EATS PL Report - Telescopes

Template by André Moitinho

Student number: 49807

Name: Ricardo Matoza Pires

Instructions

- Fill in your name and number above
- This template provides blocks of instructions that guide you through the activity and report. Put your codes and comments below the corresponding blocks.
- Add as many code and markdown cells as necessary.
- The blue background colour in the markdown cells (like this one) is reserved for the instruction blocks. Use white cells for your comments.
- Write a clear report:
  - Provide justifications and comments.
  - Figures and labels (legends, axes, etc) should be easy to read.
  - Tables should be neatly formatted and include labels.

To submit your report:

You will need to have the nbconvert[webpdf] package installed for saving to pdf.
Anaconda: conda install -c conda-forge nbconvert-webpdf
General: pip install nbconvert[webpdf]

- Clean up and make sure all works well: In the "Kernel" menu select "Restart & Run All" (or use the "fast forward/double arrow" icon). Wait until the process is completed
- Save to a pdf file:
  - JupyterNotebook users: In the "File" menu -> "Download As" -> "**PDF via HTML (.html)**".
  - JupyterLab users: In the "File" menu -> "Save and Export Notebook As" -> "**Webpdf**".
- Check that the PDF file is fine (no truncated lines, nothing missing, etc)
- Name the file: NumberFirstnameLastname.pdf* (e.g. 10101ClaudeShannon.pdf)
- Upload the pdf file to the "PL Report - Telescopes" assignement area.

Tips

- Run a cell - keyboard shortcut: shift-enter
- To create "markdown" cells like this one, For comments, not for code: create a new cell. In the menu above to the right of the "fast-forward" icon there's a dropdown button with the default value "Code". Change to "Markdown".
- To display an image from a file on disk (e.g. image.png): In a **markdown cell** you can use *`![some text](image.png)`*, or simply drag and drop!

Setup

Put all your imports (e.g. numpy,matplotlib, etc) in the cell bellow

In [ ]:

```
from pathlib import Path

import numpy as np
import matplotlib.pyplot as plt

## Astropy
from astropy.nddata import CCDData

from astropy.stats import mad_std
from astropy.modeling import models
from astropy.io import fits

## ccdproc
```

```python
import ccdproc as ccdp
from ccdproc import ImageFileCollection

## Avoid warning messages
import warnings
warnings.filterwarnings('ignore')
```

PART 1 - KNOWING YOUR DATA

Basic image information

For the set of fits files in the *0_mainover* folder, create a table with: image_name, filter exposure_time, imagetype, object_name. Show:

- the command line you used for this
- the table (use the "pandas" package and display the column names at the top)

In [ ]:
```python
## Set-up main data path and files
mainData_path = Path('0_mainData')
mainData_files = ImageFileCollection(mainData_path)
```

In [ ]:
```python
## Images info
mainData_files.summary['file', 'filter', 'exptime', 'imagetyp', 'object']
```

Out[ ]: *Table length=19*

file	filter	exptime	imagetyp	object
str10	str7	int64	str6	str8
3241o.fits	I	5	OBJECT	NGC2420
3249o.fits	I	90	OBJECT	NGC2420
3262o.fits	V	10	OBJECT	NGC2420
3265o.fits	V	180	OBJECT	NGC2420
4015b.fits	UNKNOWN	1	BIAS	UNKNOWN
4016b.fits	UNKNOWN	1	BIAS	UNKNOWN
4017b.fits	UNKNOWN	1	BIAS	UNKNOWN
4075f.fits	V	3	FLAT	Twilight
4076f.fits	V	3	FLAT	Twilight
4078f.fits	V	3	FLAT	Twilight
4079f.fits	V	3	FLAT	Twilight
4080f.fits	V	3	FLAT	Twilight
4081f.fits	I	7	FLAT	Twilight
4082f.fits	I	7	FLAT	Twilight
4083f.fits	I	7	FLAT	Twilight
4084f.fits	I	7	FLAT	Twilight
4085f.fits	I	7	FLAT	Twilight
4086f.fits	I	12	FLAT	Twilight
4087f.fits	I	12	FLAT	Twilight

Determine the CCD gain and readout noise:

For this, follow the recipe given in Howell p.71-73.

In the *gainNoise* folder, use images bias1, bias2, domeV1 and domeV2 to determine the readout noise and gain of the CCD. Don't forget to select an appropriate section of the image for the calculations (don't use the overscan section).
Update the CCD header of your images with the values you have just determined.

Verify in one image header that the values have indeed been updated.

- Show the updated header

In [ ]:
```python
## Set-up the gain noise images
gainNoise_path = 'gainNoise'
```

```python
gainNoise_files = ImageFileCollection(Path(gainNoise_path))

## Set-up path to write processed gain noise images
gainNoise_ProcPath = Path('ProcGainNoise')
gainNoise_ProcPath.mkdir(exist_ok=True)
```

In [ ]:
```python
## Subtract overscan, trim overscan and borders, and compute average
results = []
files = []
for ccd, file_name in gainNoise_files.ccds(ccd_kwargs={'unit': 'adu'}, return_fname=True):
    # Subtract the overscan
    ccd = ccdp.subtract_overscan(ccd, fits_section='[1030:1068,:]', median=True, overscan_axis=1)
    # Trim the overscan and borders
    ccd = ccdp.trim_image(ccd, fits_section='[10:1020, 10:1020]')
    # Save average values to array
    results.append(float(np.asarray(ccd.mean())))
    # Save processed files
    ccd.write(str(gainNoise_ProcPath)+'/OT_'+file_name, overwrite=True)
    # Save file names to array
    files.append('/OT_'+file_name)

B1_mean = results[0]
B2_mean = results[1]
F1_mean = results[2]
F2_mean = results[3]

print(results)
```

```
[7.79431691551196, 7.73554990064777, 19207.62623114093, 19388.97778051718]
```

In [ ]:
```python
## Subtract bias and flats
hdu_bias1 = fits.open(str(gainNoise_ProcPath)+'/'+files[0])
hdu_bias2 = fits.open(str(gainNoise_ProcPath)+'/'+files[1])
hdu_flat1 = fits.open(str(gainNoise_ProcPath)+'/'+files[2])
hdu_flat2 = fits.open(str(gainNoise_ProcPath)+'/'+files[3])

data_bias1 = hdu_bias1[0].data
data_bias2 = hdu_bias2[0].data
data_flat1 = hdu_flat1[0].data
data_flat2 = hdu_flat2[0].data

b1_2 = data_bias1 - data_bias2
f1_2 = data_flat1 - data_flat2
```

In [ ]:
```python
## Compute standard deviations
sigma_b = np.std(b1_2, ddof=1)
sigma_f = np.std(f1_2, ddof=1)

## Compute gain and noise
gain = (F1_mean+F2_mean-B1_mean-B2_mean)/(sigma_f**2-sigma_b**2)
print('Gain = ', f'{gain:.2f}')
read_noise = gain*sigma_b/np.sqrt(2)
print('Read Noise = ', f'{read_noise:.2f}')
```

```
Gain =  1.28
Read Noise =  6.33
```

In [ ]:
```python
## Update images' headers with the previously determined gain and RN

for ccd, file_name in mainData_files.ccds(return_fname=True):

    ccd.header['GAIN'] = float(f'{gain:.2f}')
    ccd.header['RDNOISE'] = float(f'{read_noise:.2f}')
    ccd.write(str(mainData_path)+'/'+file_name,overwrite=True,output_verify='ignore')
```

In [ ]:
```python
## Check it was updated
ccd.header
```

Out[ ]:
```
SIMPLE  =                    T / conforms to FITS standard
BITPIX  =                   16 / array data type
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                 1072
NAXIS2  =                 1024
ORIGIN  = 'Copyright (C) 1991-1998 GKR Computer Consulting' / FITS file originat
DATE    = '2001-06-05T23:49:49' / Date FITS file was generated
IRAF-TLM= '15:49:48 (05/06/2001)' / Time of last modification
EXPTIME =                   12
FILTER  = 'I       '
IMAGETYP= 'FLAT    '
CCDSUM  = '1 1     '
CCDSEC  = '[1:1024,1:1024]'
DATASEC = '[1:1024,1:1024]'
BIASSEC = '[1025:1072,1:1024]'
TRIMSEC = '[1:1024,1:1024]'
OBJECT  = 'Twilight'
RA      = 'UNKNOWN '
DEC     = 'UNKNOWN '
EPOCH   =               0000.0
UT      = '00:56:17'
DATE-OBS= '2001-02-22'
OBSERVAT= 'SPM     '
TELESCOP= '0.84    '
LATITUDE= '+31:02:39'
LONGITUD= '-115:27:49'
ALTITUDE=                 2800
OBSERVER= 'Moitinho'
INSTRUME= 'la cubeta'
DETECTOR= 'SITe1 1k'
GAINMODE=                    4
GAIN    =                 1.28
RDNOISE =                 6.33
BUNIT   = 'adu     '
BSCALE  =                    1
BZERO   =                32768
HISTORY  PMIS macros of 2001-01-23
HISTORY  Written by Stephen Levine
HISTORY  Modified by Gaguik Tovmassian
HISTORY  Modified by Alan Watson & Michael Richer
```

Tracking the bias level - I

Display one of the bias images in DS9. Remember, they are also called “zero” images. You can identify them by checking the list created above.

