301_HW3

May 11, 2020

Step 0: Apply bias and overscan

[17]:

```
# libraries

#for ccd proc to convert our array data to ccd data
from astropy.visualization import ZScaleInterval
from astropy.nddata import CCDData
import ccdproc as ccdp
from ccdproc import subtract_overscan
import matplotlib.pyplot as plt

#import data from UH 88 Tek Raw Data (2 options currently)
# path to directory containing image data
base = ("/Users/alix/Bunny/UH88 Tek RAW data/")
full_coll = ccdp.ImageFileCollection(
    base,
    keywords=["OBJECT", "FILTER", "EXPTIME", "GAIN"],
    glob_exclude="*tek01[1,2].fits",
)
#full_coll.summary
```

[9]:

```
# function for overscan
def overscan_Data(fname):
    # load data
    frame = CCDData.read(fname, unit="adu")

    # Subtract the overscan
    flat_reduced = ccdp.subtract_overscan(frame, overscan=frame[:, 2055:],␣
 ↪median=True)

    # Trim the overscan
    flat_reduced = ccdp.trim_image(flat_reduced[:, :2048])

    return flat_reduced
# function for bias (above?) is overscan the accumulation of bias and sky flat?
```

[18]:

```
#or make a new directory with all the data but now messed with for the purposes
```

```
#of the assignment still using "Path" where "example1..." would be UH 88 Tek␣
 ↪Raw Data
   # = Path('.', 'example5-reduced')
#calibrated_data.mkdir(exist_ok=True)
base = ("/Users/alix/Bunny/UH88 Tek RAW data/")
full_coll = ccdp.ImageFileCollection(
    base,
    keywords=["OBJECT", "FILTER", "EXPTIME", "GAIN"],
    glob_exclude="*tek01[1,2].fits",
)
#apply functions
# 'data' is nothing
calibrated_data = []
for fname in full_coll.files_filtered():
    # call function
    reduced = overscan_Data(base + fname)
    calibrated_data.append(reduced)
    #reduced.write(calibrated_data / fname)
```

[19]:

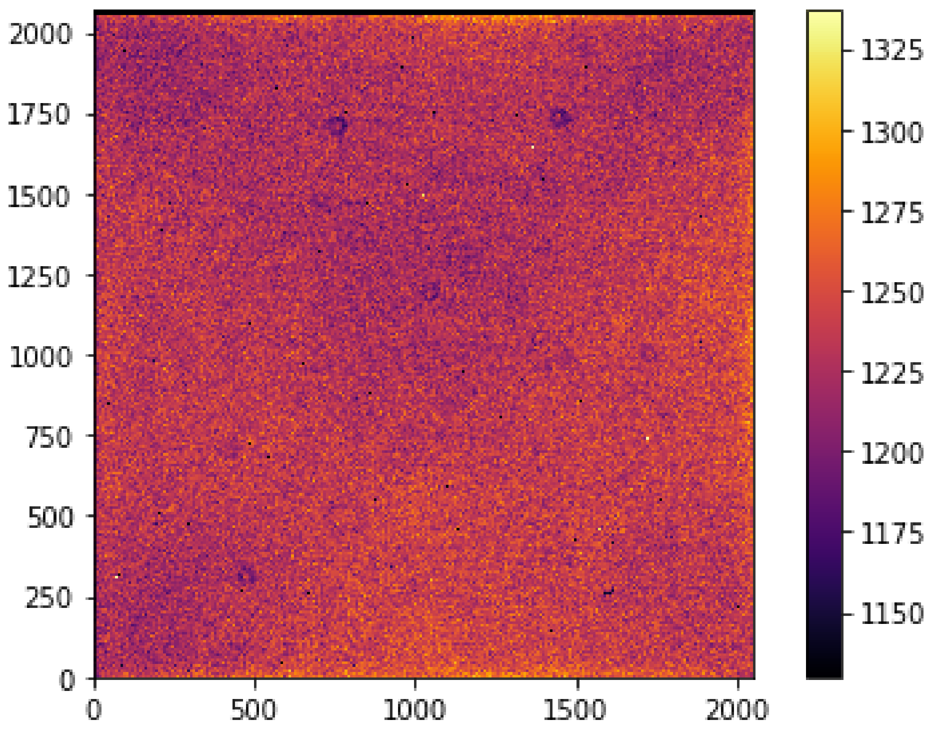
```
# bias: make bias function and master bias
from astropy.visualization import ZScaleInterval

# make a list
bias=[]
# load the data
#for fname in calibrated_data():

#for frame in calibrated_data.filter(object=None, filter=None).data():
# ccd read the data
 #   data,
 #   bias = ccd.ccd_process(
 #       CCDData(frame, unit="adu"), oscan="[2049:2080,1:2068]", trim="[1:
 ↪2048,1:2068]"
  #  )

# load that data into the list made at the top
#biases.append(bias)
#gives median stack of list until 10th item
master_bias = ccdp.combine(calibrated_data[:11])
#plt.imshow(master_bias, vmin=)
vmin, vmax = ZScaleInterval().get_limits(master_bias)
plt.imshow(master_bias, cmap="inferno", vmin=vmin, vmax=vmax, origin="lower")
plt.colorbar()

plt.tight_layout()
```



```
[ ]:
```

