Instructions

Challenge: Binapprox on FITS

Problem | Solutions

Use the binapprox algorithm to efficiently estimate the median of each pixel from a set of astronomy images in FITS files.

Write `median_bins_fits` and `median_approx_fits` functions that take a list of FITS filenames and the number of bins. They should behave like the previous problem's functions, but operate over 2D astronomy images from the FITS files.

Expand your code from the previous problem to handle 2D arrays.

To calculate the mean and standard deviation of the FITS files, we've given you a `running_stats` helper function which calculates them in a single pass without holding the images in memory. It returns a tuple of the mean and standard deviation.

While your functions from the last problem returned single numbers for the mean, standard deviation, median and bins, they should now return arrays, since we want to calculate these quantities across the stack of FITS files.

The arrays should be of the same shape as the original FITS files, i.e. `(200, 200)` and have one extra dimension for the bins.

To check that your functions work correctly, you can compare them against the following example. Since the arrays are quite big, we'll only look at the central values:

```
>>> mean, std, left_bin, bins = median_bins_fits(['image0.fits', 'image1.fits', 'image2.fits'], 5)
>>> median = median_approx_fits(['image0.fits', 'image1.fits', 'image2.fits'], 5)
>>> mean[100, 100]
0.018398113548755646
>>> std[100, 100]
0.010496325561403296
>>> left_bin[100, 100]
0.0
>>> bins[100, 100, :]
array([ 0.,  2.,  0.,  0.,  0.])
>>> median[100, 100]
0.014199583324194326
```

If you call your function using all the 12 FITS files, you should get:

```
>>> mean, std, left_bin, bins = median_bins_fits(['image{}.fits'.format(str(i)) for i in range(11)], 4)
>>> median = median_approx_fits(['image{}.fits'.format(str(i)) for i in range(11)], 4)
>>> mean[100, 100]
0.014677493579008362
>>> std[100, 100]
0.0082645758594887056
>>> left_bin[100, 100]
0.0
>>> bins[100, 100, :]
array([ 4.,  5.,  0.,  0.])
>>> median[100, 100]
0.012611349614136185
```