In the “scale” menu on the top bar, uncheck the “use DATASEC” option. You will now be able to also see the overscan strip.

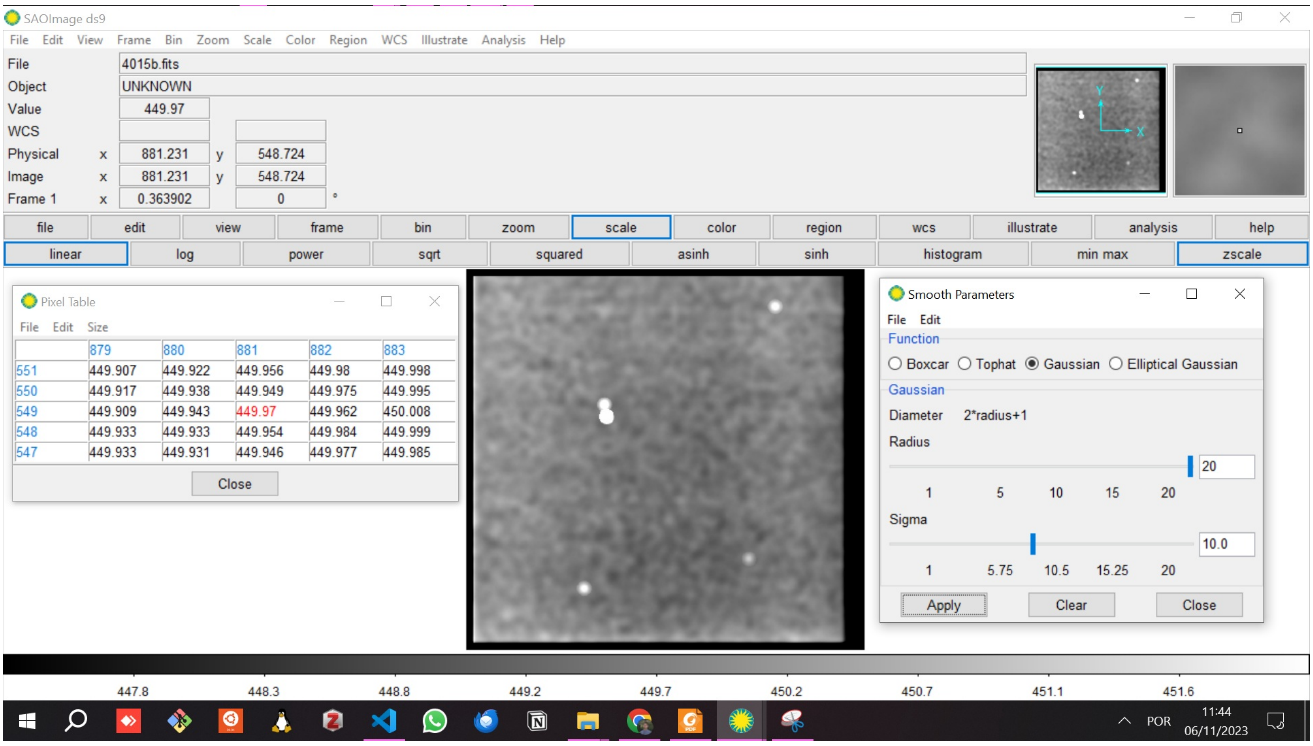
Interactively, roughly estimate by eye the mean level of the data section and of the overscan section (explore the pixel table and smooth tools offered in the “analysis” menu on the top bar).

- What values do you estimate?
- Show a snapshot of your desktop with the DS9 main window and the auxiliary tools panel(s) used in this analysis.

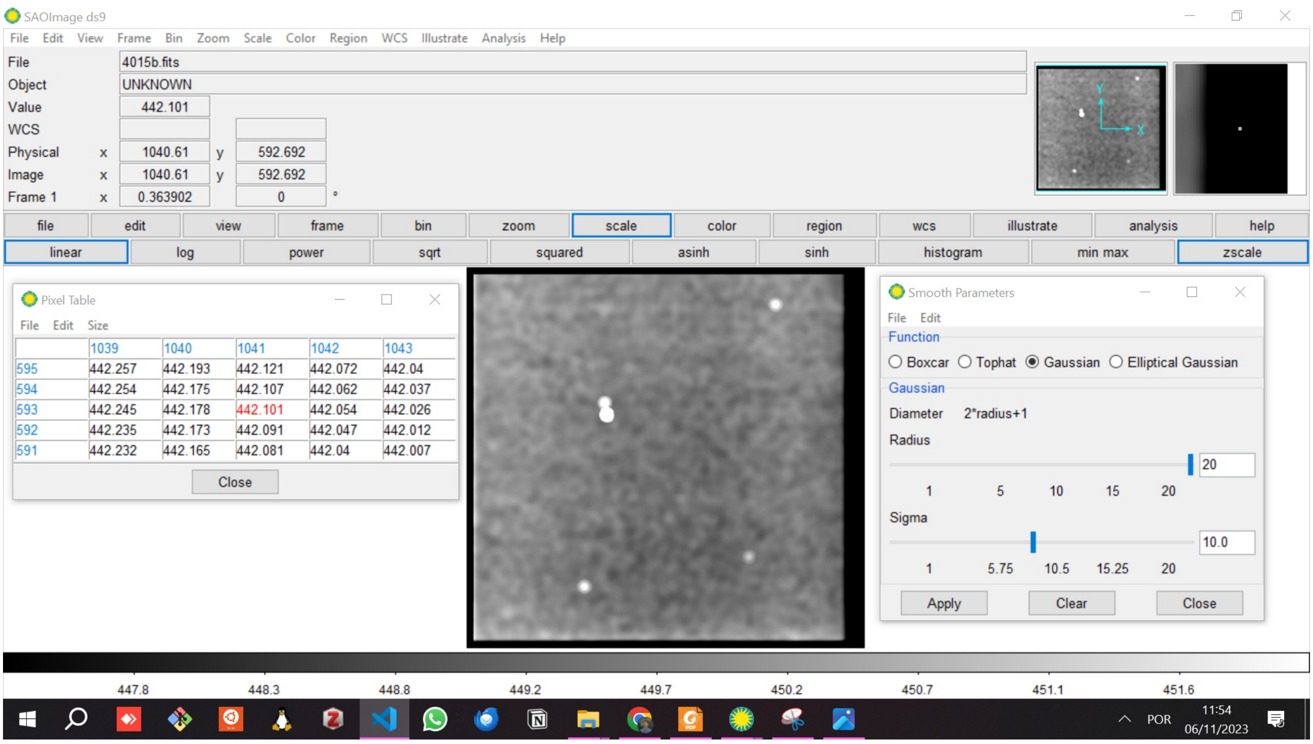
A "pixel table" indica o valor do pixel selecionado, no centro, e pixeis vizinhos. Com a ferramenta "smooth" é possível "alisar" a imagem, aplicando-lhe um género de filtro tipo "salt&pepper", fazendo de imediato os valores convergirem para o seu valor médio no raio do filtro, tornando menores as diferenças aos vizinhos.

Com estas definições determinamos:

Data section mean level ~ 450



Overscan section mean level ~ 442



Tracking the bias level - II

Now, from to the *superbias* folder. Consider appropriate sections (avoid obvious defects) of all images to compute the necessary statistics for:

- Making a scatter plot with x= image number (e.g. 5239 for image 5239o.fits) and y= mean bias level
- Making a scatter plot with x= image number and y= mean bias - mean overscan
- Place both plots side-by-side in a same figure
- What do you conclude?

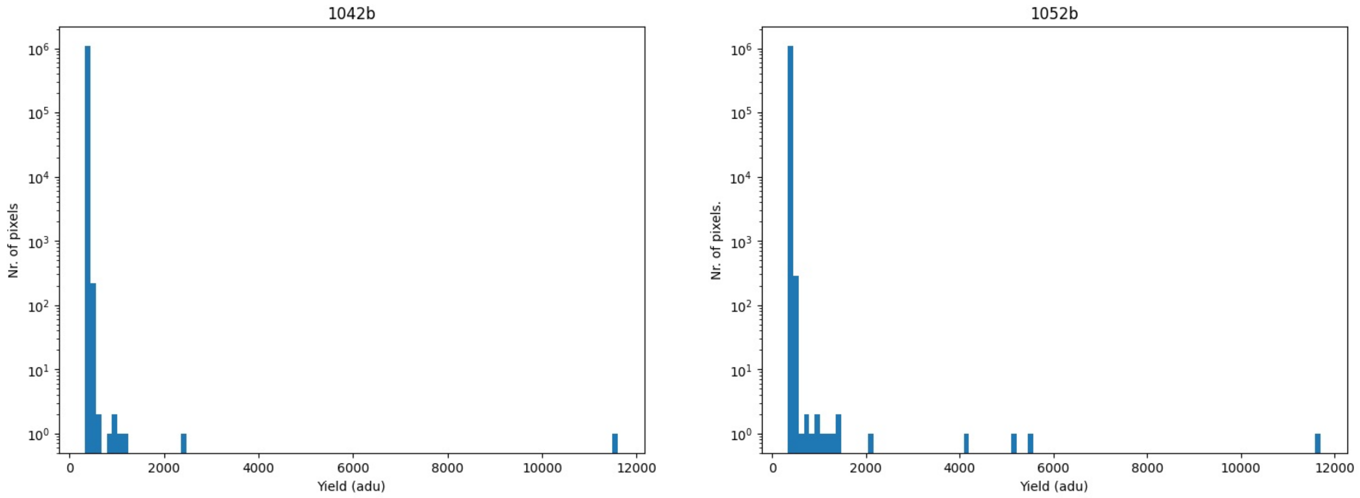
Because the bias images are approximately uniform, the standard deviation of the counts in a small box (e.g. 5x5 or 10x10 pixels) give an idea of the read out noise (here in ADUs, not electrons). From one of your bias images, compute the typical standard deviation in the Data Section and in the Overscan Section. Take 3 boxes spread out each of these sections to estimate the typical values.