Step 1: Generate median dome flat in each filter

```
[42]: base = ("/Users/alix/Bunny/UH88 Tek RAW data/")
full_coll = ccdp.ImageFileCollection(
    base,
    keywords=["OBJECT", "FILTER", "EXPTIME", "GAIN"],
    glob_exclude="*tek01[1,2].fits",
)
# median dome flat function
dome_flats = {
    "r'": [],
    "i'": [],
    "z'": [],
}
for filt_name in dome_flats:
    for frame in full_coll.filter(filter=filt_name, object="dome flat .?",␣
 ↪regex_match=True).data():
        flat = ccdp.ccd_process(
            CCDData(frame, unit="adu"),
            oscan="[2049:2080,1:2068]",
            trim="[1:2048,1:2068]",
            master_bias=master_bias,
        )
        dome_flats[filt_name].append(flat)
#print(dome_flats)
master_dome_flats = {filt:ccdp.combine(flats) for filt, flats in dome_flats.
 ↪items()}
#master_dome_flats = ccdp.combine(flats) for filt, flats in dome_flats.items()

#master_dome_flats.write("master_dome_flats_r.fits")
[master_flat.write(f"master_dome_flat_{filt[0]}.fits") for filt, master_flat in␣
 ↪master_dome_flats.items()];
```

```
{"r'": [CCDData([[-2.07619091e+04,  3.64050000e+04,  3.60933636e+04, …,
           3.36035455e+04,  3.33122727e+04,  3.86143636e+04],
         [-8.01818182e+01,  1.23675455e+04,  1.26500000e+04, …,
           1.23927273e+04,  1.23213636e+04,  1.95656364e+04],
         [-8.09090909e+01,  1.24940000e+04,  1.21234545e+04, …,
           1.22045455e+04,  1.23512727e+04,  1.92990909e+04],
         …,
         [-8.18181818e-01,  1.08181818e+01, -5.00000000e+00, …,
           4.45454545e+00,  1.27272727e+00, -1.13636364e+01],
         [ 4.90909091e+00, -1.14545455e+01,  1.48181818e+01, …,
           1.72727273e+01,  1.60000000e+01,  1.49090909e+01],
         [ 3.54545455e+00,  9.90909091e+00,  1.41818182e+01, …,
           7.63636364e+00,  1.06363636e+01, -9.09090909e-02]]),
CCDData([[-1.57974091e+04,  2.66005000e+04,  2.61548636e+04, …,
           2.42520455e+04,  2.41287727e+04,  3.45578636e+04],
         [-6.26818182e+01,  9.25904545e+03,  9.26450000e+03, …,
           9.22722727e+03,  9.11986364e+03,  1.45581364e+04],
         [-6.49090909e+01,  9.25700000e+03,  9.03045455e+03, …,
           9.01454545e+03,  9.03227273e+03,  1.43280909e+04],
         …,
         [ 3.68181818e+00,  1.13181818e+01,  1.05000000e+01, …,
           2.95454545e+00, -4.22727273e+00, -3.08636364e+01],
         [-5.90909091e-01,  1.70454545e+01,  2.43181818e+01, …,
           2.77272727e+00,  1.25000000e+01,  1.74090909e+01],
         [ 5.04545455e+00, -1.05909091e+01,  1.06818182e+01, …,
           1.13636364e+00,  1.71363636e+01,  1.44090909e+01]]),
CCDData([[-2.03799091e+04,  3.59040000e+04,  3.53473636e+04, …,
           3.32465455e+04,  3.30812727e+04,  3.89963636e+04],
         [-8.61818182e+01,  1.20875455e+04,  1.23250000e+04, …,
           1.21107273e+04,  1.20683636e+04,  1.92766364e+04],
         [-8.19090909e+01,  1.22470000e+04,  1.20024545e+04, …,
           1.19645455e+04,  1.20842727e+04,  1.89650909e+04],
