a negligible effect when looking at stars effectively at infinity.

- The bias frames and dark fields can be taken the next day as needed, as they involve no light exposure and hence are not dependent on current local lighting conditions.
- The flat field needs to be taken on the same day with the same observation setup, since it depends on the placement of dust particles on the CCD sensor. It is theoretically possible but exceedingly unlikely to have the same dust placement on the CCD sensor between 2 different observing days.