MPhys Lab: Exploring the Hubble Ultra Deep Field

Stephen Wilkins
s.wilkins@sussex.ac.uk

University of Sussex — July 16, 2020

Introduction

The Hubble Ultra Deep Field (HUDF), also known (due to a re-branding exercise) as the eXtreme Deep Field (XDF), is the *Hubble Space Telescope*'s deepest view of the Universe[1]. The HUDF

The aim of this project is to become familiar with analysing Hubble imaging using `python`. The ultimately goal is to identify candidate high-redshift galaxies.

Astronomical images

Astronomical cameras are typically sensitive to a broad wavelength range. To gain spectral information we must combine the camera (detector) with a filter that is only transparent to light over a (typically) small range of wavelengths, typically $\mathcal{O}(100nm)$. These individual images are monochromatic (i.e. they contain no colour information) and can thus be represented by a simple two-dimensional array where each element of the array denotes the signal in that pixel. Images in several filters, or bands, can be combined, as we'll see to generate colour images.

Astronomical images are typically held in the Flexible Image Transport System (FITS). In addition to the raw image data FITS file can also contain meta-data in a header and even tabular data. Examples of this meta-data are the values necessary to convert from pixel coordinates (e.g. (x, y)) to a position on the sky.

Uncertainties

Having quantified uncertainties (errors) is a critical ingredient in science. There are actually several different ways to estimate the uncertainty on, for example, the flux of an object. However, the best is when a noise (often expressed as a weight) image is provided.

About the HUDF images

Hubble imaging of the HUDF consists of imaging in 8 optical and near-IR filters stretching from the blue end of the optical (400nm) to almost 2000nm in the near-IR. These filters are named (from blue to red) f435w, f606w, f775w, f850lp, f105w, f125w, f140w, and f160w. The first 4 were obtained with the Advanced Camera for Surveys (ACS) while the final 4 were obtained with Wide Field Camera 3 (WFC3). The filter transmission curves for these filters, showing the fraction of light transmitted through each filter as a function of wavelength, are shown in Figure .

For each filter there are a pair of images: a science (sci) and weight (wht) image. These respectively contain the signal in electrons per second (e/s), and an estimate of the noise in each pixel. The noise can be estimated from the weight according to:

$$\text{noise} = \frac{1}{\sqrt{\text{weight}}}$$

Because of the way these high-level science images were constructed most of the pixels in these images are actually empty (unobserved). For this reason a mask is also provided allowing you to easily mask empty and other unwanted pixels.

[1] http://xdf.ucolick.org

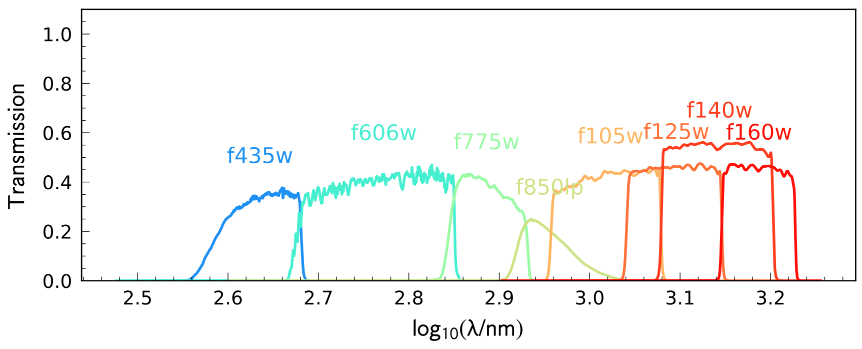

Figure 1: Hubble filters used to observe the HUDF.

This project

In this project you will learn how to analyse *Hubble* images using `python` through a series of task. In addition to `numpy`, `scipy`, and `matplotlib`, you will also need to install `astropy` and `photuils`. To aid you there are a series of examples in `examples/`.

1 Basic image analysis

We'll begin with a few basic `python` image analysis tasks to get you started.

1.1 Working with pixels

First, we'll look at analysing the pixel data in the image. `example1.py` demonstrates how to read in the image data and convert it to an array of pixel values.

Task 1a *Pixel distribution*

First, model the noise as a gaussian centred at zero and estimate σ for the F105W band. **Hint:** there should be no signal contribution to the negative pixels so you can use them to measure σ. To do this first exclude positive pixels. σ will then simply be $-P_{31.7}$ (i.e. the negative of the 31.7th percentile). Next, exclude pixels with magnitude $> 10\sigma$ and plot both a density histogram (**Hint:** use `plt.hist(..., density = True)`) of the pixel distribution and a normal distribution with the same σ as you've just calculated. They won't align perfectly as the pixel distribution unsurprisingly contains more positive pixels.