         …,
         [ 1.81818182e-01,  6.81818182e+00,  1.50000000e+01, …,
          -1.45454545e+01,  8.27272727e+00, -5.36363636e+00],
         [ 1.40909091e+00, -1.95454545e+00,  9.31818182e+00, …,
           1.77272727e+00,  1.50000000e+00, -2.59090909e+00],
         [ 5.04545455e+00,  1.04090909e+01,  2.36818182e+01, …,
           1.01363636e+01, -3.86363636e+00,  6.40909091e+00]])], "i'":
[CCDData([[-1.90189091e+04,  3.46990000e+04,  3.37493636e+04, …,
           2.97835455e+04,  2.93852727e+04,  4.03573636e+04],
         [-1.14681818e+02,  1.17300455e+04,  1.19705000e+04, …,
           1.13502273e+04,  1.13128636e+04,  1.74991364e+04],
         [-1.14409091e+02,  1.16195000e+04,  1.15739545e+04, …,
           1.09240455e+04,  1.12507727e+04,  1.72875909e+04],
         …,
         [ 4.18181818e+00, -4.18181818e+00,  0.00000000e+00, …,
          -9.54545455e+00,  1.62727273e+01,  1.16363636e+01],
         [ 9.09090909e-01, -1.34545455e+01,  7.81818182e+00, …,
           1.82727273e+01, -4.00000000e+00,  8.90909091e+00],
         [ 8.04545455e+00, -8.59090909e+00, -3.18181818e-01, …,
          -7.86363636e+00,  2.01363636e+01,  2.44090909e+01]]),
CCDData([[-1.88024091e+04,  3.40895000e+04,  3.32968636e+04, …,
           2.92180455e+04,  2.90877727e+04,  4.05738636e+04],
         [-1.15681818e+02,  1.13960455e+04,  1.17075000e+04, …,
           1.10402273e+04,  1.09708636e+04,  1.70771364e+04],
         [-1.18409091e+02,  1.16435000e+04,  1.12799545e+04, …,
           1.07340455e+04,  1.11777727e+04,  1.69475909e+04],
         …,
         [ 3.68181818e+00, -2.68181818e+00,  7.50000000e+00, …,
          -1.04545455e+00,  4.77272727e+00,  7.13636364e+00],
         [ 2.90909091e+00, -1.34545455e+01,  1.68181818e+01, …,
           6.27272727e+00, -6.00000000e+00,  1.09090909e+01],
         [ 6.54545455e+00,  4.90909091e+00, -3.81818182e+00, …,
          -5.36363636e+00,  8.63636364e+00,  3.90909091e+00]]),
CCDData([[-1.78974091e+04,  3.24865000e+04,  3.17818636e+04, …,
           2.77190455e+04,  2.77707727e+04,  3.88928636e+04],
         [-1.16681818e+02,  1.08290455e+04,  1.09025000e+04, …,
           1.05332273e+04,  1.03508636e+04,  1.64471364e+04],
         [-1.18409091e+02,  1.11315000e+04,  1.07249545e+04, …,
           1.05110455e+04,  1.03457727e+04,  1.61765909e+04],
         …,
         [ 4.18181818e+00, -1.81818182e-01,  1.70000000e+01, …,
          -1.35454545e+01, -1.37272727e+01, -1.83636364e+01],
         [ 3.40909091e+00,  9.04545455e+00,  7.31818182e+00, …,
          -5.22727273e+00, -1.35000000e+01, -5.90909091e-01],
         [ 2.04545455e+00,  7.40909091e+00,  2.68181818e+00, …,
          -2.78636364e+01, -1.38636364e+01,  5.40909091e+00]]),
CCDData([[-1.80789091e+04,  3.26710000e+04,  3.16813636e+04, …,
           2.82865455e+04,  2.80512727e+04,  3.93893636e+04],
         [-1.10681818e+02,  1.09120455e+04,  1.12435000e+04, …,
           1.05362273e+04,  1.05948636e+04,  1.66301364e+04],