- What do you conclude?

In [ ]:
```
## Set-up superbias files' path
superbias_path = Path('superbias')
```

In [ ]:
```
## Lets inspect two of the bias images' pixel level histogram

bias_img1 = CCDData.read(str(superbias_path)+'/1042b.fits', unit='adu').data
bias_img2 = CCDData.read(str(superbias_path)+'/1052b.fits', unit='adu').data


fig = plt.figure(figsize=(18,6))
nbins = 100

ax = fig.add_subplot(1,2,1)
plt.hist(bias_img1.flat, nbins, log=True)
plt.xlabel('Yield (adu)')
plt.ylabel('Nr. of pixels')
plt.title('1042b')

ax = fig.add_subplot(1,2,2)
plt.hist(bias_img2.flat, nbins, log=True)
plt.xlabel('Yield (adu)')
plt.ylabel('Nr. of pixels.')
plt.title('1052b')
```

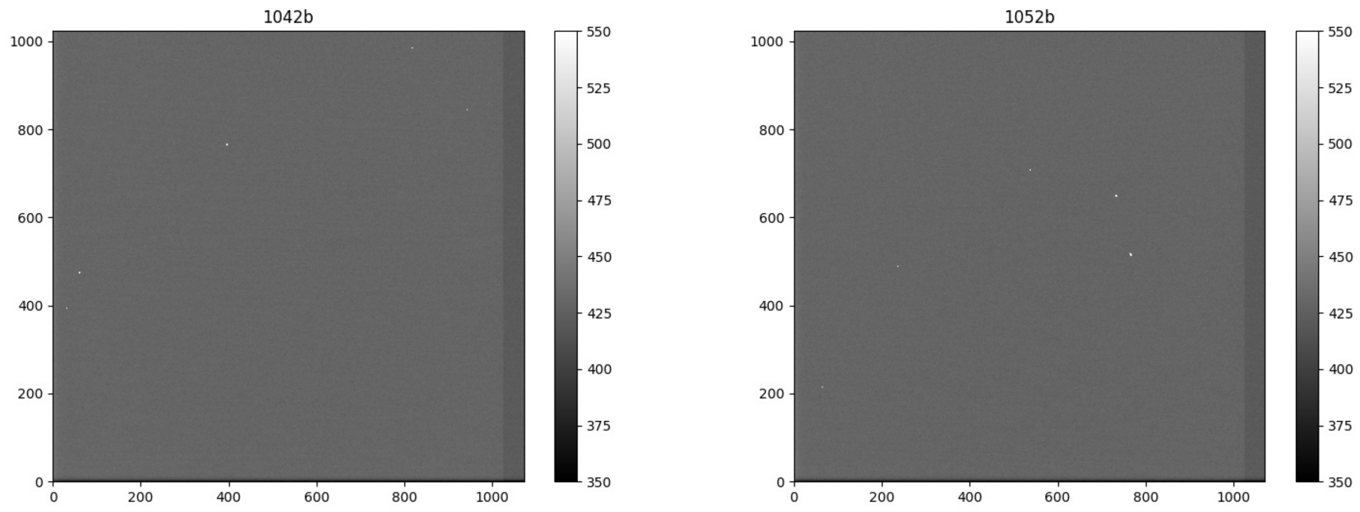
Out[ ]: Text(0.5, 1.0, '1052b')



Verificamos que nas imagens de bias, a grande maioria dos pixeis tem valores de conatgens muito próximos uns dos outros, à exceção algumas regiões onde se vêm valores extremos. Podemos verificar visualmente olhando para estas imagens:

In [ ]:
```
fig = plt.figure(figsize=(18,6))

ax = fig.add_subplot(1,2,1)
plt.imshow(bias_img1, cmap='gray', vmin=350, vmax=550, origin='lower')
plt.title('1042b')
plt.colorbar()

ax = fig.add_subplot(1,2,2)
plt.imshow(bias_img2, cmap='gray', vmin=350, vmax=550, origin='lower')
plt.title('1052b')
plt.colorbar()
```

Out[ ]: <matplotlib.colorbar.Colorbar at 0x7fe42b8e3430>



Os pontos extremos são aqueles onde se veêm os sinais mais brilhantes nas imgens. Nos cálculos seguintes estes pontos devem ser preferencialmente excluidos.

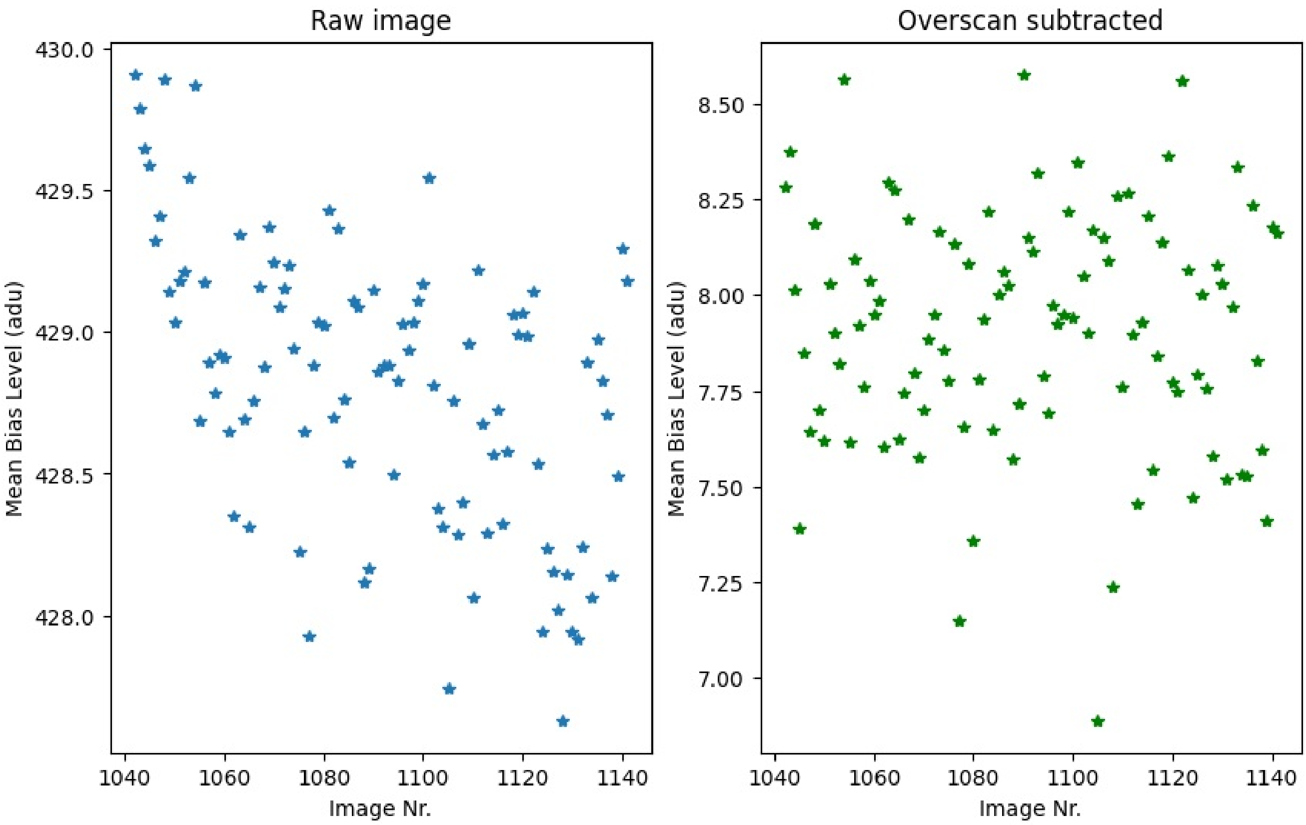
Também as primeiras colunas da imagem (as da esquerda) bem como as primeiras linhas (as de baixo) apresentam valores que se desviam significativamente (visto "a olho") dos valores nos restantes pixeis, pelo que não devem ser consideradas.

Posto isto, e após analisar rapidamente as imagens superbias com recurso ao DS9, para os cálculos seguintes usamos, em cada imagem, a região central de 100x100 pixeis que, para além de evitar a inclusão de pixeis com valores extremos de contagens, acelera consideravelmente o tempo de computação uma vez que reduz a imagem a uma região muito menor.