1.2 Cutting out an image

Often we only want to analyse a small portion (a cutout) of an image instead of the full image. This can be done by slicing the image array, for example `cutout = img[xmin:xmax, xmin:xmax]` or via a `python slice`. An example of slicing is given in `example2.py`.

1.3 Making plots of images

We'll now look at exploring some image data. The image data you've read in is simply stored as a 2D array of pixel values. As such we can simply use `plt.imshow(image)` to produce a plot of the image. `example2.py` demonstrates how to do this.

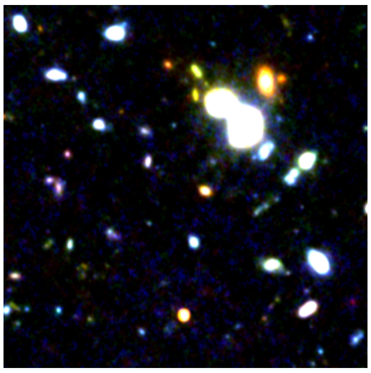

Figure 2: RGB image of the centre of the HUDF created by `example3.py`.

Task 1b *(with optional title)*

Produce plots of each un-masked weight map. You should do this efficiently with a loop: **do not** simply repeat the code 8 times. You should notice that the weight maps for the f435w, f606w, f775w, and f850lp are different from those for f105w, f125w, f140w, and f160w. This is because images in the former filters were obtained using the advanced camera for surveys (ACS) instrument while the latter were obtained with Wide Field Camera 3 (WFC3). ACS and WFC3 have different field-of-views. For the WFC3 filters also notice the "holes" in the weight maps corresponding to bad areas of the detector (camera).

1.4 Combining (stacking) images

A common task is to combine images either taken with the same filter (often) or with different filters (occasionally). Doing so boosts the sensitivity of the image, albeit, in the latter case, at the expense of the loss of spectral information. To optimise the sensitivity images should be combined by weighting each image with its corresponding weight image. An example of this process is shown in `example4.py`.

1.5 Making colour images

Most people's experience with *Hubble* imaging is from the glorious colour images available here. As explained in the introduction *Hubble*'s does not capture 'colour' iamges. Instead images in multiple filters are combined together. To obtain 'full-colour' requires at least 3 filters, thereby mimicing the human visual system. The simplest application is to simply map 3 filters to the red (R), green (G), and blue (B) channels. `example3.py` demonstrates how to do this using 3 of the ACS bands. Figure 1.5 shows one of the outputs of `example3.py`.

Task 1c *(with optional title)*

Using `example3.py` and `example4.py` as guides produce a false-colour image of the entire masked XDF using all 8 filters. You should define 3 groups of consecutive filters (e.g. `['f435w','f606w']`, `['f775w','f850lp']`, `['f105w','f125w','f140w','f160w']`), combine each group, and then combine those stacks together into an RGB image. Congratulations you've now created your own pretty HUDF image. By choosing different filters in each group and playing with the scaling you can make an entirely unique and original version.

2 Detecting and measuring sources

The next part of the project concentrates on identifying, and measuring the properties of sources or objects.

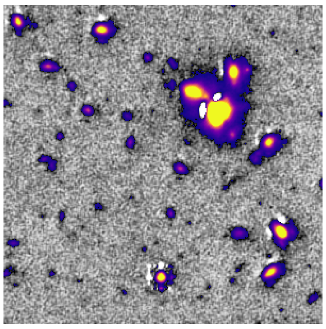

Figure 3: F105W significance image of the centre of the HUDF created by `example5.py`. Pixels coloured in grey have a signal-to-noise < 2 with the colour scale stretching from -2 to 2. Coloured pixels have a signal-to-noise > 2 with a scale stretching from 2 to 50.

Task 2a *(with optional title)*

First of all, following `example4.py`, create a detection science and weight image by stacking the F105W, F125W, F140W, and F160W images together. You will use this image to detect faint sources.

2.1 Significance maps

To identify sources we need to have an estimate of the noise in each pixel. In the context of *Hubble* this is provided by the weight (wht) map in each filter. The values in this image correspond to:

$$\text{weight} = \frac{1}{\text{noise}^2}$$

By dividing the signal (science, or sci) map by the noise map (derived from the weight map) we can obtain a signficance map, essentially the signa-to-noise in every pixel. `example5.py` demonstrates this and Figure 2.1 shows the output.

Task 2b *(with optional title)*

Create a significance map of a 400 pixel wide area centred on (3100, 1800).