         [-1.18909091e+02,  1.10640000e+04,  1.08844545e+04, …,
           1.04105455e+04,  1.05242727e+04,  1.63460909e+04],
         …,
         [ 8.18181818e+00,  6.81818182e+00,  5.00000000e+00, …,
           4.45454545e+00,  7.27272727e+00,  1.26363636e+01],
         [ 3.90909091e+00, -2.45454545e+00,  3.81818182e+00, …,
          -5.72727273e+00,  4.00000000e+00,  1.90909091e+00],
         [ 8.04545455e+00,  1.44090909e+01, -1.93181818e+01, …,
          -6.86363636e+00,  3.71363636e+01,  1.40909091e+00]]),
CCDData([[-1.79464091e+04,  3.17405000e+04,  3.11248636e+04, …,
           2.77460455e+04,  2.74367727e+04,  3.87388636e+04],
         [-1.15181818e+02,  1.07285455e+04,  1.10660000e+04, …,
           1.03597273e+04,  1.04483636e+04,  1.65086364e+04],
         [-1.12909091e+02,  1.09250000e+04,  1.07664545e+04, …,
           1.03805455e+04,  1.04582727e+04,  1.60110909e+04],
         …,
         [ 2.18181818e+00, -1.51818182e+01,  7.00000000e+00, …,
           1.94545455e+01, -7.27272727e-01,  7.63636364e+00],
         [-1.09090909e+00,  8.54545455e+00, -1.11818182e+01, …,
          -2.87272727e+01,  3.00000000e+00, -9.09090909e+00],
         [ 2.54545455e+00, -1.20909091e+01, -8.18181818e-01, …,
           6.36363636e-01, -3.63636364e-01,  2.29090909e+01]])], "z'":
[CCDData([[-1.98229091e+04,  3.59820000e+04,  3.57023636e+04, …,
           3.01755455e+04,  3.01822727e+04,  3.95533636e+04],
         [-1.43181818e+02,  1.26685455e+04,  1.28990000e+04, …,
           1.19097273e+04,  1.17663636e+04,  1.78376364e+04],
         [-1.42409091e+02,  1.28205000e+04,  1.27239545e+04, …,
           1.16990455e+04,  1.17747727e+04,  1.76045909e+04],
         …,
         [ 6.68181818e+00, -3.68181818e+00,  1.05000000e+01, …,
           1.09545455e+01, -8.22727273e+00, -3.86363636e+00],
         [ 3.90909091e+00,  3.54545455e+00,  1.38181818e+01, …,
          -8.72727273e+00,  7.00000000e+00,  1.19090909e+01],
         [ 1.54545455e+00, -3.09090909e+00,  5.18181818e+00, …,
           1.06363636e+01,  1.86363636e+01,  2.09090909e+01]]),
CCDData([[-1.91154091e+04,  3.59525000e+04,  3.53138636e+04, …,
           2.99630455e+04,  2.93457727e+04,  4.02608636e+04],
         [-1.48681818e+02,  1.25270455e+04,  1.27575000e+04, …,
           1.17272273e+04,  1.16238636e+04,  1.76071364e+04],
         [-1.47909091e+02,  1.27300000e+04,  1.25414545e+04, …,
           1.15025455e+04,  1.16422727e+04,  1.75060909e+04],
         …,
         [ 4.68181818e+00,  6.31818182e+00, -1.25000000e+01, …,
           1.69545455e+01,  7.72727273e-01, -1.18636364e+01],
         [ 1.40909091e+00,  3.40454545e+01,  5.31818182e+00, …,
           9.77272727e+00, -3.50000000e+00,  2.40909091e+00],
         [ 3.54545455e+00,  3.90909091e+00, -1.81818182e+00, …,
           4.63636364e+00,  3.06363636e+01, -7.09090909e+00]]),
CCDData([[-1.88099091e+04,  3.48740000e+04,  3.45273636e+04, …,
           2.90545455e+04,  2.87732727e+04,  3.97453636e+04],
