HW02

In this homework assignment, you will learn how to do 2-D CCD combining. I will use **Python >= 3.6** for the demonstration.

Prob 01 [Master Bias and Flat]

In this first problem, I will show you how the 2-D median combine works in a rather general way. This is the way of coding used in virtually any field of science (including math, physics, astronomy, biology, and statistics).

First, let's generate fake data from a test case, say:

- We have a CCD with 10 (height) by 30 (width) pixels
- gain = 2.3 electrons / ADU and read noise = 10 electrons (provided by the CCD manufacturer)
- The true bias and flat values (which of course is unknown in real observation) are 1000 and 20000 ADU, respectively.

I made 11 bias and flat images, and saved them in `biass` and `flats` (which are python `list`s). Then I will use `numpy` to median combine them along the "same (x, y)" direction, which is `axis = 0` in `numpy` :

```
import numpy as np
np.random.seed(1)

biass = []
flats = []
NXNY = (10, 30)
BIAS = 1000   # intended bias in ADU (designed by CCD manufacturer)
FLAT = 20000  # intended flat in ADU (fake value for this tutorial)
GAIN = 2.3    # electrons / ADU unit
RDNOISE = 10  # electrons unit

for i in range(19):
    bias = BIAS  # intended offset
    bias = bias + np.random.normal(loc=0, scale=RDNOISE, size=NXNY) / GAIN

    # add read noise in ADU
    biass.append(np.around(bias).astype(int))  # round-off

    flat = bias.copy()  # bias level
    flat = flat + np.random.poisson(lam=20000 * GAIN, size=NXNY) / GAIN
    # intended input flux in ADU.
    # lambda for the poisson must be multiplied by GAIN.
```

```
    flat = flat + np.random.normal(loc=0, scale=RDNOISE, size=NXNY)
    # add read noise in ADU
    flats.append(np.around(flat).astype(int))  # round-off


bias = np.median(biass, axis=0)
flat = np.median(flats, axis=0) - bias
flat = flat / np.average(flat)
print(bias)
print(flat)
```

1. **Show the combined bias and flat frames.**

Hint:

```
from matplotlib import pyplot as plt

fig, axs = plt.subplots(2, 1)
axb, axf = axs[0], axs[1]
axb.set_title("Bias")
axf.set_title("Normalized Flat")

imb = axb.imshow(bias, origin='lower')
fig.colorbar(imb, ax=axb)
imf = axf.imshow(flat, origin='lower')
fig.colorbar(imf, ax=axf)
plt.tight_layout()
```

ADDITIONAL QUESTION: Why did I use average for flat, not the median?

- **ANSWER**: Traditionally we normalize flat frames with the average value, not by median value. The reason is that calculating median will take longer time than average. Someone may ask: "But average is vulnerable to some high-valued pixels," and yes that's true and that's why we used median combine for bias and flat combine process. But `flat` itself is already the result of the median combine of many (19 in this case) images, so the serious outliers have already been gone. By serious outliers, I mean something like cosmic-rays. Moreover, there are a myriad of samples (e.g., 1K by 1K CCD has 1 mega pixels) when we are normalizing flat, which is different than image combination, so few outliers do not significantly affect the average value. In that sense, averaging suffice to normalize the flat frame.

The example above is a general way to do medianing. But in observational astronomy, we need few more technical steps since we are using FITS format. We will use `ccdproc` in the future to deal with FITS in python.

Prob 02 [Low Signal-to-Noise Ratio]

[Let's forget about read noise, bias, dark, and flat issues in this problem.]

When the object is too faint, we cannot even see whether there is actually an object in the image. So even the computer fails to find the correct centroid or cannot fit any function to the data. One possibility to circumvent this issue is to do, e.g., average-combine.

> How can that work..!?
>
> - Remember that Poisson error is square root of the mean value: $\text{pixel value} = \text{value} \pm \sqrt{\text{value}}$.
> - Assume pixels of sky: $p_s = 100 \pm \sqrt{100}[\text{ADU}]$ after 1 s exposure.
> - Assume pixels of object: $p_o (= \text{obj} \times e^{-\tau} + \text{sky}) = 101 \pm \sqrt{101}[\text{ADU}]$ after 1 s exposure.
> - For t-sec exposure, $\text{value} = (\text{value at 1s}) \times t$, so $p_s = 100t \pm \sqrt{100t}$ and $p_o = 101t \pm \sqrt{101t}$.
> - If, e.g., $t = 1000$, $p_s = 10^5 \pm 316$ and $p_o = 101000 \pm 318$, so now the object pixels are distinguishable from the sky pixels.
> - Increasing t is somewhat similar to adding many images of the same target.

Consider we have 9 pre-processed images of the same object's spectrum, but very faint (S/N ratio ~ 3). horizontal = wavelength, vertical = spatial directions.

```
import numpy as np
from matplotlib import pyplot as plt
np.random.seed(1)

def fake_obj():
    ''' Makes a new fake object frame (pre-processed)
    '''
    x = np.arange(NX)
    sky = 10 + 0.1 * x + np.random.normal(loc=0, scale=1, size=(NY, NX))
    obj = 1.5 + 0.001*(x/4 - 12)**2
    img = sky.copy()
    img[4, :] += obj
    return img

N = 9
imgs = []
for i in range(N):
    imgs.append(fake_obj())
```

1. **Display all the 9 sample images. Can you see where our target's spectrum is?**
2. **Combine all those into one single file by medianing and averaging, and see if they differ.**

Hint:

```
ncol = 3

fig, axs = plt.subplots(N//ncol + 1, ncol)
for i, img in enumerate(imgs):
    ax = axs[i//ncol, i%ncol]
    im = ax.imshow(img, vmin=9, vmax=15, origin='lower')
    fig.colorbar(im, ax=ax)
    ax.set_title(f"sample {i+1}")

lastrow = dict(avg=np.average(imgs, axis=0),
               med=np.median(imgs, axis=0),
               avg_minus_med=avg-med)

for i, k in enumerate(lastrow.keys()):
    ax = axs[-1, i]
    im = ax.imshow(lastrow[k], origin='lower')
    fig.colorbar(im, ax=ax)
    ax.set_title(k)

plt.tight_layout()
```