In [ ]:
```
## Image colection
superbias_files = ImageFileCollection(superbias_path)

## Arrays for image nr and mean bias level values
image_nr = []
mean_bias_level_raw = []
mean_bias_level_over = []

for ccd, file_name in superbias_files.ccds(ccd_kwargs={'unit':'adu'}, return_fname=True):

    ## Subtract overscan ## median=False -> applies average
    ccd_over = ccdp.subtract_overscan(ccd, fits_section='[1030:1068,:]', overscan_axis=1)
    ## Trim subtracted overscan to center 100x100
    ccd_over = ccdp.trim_image(ccd_over, fits_section='[502:522, 502:522]')
    ## Save overscan subtracted mean bias level ()
    mean_bias_level_over.append(float(np.asarray(ccd_over.mean())))

    ## Trim raw image to center 100x100
    ccd = ccdp.trim_image(ccd, fits_section='[502:522, 502:522]')
    ## Save raw image mean bias level to array
    mean_bias_level_raw.append(float(np.asarray(ccd.mean())))

    ## Save image nr to array
    image_nr.append(int(file_name.split('b')[0]))
```

In [ ]:
```
## Scatt plot mean bias level vs image nr
nrow = 1
ncol = 2

fig = plt.figure(figsize=(10,6))

ax = fig.add_subplot(nrow,ncol,1)
plt.plot(image_nr, mean_bias_level_raw, '*')
plt.xlabel('Image Nr.')
plt.ylabel('Mean Bias Level (adu)')
plt.title('Raw image')

ax = fig.add_subplot(nrow,ncol,2)
plt.plot(image_nr, mean_bias_level_over, '*', color='g')
plt.xlabel('Image Nr.')
plt.ylabel('Mean Bias Level (adu)')
plt.title('Overscan subtracted')
```

Out[ ]:
```
Text(0.5, 1.0, 'Overscan subtracted')
```



Analisando os gráficos anteriores concluimos que é extremamente necessário retirar o overscan às imagens de bias antes de calcular o valor médio uma vez que o valor médio do overscan é a componente dominante, e que deve ser constante em todas as imagens, tratando-se duma componente DC no valor dos pixeis [Howell, pág 78].

Cálculo do ruído de leitura numa das imagens, a partir do desvio padrão em pequenas regiões, nas zonas de dados e de overscan:

In [ ]:
```
## Set-up the bias image
bias_img = CCDData.read(str(superbias_path)+'/1042b.fits', unit='adu')

## Create regions
data1 = bias_img.data[50:60,50:60]
data2 = bias_img.data[550:560,550:560]
data3 = bias_img.data[950:960,950:960]
over1 = bias_img.data[50:60,1030:1040]
over2 = bias_img.data[550:560,1040:1050]
over3 = bias_img.data[950:960,1050:1060]

## Compute standard deviations
dev_data1 = np.std(data1)
dev_data2 = np.std(data2)
dev_data3 = np.std(data3)
dev_over1 = np.std(over1)
dev_over2 = np.std(over2)
```

```python
dev_over3 = np.std(over3)

## Compute std differences:
diff1 = dev_data1-dev_over1
diff2 = dev_data2-dev_over2
diff3 = dev_data3-dev_over3

print('Santard deviations in the data section:')
print('Std data 1 = ', f'{dev_data1:.2f}')
print('Std data 2 = ', f'{dev_data2:.2f}')
print('Std data 3 = ', f'{dev_data3:.2f}')
print('Santard deviations in the overscan section:')
print('Std over 1 = ', f'{dev_over1:.2f}')
print('Std over 2 = ', f'{dev_over2:.2f}')
print('Std over 3 = ', f'{dev_over3:.2f}')
print('Standard deviation differentce between data and overscan')
print('Diff image 1 = ', f'{diff1:.2f}')
print('Diff image 2 = ', f'{diff2:.2f}')
print('Diff image 3 = ', f'{diff3:.2f}')
```

```
Santard deviations in the data section:
Std data 1 =  5.07
Std data 2 =  4.89
Std data 3 =  5.33
Santard deviations in the overscan section:
Std over 1 =  4.22
Std over 2 =  3.81
Std over 3 =  4.36
Standard deviation differentce between data and overscan
Diff image 1 =  0.85
Diff image 2 =  1.07
Diff image 3 =  0.96
```

Ao analisar os valores anteriores, do desvio padrão nas diferentes regiões predefinidas para as zonas de dados e overscan, confirmamos aquilo que tihamos visto nos gráficos anteriores, ou seja, a componente de ruído de leitura dominante das imagens bias é a relativa ao ruido de leitura da região de overscan. O ruido de leitura em cada imagem, será então, aproximadamente, a diferença entre a região de dados e a região de oversacan - neste caso, cerca de 1 adu/pixel.

Olhando só para os valores da região de dados, sem subtrair o overscan, verificamos que as estimativas d ruido de leitura com as contribuições de bias e do overscan já perfazem práticamente a totalidade do ruído de leitura calculado na secção anterior (~ 6.33) pelo que estamos completamente mais próximos de reduir otimalmente as fontes de ruído dads imagens dos objetos.

PART 2 - DATA PROCESSING

Overscan and Trim

Select appropriate overscan and trim areas.

- Justify your selection.

Apply the overscan correction and trim all the images. in the *0_mainData* folder. Save the processed images in another folder (e.g. 1_processed).

Verify that the images have been processed:

- Show the header of one of the newly corrected images.
- Present a table showing that all images have been processed.

A zona de overscan da imagem é o conjunto de colunas entre 1025 a 1072. Uma boa região para calcular valores estatísticos na região de overscan (media, mediana, etc.) é uma região mais estreita dentro desta, por exemplo, as colunas entre 1030 a 1068. Com esta seleção estamos a eliminar as 5 colunas de cada um dos lados da região de overscan para evitar a inclusão de outliers nas estatísticas calculadas.

Quanto ao corte da imagem (trim), a região da imagem que efetivamente corresponde à leitura do detetor é toda a região da imagem menos a região de overscan, ou seja as colunas e linhas entre 1 e 1024. À semelhança do que fazemos para a região de overscan usada para a subtração, também no corte da imagem correspondente à leitura do detetor é conveninete evitar a inclusão dos extremos da imagem para excluir possíveis defeitos do contorno. Por exemplo, uma região para aplicar o corte seriam as colunas e linhas entre 4 e 1020 (excluíndo as primeiras 4 linhas e colunas de cada lado da imagem).

In [ ]:
```python
## Set main data write path

mainData_ProcPath = Path('1_processed')
mainData_ProcPath.mkdir(exist_ok=True)
```

In [ ]:
```python
## Loop over the images
```

```python
for ccd, file_name in mainData_files.ccds(return_fname=True):

    ## Subtract the overscan median value
    ccd = ccdp.subtract_overscan(ccd, fits_section='[1030:1068,:]', median=True, overscan_axis=1)

    ## Trim the overscan and edges of the image
    ccd = ccdp.trim_image(ccd, fits_section='[4:1020, 4:1020]')

    ## Save the result
    ccd.write(str(mainData_ProcPath)+'/OT_'+file_name, overwrite=True)
```

In [ ]:

```python
## Verify that the images have been processed

## Header of a processed image
fits.open(str(mainData_ProcPath)+'/'+'OT_4015b.fits')[0].header
```

Out[ ]:

```
SIMPLE  =                    T / conforms to FITS standard
BITPIX  =                  -64 / array data type
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                 1017
NAXIS2  =                 1017
ORIGIN  = 'Copyright (C) 1991-1998 GKR Computer Consulting' / FITS file originat
DATE    = '2001-06-05T23:49:42' / Date FITS file was generated
IRAF-TLM= '15:49:42 (05/06/2001)' / Time of last modification
EXPTIME =                    1
FILTER  = 'UNKNOWN '
IMAGETYP= 'BIAS    '
CCDSUM  = '1 1     '
CCDSEC  = '[1:1024,1:1024]'
DATASEC = '[1:1024,1:1024]'
BIASSEC = '[1025:1072,1:1024]'
TRIMSEC = '[1:1024,1:1024]'
OBJECT  = 'UNKNOWN '
RA      = 'UNKNOWN '
DEC     = 'UNKNOWN '
EPOCH   =               0000.0
UT      = '23:58:12'
DATE-OBS= '2001-02-21'
OBSERVAT= 'SPM     '
TELESCOP= '0.84    '
LATITUDE= '+31:02:39'
LONGITUD= '-115:27:49'
ALTITUDE=                 2800
OBSERVER= 'Moitinho'
INSTRUME= 'la cubeta'
DETECTOR= 'SITe1 1k'
GAINMODE=                    4
GAIN    =                 1.28
RDNOISE =                 6.33
BUNIT   = 'adu     '
HIERARCH SUBTRACT_OVERSCAN = 'suboscan' / Shortened name for ccdproc command
SUBOSCAN= 'ccd=<CCDData>, fits_section=[1030:1068,:], median=True, &'
CONTINUE  'overscan_axis=1'
HIERARCH TRIM_IMAGE = 'trimim  ' / Shortened name for ccdproc command
TRIMIM  = 'ccd=<CCDData>, fits_section=[4:1020, 4:1020]'
HISTORY  PMIS macros of 2001-01-23
HISTORY  Written by Stephen Levine
HISTORY  Modified by Gaguik Tovmassian
HISTORY  Modified by Alan Watson & Michael Richer
```

In [ ]:

```python
## Table with processed images' info
mainData_ProcFiles = ImageFileCollection(mainData_ProcPath)
mainData_ProcFiles.summary['file', 'suboscan', 'trimim']
```