         [-1.49681818e+02,  1.20440455e+04,  1.23535000e+04, …,
           1.15752273e+04,  1.11878636e+04,  1.71281364e+04],
         [-1.43409091e+02,  1.23415000e+04,  1.20969545e+04, …,
           1.12610455e+04,  1.13617727e+04,  1.68255909e+04],
         …,
         [ 7.18181818e+00,  3.81818182e+00,  1.10000000e+01, …,
          -1.25454545e+01, -2.72727273e+00, -1.36363636e+00],
         [ 2.40909091e+00,  1.04545455e+00,  1.73181818e+01, …,
          -1.72272727e+01, -1.15000000e+01,  5.40909091e+00],
         [ 2.54545455e+00, -1.70909091e+01, -3.08181818e+01, …,
          -1.93636364e+01,  4.63636364e+00, -3.09090909e+00]]),
CCDData([[-1.79904091e+04,  3.33685000e+04,  3.29848636e+04, …,
           2.85850455e+04,  2.76077727e+04,  3.78228636e+04],
         [-1.37681818e+02,  1.16590455e+04,  1.18725000e+04, …,
           1.11052273e+04,  1.07678636e+04,  1.62761364e+04],
         [-1.34409091e+02,  1.16545000e+04,  1.15569545e+04, …,
           1.07730455e+04,  1.08657727e+04,  1.63045909e+04],
         …,
         [ 5.68181818e+00,  1.83181818e+01,  2.50000000e+00, …,
           9.95454545e+00,  1.07727273e+01, -3.86363636e+00],
         [ 4.40909091e+00, -9.95454545e+00,  2.13181818e+01, …,
          -7.22727273e+00,  1.35000000e+01, -1.59090909e+00],
         [ 3.04545455e+00, -1.25909091e+01, -3.31818182e+00, …,
          -8.63636364e-01, -4.86363636e+00,  2.14090909e+01]]),
CCDData([[-1.35409091e+04,  2.32030000e+04,  2.31063636e+04, …,
           1.92495455e+04,  1.89172727e+04,  2.66343636e+04],
         [-1.04681818e+02,  8.41304545e+03,  8.65850000e+03, …,
           7.97422727e+03,  7.80786364e+03,  1.15291364e+04],
         [-9.99090909e+01,  8.59700000e+03,  8.32345455e+03, …,
           7.87854545e+03,  7.74427273e+03,  1.16220909e+04],
         …,
         [ 2.18181818e+00,  8.81818182e+00,  1.50000000e+01, …,
           4.54545455e-01, -1.07272727e+01,  6.36363636e-01],
         [ 4.09090909e-01, -8.95454545e+00, -6.68181818e+00, …,
          -8.22727273e+00,  5.00000000e-01, -3.59090909e+00],
         [ 2.54545455e+00, -8.09090909e+00, -1.58181818e+01, …,
          -1.23636364e+01, -9.36363636e+00,  4.90909091e+00]])]}
```

```
      ␣
↪---------------------------------------------------------------------------

    OSError                                   Traceback (most recent call␣
↪last)

    <ipython-input-42-f79e786d459e> in <module>
     25
```

```
        26 #master_dome_flats.write("master_dome_flats_r.fits")
   ---> 27 [master_flat.write(f"master_dome_flat_{filt[0]}.fits") for filt,␣
↪master_flat in master_dome_flats.items()];


       <ipython-input-42-f79e786d459e> in <listcomp>(.0)
        25
        26 #master_dome_flats.write("master_dome_flats_r.fits")
   ---> 27 [master_flat.write(f"master_dome_flat_{filt[0]}.fits") for filt,␣
↪master_flat in master_dome_flats.items()];


       ~/opt/anaconda3/lib/python3.7/site-packages/astropy/nddata/mixins/ndio.
↪py in __call__(self, *args, **kwargs)
        94
        95     def __call__(self, *args, **kwargs):
   ---> 96         registry.write(self._instance, *args, **kwargs)
        97
        98


       ~/opt/anaconda3/lib/python3.7/site-packages/astropy/io/registry.py in␣
↪write(data, format, *args, **kwargs)
       564
       565     writer = get_writer(format, data.__class__)
   --> 566     writer(data, *args, **kwargs)
       567
       568


       ~/opt/anaconda3/lib/python3.7/site-packages/astropy/nddata/ccddata.py in␣
↪fits_ccddata_writer(ccd_data, filename, hdu_mask, hdu_uncertainty, hdu_flags,␣
↪key_uncertainty_type, **kwd)
       674         hdu_mask=hdu_mask, hdu_uncertainty=hdu_uncertainty,
       675         key_uncertainty_type=key_uncertainty_type,␣
↪hdu_flags=hdu_flags)
   --> 676     hdu.writeto(filename, **kwd)
       677
       678


       ~/opt/anaconda3/lib/python3.7/site-packages/astropy/utils/decorators.py␣
↪in wrapper(*args, **kwargs)
       519                             kwargs[new_name[i]] = value
       520
   --> 521             return function(*args, **kwargs)
       522
```

```
    523         return wrapper


    ~/opt/anaconda3/lib/python3.7/site-packages/astropy/io/fits/hdu/hdulist.
↪py in writeto(self, fileobj, output_verify, overwrite, checksum)
    917         # This can accept an open file object that's open to write␣
↪only, or in
    918         # append/update modes but only if the file doesn't exist.
--> 919         fileobj = _File(fileobj, mode=mode, overwrite=overwrite)
    920         hdulist = self.fromfile(fileobj)
    921         try:


    ~/opt/anaconda3/lib/python3.7/site-packages/astropy/utils/decorators.py␣
↪in wrapper(*args, **kwargs)
    519                             kwargs[new_name[i]] = value
    520
--> 521             return function(*args, **kwargs)
    522
    523         return wrapper


    ~/opt/anaconda3/lib/python3.7/site-packages/astropy/io/fits/file.py in␣
↪__init__(self, fileobj, mode, memmap, overwrite, cache)
    176             self._open_fileobj(fileobj, mode, overwrite)
    177         elif isinstance(fileobj, str):
--> 178             self._open_filename(fileobj, mode, overwrite)
    179         else:
    180             self._open_filelike(fileobj, mode, overwrite)


    ~/opt/anaconda3/lib/python3.7/site-packages/astropy/io/fits/file.py in␣
↪_open_filename(self, filename, mode, overwrite)
    542
    543         if mode == 'ostream':
--> 544             self._overwrite_existing(overwrite, None, True)
    545
    546         if os.path.exists(self.name):


    ~/opt/anaconda3/lib/python3.7/site-packages/astropy/io/fits/file.py in␣
↪_overwrite_existing(self, overwrite, fileobj, closed)
    436                     os.remove(self.name)
    437             else:
--> 438                 raise OSError("File {!r} already exists.".
↪format(self.name))
    439
```

```
440     def _try_read_compressed(self, obj_or_name, magic, mode, ext=''):


OSError: File 'master_dome_flat_r.fits' already exists.
```