Out[ ]: *Table length=19*

file	suboscan	trimim
str13	str71	str44
OT_3241o.fits	ccd=<CCDData>, fits_section=[1030:1068,:], median=True, overscan_axis=1	ccd=<CCDData>, fits_section=[4:1020, 4:1020]
OT_3249o.fits	ccd=<CCDData>, fits_section=[1030:1068,:], median=True, overscan_axis=1	ccd=<CCDData>, fits_section=[4:1020, 4:1020]
OT_3262o.fits	ccd=<CCDData>, fits_section=[1030:1068,:], median=True, overscan_axis=1	ccd=<CCDData>, fits_section=[4:1020, 4:1020]
OT_3265o.fits	ccd=<CCDData>, fits_section=[1030:1068,:], median=True, overscan_axis=1	ccd=<CCDData>, fits_section=[4:1020, 4:1020]
OT_4015b.fits	ccd=<CCDData>, fits_section=[1030:1068,:], median=True, overscan_axis=1	ccd=<CCDData>, fits_section=[4:1020, 4:1020]
OT_4016b.fits	ccd=<CCDData>, fits_section=[1030:1068,:], median=True, overscan_axis=1	ccd=<CCDData>, fits_section=[4:1020, 4:1020]
OT_4017b.fits	ccd=<CCDData>, fits_section=[1030:1068,:], median=True, overscan_axis=1	ccd=<CCDData>, fits_section=[4:1020, 4:1020]
OT_4075f.fits	ccd=<CCDData>, fits_section=[1030:1068,:], median=True, overscan_axis=1	ccd=<CCDData>, fits_section=[4:1020, 4:1020]
OT_4076f.fits	ccd=<CCDData>, fits_section=[1030:1068,:], median=True, overscan_axis=1	ccd=<CCDData>, fits_section=[4:1020, 4:1020]
OT_4078f.fits	ccd=<CCDData>, fits_section=[1030:1068,:], median=True, overscan_axis=1	ccd=<CCDData>, fits_section=[4:1020, 4:1020]
OT_4079f.fits	ccd=<CCDData>, fits_section=[1030:1068,:], median=True, overscan_axis=1	ccd=<CCDData>, fits_section=[4:1020, 4:1020]
OT_4080f.fits	ccd=<CCDData>, fits_section=[1030:1068,:], median=True, overscan_axis=1	ccd=<CCDData>, fits_section=[4:1020, 4:1020]
OT_4081f.fits	ccd=<CCDData>, fits_section=[1030:1068,:], median=True, overscan_axis=1	ccd=<CCDData>, fits_section=[4:1020, 4:1020]
OT_4082f.fits	ccd=<CCDData>, fits_section=[1030:1068,:], median=True, overscan_axis=1	ccd=<CCDData>, fits_section=[4:1020, 4:1020]
OT_4083f.fits	ccd=<CCDData>, fits_section=[1030:1068,:], median=True, overscan_axis=1	ccd=<CCDData>, fits_section=[4:1020, 4:1020]
OT_4084f.fits	ccd=<CCDData>, fits_section=[1030:1068,:], median=True, overscan_axis=1	ccd=<CCDData>, fits_section=[4:1020, 4:1020]
OT_4085f.fits	ccd=<CCDData>, fits_section=[1030:1068,:], median=True, overscan_axis=1	ccd=<CCDData>, fits_section=[4:1020, 4:1020]
OT_4086f.fits	ccd=<CCDData>, fits_section=[1030:1068,:], median=True, overscan_axis=1	ccd=<CCDData>, fits_section=[4:1020, 4:1020]
OT_4087f.fits	ccd=<CCDData>, fits_section=[1030:1068,:], median=True, overscan_axis=1	ccd=<CCDData>, fits_section=[4:1020, 4:1020]

Creating the Bias correction image

From the *superbias* folder, apply the overscan correction and trim all the 100 images.

Combine them to create the master bias correction image (we'll refer to it as the superZero image).

- Explain your choices for the image combination (combine method, parameters, etc)

Estimate the typical mean level and standard deviation (noise) of the superZero image.

- How does it compare with the values from any of the individual images used to produce it? Why?
- Display a picture of superZero.

In [ ]:
```
## Set-up path to write processed superbias files
superbias_ProcPath = Path('ProcSuperbias')
```

In [ ]:
```
## Loop over the bias images
for ccd, file_name in superbias_files.ccds(ccd_kwargs={'unit': 'adu'}, return_fname=True):

    ## Subtract the overscan median value
    ccd = ccdp.subtract_overscan(ccd, fits_section='[1030:1068,:]', median=True, overscan_axis=1)

    ## Trim the overscan and edges of the image
    ccd = ccdp.trim_image(ccd, fits_section='[4:1020, 4:1020]')

    ## Save the result
    ccd.write(str(superbias_ProcPath)+'/OT_'+file_name, overwrite=True)
```

In [ ]:
```
## Set-up processed superbias image file colection
superbias_ProcFiles = ImageFileCollection(superbias_ProcPath, glob_include='OT_*')
list_bias = superbias_ProcFiles.files_filtered(imagetyp='zero',include_path=True)
```

In [ ]:
```
## Combine superbias images
superZero = ccdp.combine(list_bias,
                         method='average',
                         sigma_clip=True,
                         sigma_clip_low_thresh=5,
                         sigma_clip_high_thresh=5,
                         sigma_clip_func=np.ma.median,
                         sigma_clip_dev_func=mad_std,
                         mem_limit=350e6)
superZero.meta['combined'] = True ## update info in the header
```

```
superZero.write(str(superbias_ProcPath)+'/superZero.fits', overwrite=True)
```

```
INFO:astropy:splitting each image into 14 chunks to limit memory usage to 350000000.0 bytes.
INFO: splitting each image into 14 chunks to limit memory usage to 350000000.0 bytes. [ccdproc.combiner]
```

Bias correction

In the processed folder, use the superZero image to apply the bias correction to the images that should be corrected. Make sure a keyword to was added to flag that the images have been bias subtracted.

Verify that the images have been processed:

- Show the header of one of the newly corrected images.
- Present a table showing that all images have been processed.

In [ ]:
```
## set-up the processed bias images
processed_files = ImageFileCollection(mainData_ProcPath, glob_include='OT_*')

## Select object and flat images
list_OTZ = processed_files.files_filtered(regex_match=True, imagetyp='object|flat', include_path=True)

ic_OTZ = processed_files.filter(regex_match=True, imagetyp='object|flat')
```

In [ ]:
```
## Read the superBias image
superBias = CCDData.read(str(superbias_ProcPath)+'/superZero.fits')
```

In [ ]:
```
## Subtract the superbias from the objects and flats main data

## Loop over object and flat images
for ccd, file_name in ic_OTZ.ccds(return_fname=True):

    ccd = ccdp.subtract_bias(ccd, superBias) ## Subtract super bias
    ccd.header['zerocor'] = 'superZero.fits' ## Update header info
    ccd.write(str(mainData_ProcPath)+'/Z'+file_name, overwrite=True)
```

Creating the flatfield correction images

Create the flatfield correction images for both filters. Use the task default parameters. What problem did you find? Fix it!

- How did you fix the problem? (NOTE: take care not to use the combined flat images you've created in the previous attempt when generating the new good ones)
- Show the good and the bad flatfield correction images you have created. Identify the images clearly with labels.

In [ ]:
```
## Image file colection of the processed files
ZOT_files = ImageFileCollection(mainData_ProcPath, glob_include='ZOT_*')
```

In [ ]:
```
## List existing filters
flat_filters = np.unique(ZOT_files.summary['filter'])
flat_filters
```

Out[ ]: <MaskedColumn name='filter' dtype='str1' length=2>

I
V

In [ ]:
```
## Set-up list of flat images for each filter
ZOT_files_flatI = ZOT_files.files_filtered(filter='I', imagetyp='FLAT',include_path=True)
ZOT_files_flatV = ZOT_files.files_filtered(filter='V', imagetyp='FLAT',include_path=True)

ZOT_flats_list = ZOT_files.files_filtered(imagetyp='FLAT')
```

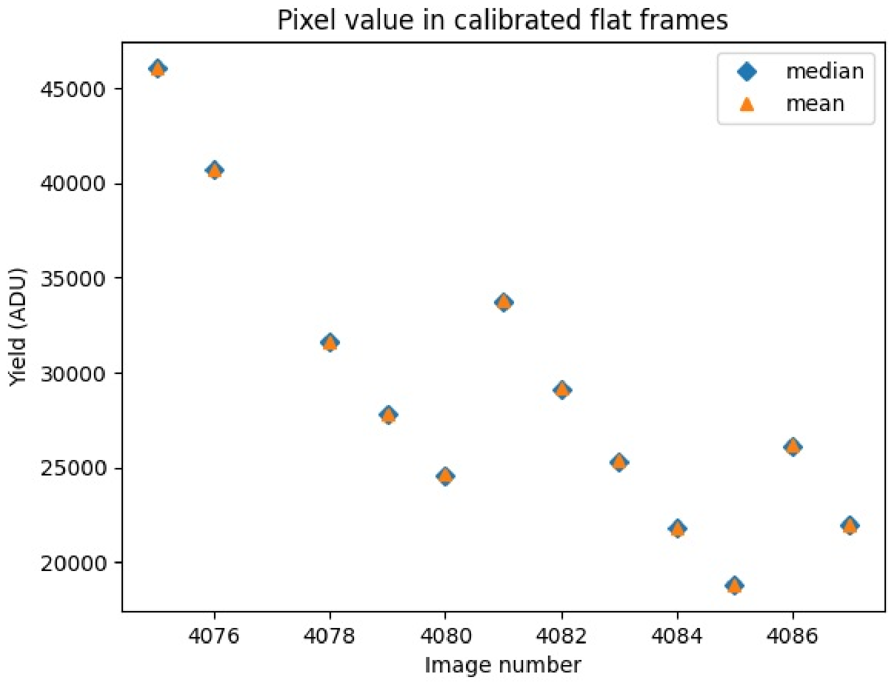
Normalize images before combining

In [ ]:
```
## Image section avoiding edges
imsection = np.index_exp[5:1020,5:1020]

## Compute median and mean over the section area of the flats
median_yield = [np.median(data[imsection]) for data in ZOT_files.data(imagetyp='FLAT')]
mean_yield = [np.mean(data[imsection]) for data in ZOT_files.data(imagetyp='FLAT')]
```

In [ ]:
```
## Get flat images' numbers
img_nr = []
for file in ZOT_flats_list:
    img_nr.append(int(file.replace('ZOT_','').split('f')[0]))
```

```
## Plot
plt.plot(img_nr, median_yield,'D', label='median')
plt.plot(img_nr, mean_yield,'^', label='mean')
plt.xlabel('Image number')
plt.ylabel('Yield (ADU)')
plt.title('Pixel value in calibrated flat frames')
plt.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x7fe4358f6f40>



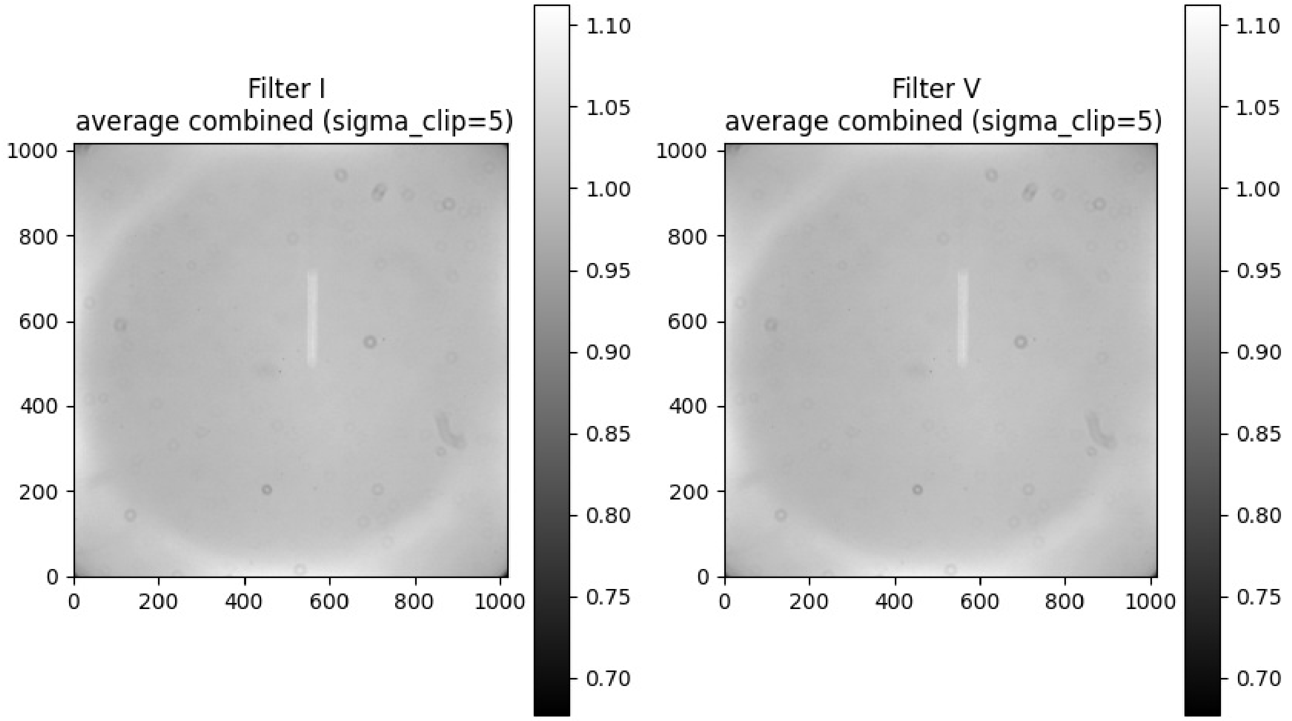
No gráfico anterior verificamios que, em cada imagem flat, os valores médio e mediano de contagens nos pixeis são iguais. Sendo assim, é indiferente a grandeza que se usa para normalizar as imagens. Definimos de seguida a função "inverso da mediana" para normalizar as imagens durante a combinação.

In [ ]:
```
## Inverse of the median
def inv_median(a):
    return 1/np.median(a)
```

In [ ]:
```
## Combine flats using the default parameters: method=average and simetric sigma_clip=5
for filt in flat_filters:
    to_combine = ZOT_files.files_filtered(imagetyp='FLAT', filter=filt, include_path=True)
    combined_flat = ccdp.combine(to_combine,
                                 method='average',
                                 scale=inv_median,
                                 sigma_clip=True,
                                 sigma_clip_low_thresh=5,
                                 sigma_clip_high_thresh=5,
                                 sigma_clip_func=np.ma.median,
                                 signma_clip_dev_func=mad_std,
                                 mem_limit=350e6
                                )
    ## Parameters definition: https://ccdproc.readthedocs.io/en/latest/api/ccdproc.Combiner.html#ccdproc.Combin
    combined_flat.meta['combined'] = True
    flat_name = 'Average_SC5_Flat_{}.fits'.format(filt.replace("'", "p"))
    combined_flat.write(str(mainData_ProcPath)+'/'+flat_name, overwrite=True)
```

In [ ]:
```
## Plot the combined flats via the average method, using simetric sigma_clip=5

fig = plt.figure(figsize=(10,6))

ax = fig.add_subplot(nrow,ncol,1)
plt.imshow(fits.open(str(mainData_ProcPath)+'/Average_SC5_Flat_I.fits')[0].data, cmap='gray', origin='lower')
plt.colorbar()
plt.title('Filter I \n average combined (sigma_clip=5)')

ax = fig.add_subplot(nrow,ncol,2)
plt.imshow(fits.open(str(mainData_ProcPath)+'/Average_SC5_Flat_I.fits')[0].data, cmap='gray', origin='lower')
plt.colorbar()
plt.title('Filter V \n average combined (sigma_clip=5)')
```

Out[ ]: Text(0.5, 1.0, 'Filter V \n average combined (sigma_clip=5)')



Combinando as imagens flat com o método da média, e sigma_clip_low = sigma_clip_high = 5, verifica-se um defeito na imagem combinada - uma faixa brilhante, vertical, no centro da imagem, aparece.

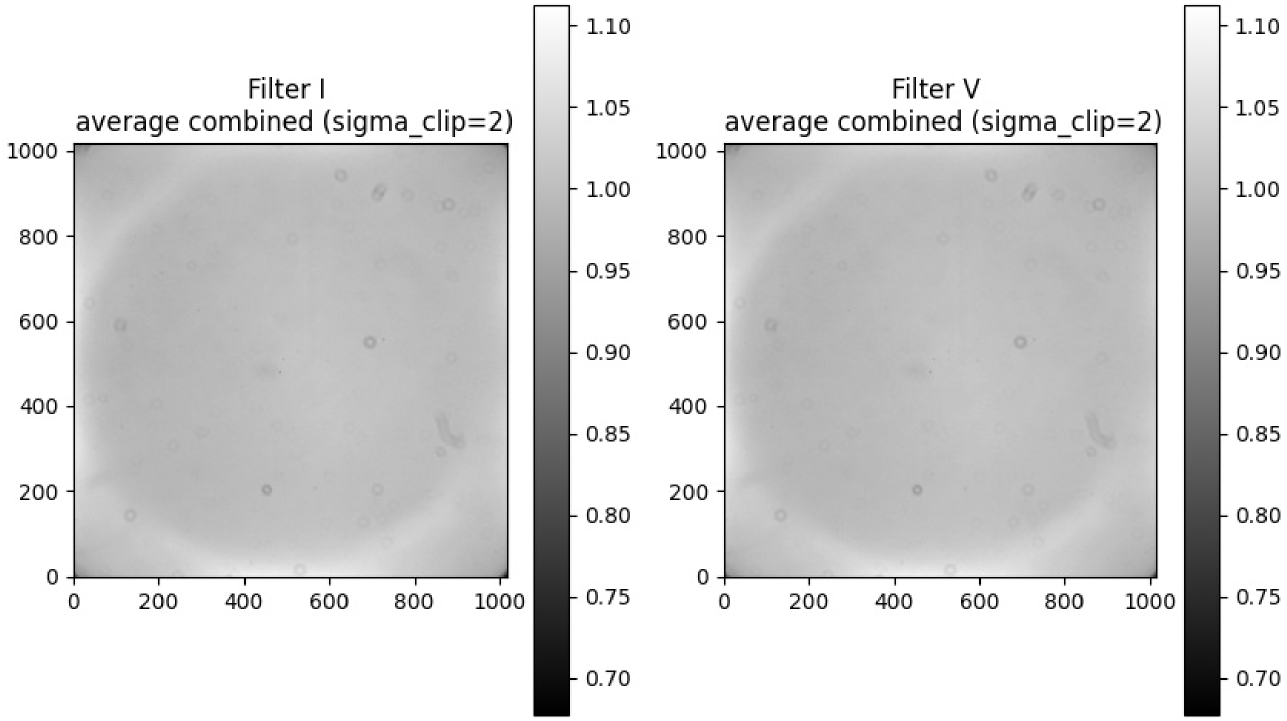
Este efeito pode ser devido, por exemplo, à presença de um evento, como uma estrela brilhante, durante a captura das imagens flat. Uma forma de contornar este constrangimento é comum combinar as imagens com o método da mediana [Howel, pág 68], em vez da média, uma vez que a mediana é muito menos sensível a valores muito afastados do valor mediano.

No entanto, vamos ainda analisar uma segunda alternativa, explorando as funcionalidades da função "combine". Em cada imagem, é calculada a mediana (sigma_clip_func) dos valores, o desvio padrão mediano (signma_clip_dev_func) e são tidos em conta os pixeis com valores entre sigma_clip_low e sigma_clip_high.

In [ ]:
```
## Combine flats using the parameters: method=average and simetric sigma_clip=2
for filt in flat_filters:
    to_combine = ZOT_files.files_filtered(imagetyp='FLAT', filter=filt, include_path=True)
    combined_flat = ccdp.combine(to_combine,
                                 method='average',
                                 scale=inv_median,
                                 sigma_clip=True,
                                 sigma_clip_low_thresh=2,
                                 sigma_clip_high_thresh=2,
                                 sigma_clip_func=np.ma.median,
                                 signma_clip_dev_func=mad_std,
                                 mem_limit=350e6
                                )
    combined_flat.meta['combined'] = True
    flat_name = 'Average_SC2_Flat_{}.fits'.format(filt.replace("''", "p"))
    combined_flat.write(str(mainData_ProcPath)+'/'+flat_name, overwrite=True)
```

In [ ]:
```
## Plot the combined flats via the average method, using simetric sigma_clip=2