[21]:

```
print("hello exoworld")
```

```
hello exoworld
```

Step 2: Description of images

Step 3: Apply dome flat and explain how correction works

[ ]:

```
# apply dome flat
#dome_Flat()
```

"Dome flats are images of the inside of the dome (typically of a smooth surface, not of the dome itself), illuminated by some light source in the dome. For smaller telescopes an electroluminescent or LED illuminated panel can be used as the light source," (https://mwcraig.github.io/ccd-as-book/05-00-Flat-corrections.html)

[ ]:

```

```

Step 4: Median sky flat for each filter

[60]:

```
# function for sky flat # same as above with normalization of dome flats
# median sky flat function
import ccdproc as ccdp

sky_flats = {
    "r'": [],
    "i'": [],
    "z'": [],

}

for filt_name in sky_flats:
    # why .data ?
    for frame in full_coll.filter(filter=filt_name, object="sky flat .?",␣
 ↪regex_match=True).data():
        flat = ccdp.ccd_process(
            CCDData(frame, unit="adu"),
            oscan="[2049:2080,1:2068]",
            trim="[1:2048,1:2068]",
            master_bias=master_bias,
            #now flat correct using dome flats
            master_flat = master_dome_flats[filt_name]
        )
        sky_flats[filt_name].append(flat)
#the filt for each is r' z' i' and the sky is the values that the keys link up␣
↪to (lists) with sky_flats is dictionary(with three keys (each filter)),
#the three keys lead to three values (three lists)
# for statement accesses key and value separately
# master_sky_flat is declared we know bc the = and {}; combine returns keys and␣
↪values pair sky and filt are that pair bc can handle a string key and a list␣
↪value
master_sky_flats = {filt_name_stringkey:ccdp.combine(sky_flat_list) for␣
↪filt_name_stringkey, sky_flat_list in sky_flats.items()}


#master_sky_flats = {filt:ccdp.combine(flats) for filt, sky in sky_flats.
↪items()}
[master_flat.write(f"master_sky_flat_{filt[0]}.fits", overwrite=True) for filt,␣
↪master_flat in master_sky_flats.items()];
```

Step 5: What structures remain after correction compared to original sky flats?

Step 6: Smooth curve with various filters - explain which you tried and which filters worked and why they worked

[62]:

```
base = ("/Users/alix/Bunny/UH88 Tek RAW data/")
full_coll = ccdp.ImageFileCollection(
    base,
    keywords=["OBJECT", "FILTER", "EXPTIME", "GAIN"],
    glob_exclude="*tek01[1,2].fits",
)
# median filtered flat function
filtered_flats = {
    "r'": [],
    "i'": [],
    "z'": [],
}
for filt_name in filtered_flats:
#     for frame in full_coll.filter(filter=filt_name, object="filtered flat .?
↪", regex_match=True).data():
#         flat = ccdp.ccd_process(
#             CCDData(frame, unit="adu"),
#             oscan="[2049:2080,1:2068]",
#             trim="[1:2048,1:2068]",
#             master_bias=master_bias,
#             master_flat = master_dome_flats[filt_name]
#         )
    filt_flat=ccdp.median_filter(master_dome_flats[filt_name], size=20)
    filtered_flats[filt_name].append(filt_flat)
    print(filt_flat)
```

```
filtered_flats = {filt:ccdp.combine(flats) for filt, flats in filtered_flats.
 ↪items()}
#master_dome_flats = ccdp.combine(flats) for filt, flats in dome_flats.items()

#master_dome_flats.write("master_dome_flats_r.fits")
[master_flat.write(f"master_filtered_flat_{filt[0]}.fits") for filt,␣
 ↪master_flat in filtered_flats.items()];
```

Step 7: Apply the appropriate smoothed sky flat correction to each of the science images taken during the night.

[ ]:

```
files = log[file].loc[]
```

[68]:

```
# median filtered flat function
ccc = {
    "r'": [],
    "i'": [],
    "z'": [],
}
for filt_name in ccc:
    for frame in full_coll.ccds(filter=filt_name, object="2M2249",␣
 ↪ccd_kwargs=dict(unit="adu")):
        flat = ccdp.ccd_process(
            frame,
            oscan="[2049:2080,1:2068]",
            trim="[1:2048,1:2068]",
            master_bias=master_bias,
            master_flat = master_dome_flats[filt_name]
        )
    ccc[filt_name].append(flat)

# filtered_flats = {filt:ccdp.combine(flats) for filt, flats in filtered_flats.
 ↪items()}
#master_dome_flats = ccdp.combine(flats) for filt, flats in dome_flats.items()

#master_dome_flats.write("master_dome_flats_r.fits")
[master_flat[0].write(f"2M2249_{filt[0]}_dome.fits") for filt, master_flat in␣
 ↪ccc.items()];
```

[ ]:

```
```

[ ]:

```
```