fig = plt.figure(figsize=(10,6))

ax = fig.add_subplot(nrow,ncol,1)
plt.imshow(fits.open(str(mainData_ProcPath)+'/Average_SC2_Flat_I.fits')[0].data, cmap='gray', origin='lower')
plt.colorbar()
plt.title('Filter I \n average combined (sigma_clip=2)')

ax = fig.add_subplot(nrow,ncol,2)
plt.imshow(fits.open(str(mainData_ProcPath)+'/Average_SC2_Flat_I.fits')[0].data, cmap='gray', origin='lower')
plt.colorbar()
plt.title('Filter V \n average combined (sigma_clip=2)')
```

Out[ ]:
```
Text(0.5, 1.0, 'Filter V \n average combined (sigma_clip=2)')
```



Usando sigma_clip = 2 (ainda com o método da média para a combinação das imagem) o defeito praticamente desaparece, mas podemos estar a perder demasiada estatistica de pixeis que não estão a ser contabilizados. Neste caso, tratando-se de um defeito de elevada luminosidade até poderiamos apenas reduzir o sigma_clip superior.

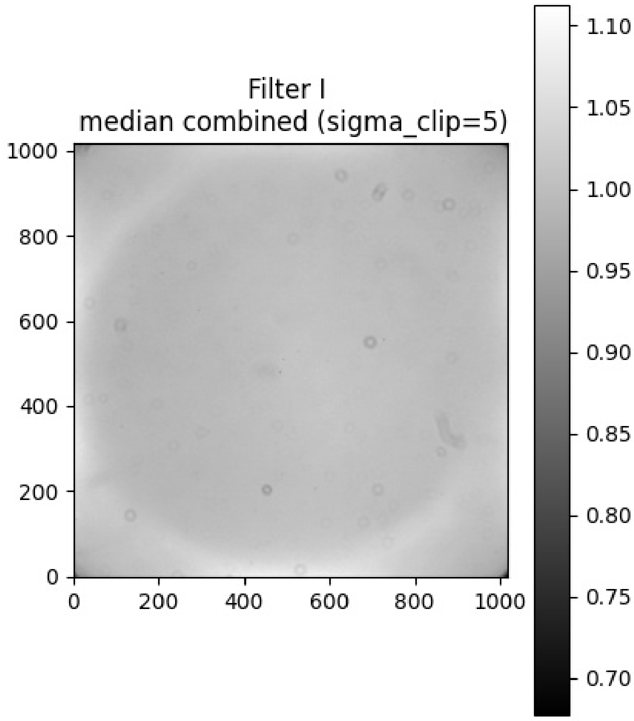
Usando agora a combinação através do método da mediana, e tomando os valores dos pixeis, em cada imagem, entre valor mediano +- 5 sigma:

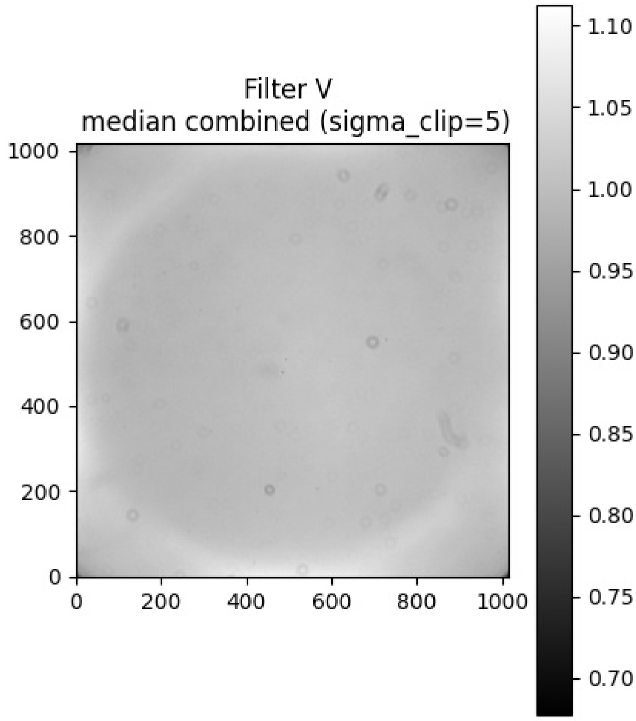
In [ ]:
```
## Combine flats using the parameters: method=median and simetric sigma_clip=5
for filt in flat_filters:
    to_combine = ZOT_files.files_filtered(imagetyp='FLAT', filter=filt, include_path=True)
    combined_flat = ccdp.combine(to_combine,
                                 method='median',
                                 scale=inv_median,
                                 sigma_clip=True,
                                 sigma_clip_low_thresh=5,
                                 sigma_clip_high_thresh=5,
                                 sigma_clip_func=np.ma.median,
                                 signma_clip_dev_func=mad_std,
                                 mem_limit=350e6
                                )
    combined_flat.meta['combined'] = True
    flat_name = 'Flat_{}.fits'.format(filt.replace("''", "p"))
    combined_flat.write(str(mainData_ProcPath)+'/'+flat_name, overwrite=True)
```

```
INFO:astropy:splitting each image into 2 chunks to limit memory usage to 350000000.0 bytes.
INFO: splitting each image into 2 chunks to limit memory usage to 350000000.0 bytes. [ccdproc.combiner]
```

In [ ]:
```
## Plot the combined flats via the median method, using simetric sigma_clip=5

fig = plt.figure(figsize=(10,6))

ax = fig.add_subplot(nrow,ncol,1)
plt.imshow(fits.open(str(mainData_ProcPath)+'/Flat_I.fits')[0].data, cmap='gray', origin='lower')
plt.colorbar()
plt.title('Filter I \n median combined (sigma_clip=5)')

ax = fig.add_subplot(nrow,ncol,2)
plt.imshow(fits.open(str(mainData_ProcPath)+'/Flat_I.fits')[0].data, cmap='gray', origin='lower')
plt.colorbar()
plt.title('Filter V \n median combined (sigma_clip=5)')
```

Out[ ]:
```
Text(0.5, 1.0, 'Filter V \n median combined (sigma_clip=5)')
```





Usando a combinação com a mediana, outliars nas imagens flat originais não têm um impacto significativo no valor das imagens combinadas, não sendo necessário reduzir a região de interesse com um sigma_clip menor.

Flatfield correction

Apply the flatfield correction to the object images. Verify that the images have been processed:

- Show the header of one of the newly corrected images.
- Present a table showing that all images have been processed.

In [ ]:
```
## Loop over the different filters
for filt in flat_filters:

    ## Select the appropriate combined flat image
    flat = CCDData.read(str(mainData_ProcPath)+'/Flat_'+str(filt)+'.fits')

    # Select the object images with that filter
    objects = ZOT_files.filter(imagetyp='object', filter=filt)
    for ccd, file_name in objects.ccds(return_fname=True):
        ccd = ccdp.flat_correct(ccd, flat)
        ccd.header['flatcor'] = 'Flat_'+str(filt)
        # Save the result
        ccd.write(str(mainData_ProcPath)+'/F'+file_name, overwrite=True)
```

In [ ]:
```
## Header of one of the newly corrected images
ccd.header
```

Out[ ]:
```
SIMPLE  =                    T / conforms to FITS standard
BITPIX  =                  -64 / array data type
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                 1017
NAXIS2  =                 1017
EXTEND  =                    T
ORIGIN  = 'Copyright (C) 1991-1998 GKR Computer Consulting' / FITS file originat
DATE    = '2001-06-06T00:19:27' / Date FITS file was generated
IRAF-TLM= '15:49:41 (05/06/2001)' / Time of last modification
EXPTIME =                  180
FILTER  = 'V       '
IMAGETYP= 'OBJECT  '
CCDSUM  = '1 1     '
CCDSEC  = '[1:1024,1:1024]'
DATASEC = '[1:1024,1:1024]'
BIASSEC = '[1025:1072,1:1024]'
TRIMSEC = '[1:1024,1:1024]'
OBJECT  = 'NGC2420 '
RA      = '07:38:27.6'
DEC     = '+21:34:00'
UT      = '08:45:32'
DATE-OBS= '2001-02-21'
TELESCOP= '0.84    '
LATITUDE= '+31:02:39'
LONGITUD= '-115:27:49'
ALTITUDE=                 2800
OBSERVER= 'Moitinho'
INSTRUME= 'La Cubeta'
DETECTOR= 'SITe1 1k'
GAINMODE=                    4
HISTORY  PMIS macros of 2001-01-23
HISTORY  Written by Stephen Levine
HISTORY  Modified by Gaguik Tovmassian
HISTORY  Modified by Alan Watson & Michael Richer
OBSERVAT= 'spm     '
EPOCH   =     2001.14131404163
ST      = '11:08:57.32'
AIRMASS =      1.4933997403543
GAIN    =                 1.28
RDNOISE =                 6.33
BUNIT   = 'adu     '
HIERARCH SUBTRACT_OVERSCAN = 'suboscan' / Shortened name for ccdproc command
SUBOSCAN= 'ccd=<CCDData>, fits_section=[1030:1068,:], median=True, &'
CONTINUE  'overscan_axis=1'
HIERARCH TRIM_IMAGE = 'trimim  ' / Shortened name for ccdproc command
TRIMIM  = 'ccd=<CCDData>, fits_section=[4:1020, 4:1020]'
HIERARCH SUBTRACT_BIAS = 'subbias ' / Shortened name for ccdproc command
SUBBIAS = 'ccd=<CCDData>, master=<CCDData>'
ZEROCOR = 'superZero.fits'
HIERARCH FLAT_CORRECT = 'flatcor ' / Shortened name for ccdproc command
FLATCOR = 'Flat_V  '
```

In [ ]:
```python
## Table showing that all object images have been processed
## overcsan subtracted, trimmed, bias corrected and flat corrected

processedObjectImages = ImageFileCollection(mainData_ProcPath, glob_include='FZOT_*')
processedObjectImages.summary['file', 'suboscan', 'trimim', 'zerocor', 'flatcor']
```

Out[ ]: *Table length=4*

file	suboscan	trimim	zerocor	flatcor
str15	str71	str44	str14	str6
FZOT_3241o.fits	ccd=<CCDData>, fits_section=[1030:1068,:], median=True, overscan_axis=1	ccd=<CCDData>, fits_section=[4:1020, 4:1020]	superZero.fits	Flat_I
FZOT_3249o.fits	ccd=<CCDData>, fits_section=[1030:1068,:], median=True, overscan_axis=1	ccd=<CCDData>, fits_section=[4:1020, 4:1020]	superZero.fits	Flat_I
FZOT_3262o.fits	ccd=<CCDData>, fits_section=[1030:1068,:], median=True, overscan_axis=1	ccd=<CCDData>, fits_section=[4:1020, 4:1020]	superZero.fits	Flat_V
FZOT_3265o.fits	ccd=<CCDData>, fits_section=[1030:1068,:], median=True, overscan_axis=1	ccd=<CCDData>, fits_section=[4:1020, 4:1020]	superZero.fits	Flat_V

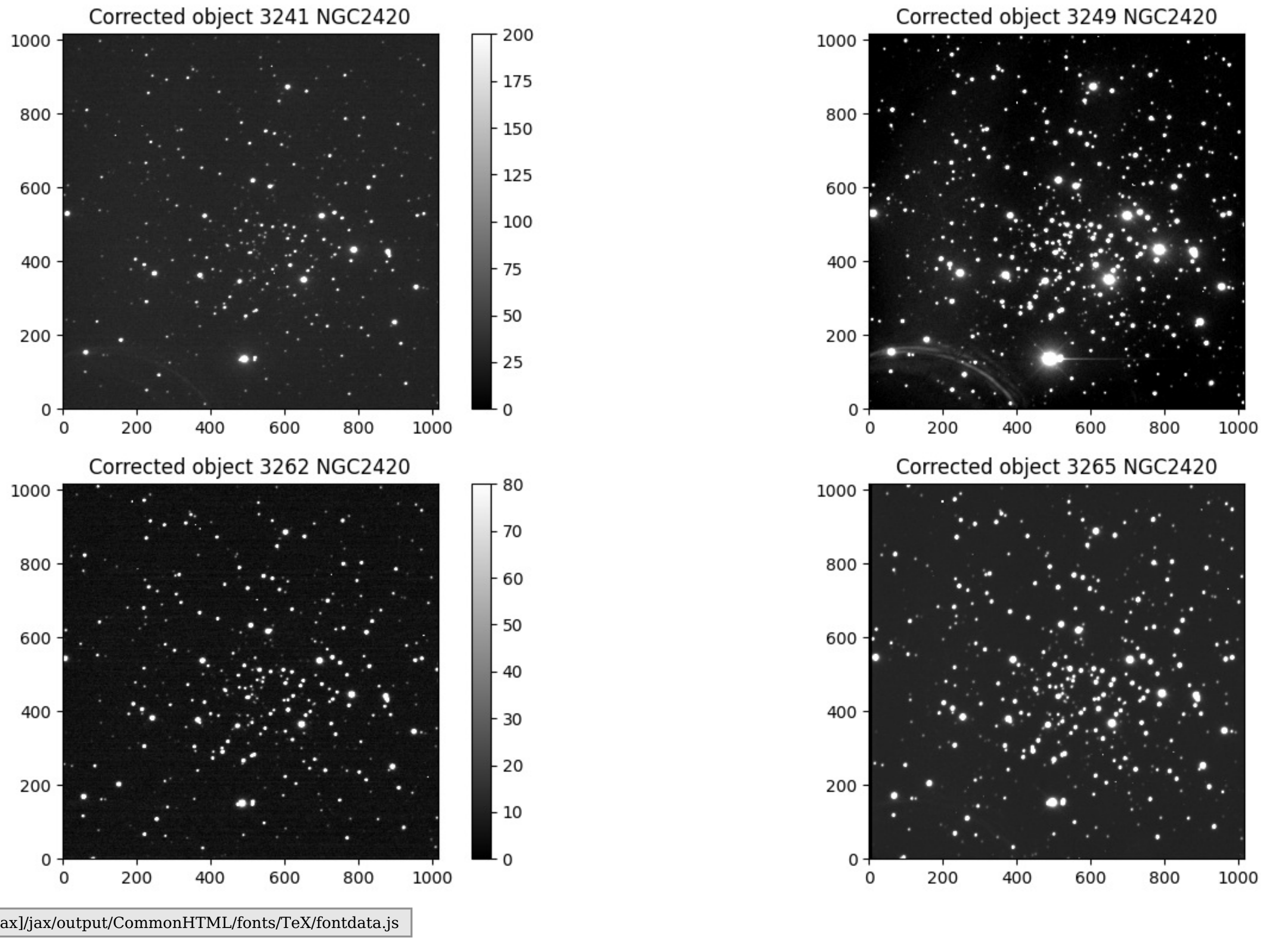
You did it! The final science images are ready for delivery to the client. You have processed and removed the detector footprint down to the percent level.

- Display the 4 final corrected images. Identify each images clearly with labels

In [ ]:
```python
## Display the 4 final corrected images
```

```
ncols = 2
nrows = 2

fig = plt.figure(figsize=(16,9))

ax = fig.add_subplot(nrows,ncols,1)
plt.imshow(fits.open(str(mainData_ProcPath)+'/FZOT_3241o.fits')[0].data,
           cmap='gray', origin='lower', vmin=0, vmax=200)
plt.colorbar()
plt.title('Corrected object 3241 NGC2420')

ax = fig.add_subplot(nrows,ncols,2)
plt.imshow(fits.open(str(mainData_ProcPath)+'/FZOT_3249o.fits')[0].data,
           cmap='gray', origin='lower', vmin=550, vmax=1000)
plt.colorbar()
plt.title('Corrected object 3249 NGC2420')

ax = fig.add_subplot(nrows,ncols,3)
plt.imshow(fits.open(str(mainData_ProcPath)+'/FZOT_3262o.fits')[0].data,
           cmap='gray', origin='lower', vmin=0, vmax=80)
plt.colorbar()
plt.title('Corrected object 3262 NGC2420')

ax = fig.add_subplot(nrows,ncols,4)
plt.imshow(fits.open(str(mainData_ProcPath)+'/FZOT_3265o.fits')[0].data,
           cmap='gray', origin='lower', vmin=10, vmax=800)
plt.colorbar()
plt.title('Corrected object 3265 NGC2420')
```

Out[ ]: Text(0.5, 1.0, 'Corrected object 3265 NGC2420')



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js