2.2 Segmentation

Segmentation (`https://en.wikipedia.org/wiki/Image_segmentation`) is one way of detecting sources (objects) in an image. In the simplest implementation we can identify collections of connected pixels which are all above some threshold. Simple segmentation is controlled by two parameters: the minumum number of connected pixels $n_{\rm pixels}$ and the required significance $\rm threshold$ for each pixel. `example6.py` demonstrates the use of simple segmentation routines using the `astropy.photutils` module with the results of simple segmentation shown in Figure 2.2.

One problem with simple segmentation like this is that nearby objects are often merged together. To overcome this we can use de-blending techniques, again this is demonstrated in `example6.py`.

Task 2c *Detecting sources with segmentation*

Create a segmentation image (with no de-blending) of the same region you looked at in 2b. Assuming $n_{\rm pixels} = 5$ and $\rm threshold = 2.5$. Next, systematically explore the impact of changing $n_{\rm pixels}$ (must be an integer) and $\rm threshold$ on the number of sources detected.

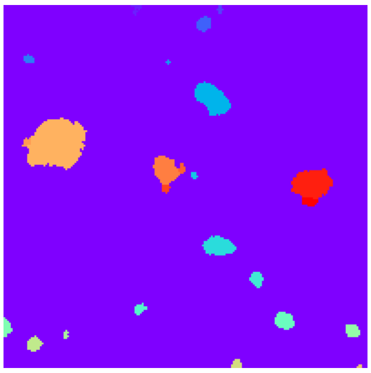

Figure 4: F125W segmentation map assuming $n_{\rm pixels} = 5$ and $\rm threshold = 2.5$.

Task 2d *The impact of de-blending*

Sticking with $n_{\rm pixels} = 5$ and $\rm threshold = 2.5$ now explore the impact of the parameters that control de-blending on the number of sources.

2.3 Measuring the signal (and noise) of sources

Our next task is to measure the signal (and noise) of our sources. Again, there are many of ways of doing this. One of the most popular methods is to place an aperture over each source and calculate the flux through in that aperture. This can be done easily using `photutils.aperture`. However, in this project we'll do something different. Instead of using an apterure we will simply sum the flux in the segmentation region of each object. This is demonstrated in `example7.py`.

Task 2e *Measure the signal of all sources*

Measure the signal (e/s) of all the sources in the region. To do this you can combine the segmentation map with the detection science image. Plot a histogram. Do the same for the de-blended image and discuss the difference.

Task 2f *Make a multi-band catalogue*

Using the original (un-blended) segmentation map measure the signal and noise (or error) of every object in every single filter and create a catalogue using a dictionary. Save this catalogue for use later.

3 Finding distant galaxies

High-redshift galaxies can be identified using the Lyman-break technique. This takes advantage of a strong break in the spectrum of galaxies caused by the absorption of ionising photons by inter-stellar and inter-galactic hydrogen.

3.1 Changing units

The units of the original images are electrons per second (e/s). However, we want units of flux[2], for example in nano-Jansky (nJy). The conversion from from e/s to nJy depends on the observatory, instrument,

[2]Actually spectral flux density.

and filter, and thus is unique for each filter: `example9.py` contains the relevant conversion in the form of a dictionary.

Task 3a *Convert to flux*

Read in the catalogue you created in Task 2f and convert the signal into a flux (nJy) using the conversion dictionary in `example9.py`. Plot f_{f105w}/f_{125w} vs. f_{f850lp}/f_{105w} for all the objects in the catalogue.

3.2 Finding distant galaxies

Firstly, we want to guard against objects which are detected a low-S/N, as these are more likely to be contaminants (or not even real sources). To do this we can simply place a constraint on the signal-to-noise (S/N) in a filter where we know any real high-redshift object should be detected. We are somewahat free to choose the band and threshold but $f_{\rm f125w}$ and a S/N> 10 is a reasonable choice.

Next, we know that high-redshft galaxies have a strong spectral break. If the break falls between two bands A and B we'd expect that $f_{\rm A}/f_{\rm B}$ should be small. Galaxies at $z \sim 7$ have a break between the $f850lp$ and $f105w$ bands. A reasonable choice of ratio upper-limit is ~ 0.4.

We also expect the shape of the continuum above the break to be flat, or even negative (i.e. decreasing to longer-wavelength). Using a pair of bands above the break (e.g. $f105w$ and $f125w$) we can then place an additional constraint allowing us to further weed out contamination. A reasonable choice for ratio lower-limit is ~ 0.75

Finally, any truely high-redshift object should be undetected in any filter shortward of the break. For $z \sim 7$ objects we wouldn't expect them to be detected in $f435w$, $f606w$, or $f775w$. This can be implemented by enforcing that any candidate object is detected at less than S/N$= 2$ in those bands.

In conclusion, our selection criteria can be expressed as follows:

$$S/N(f_{\rm f125w}) > 10$$
$$f_{\rm f850lp}/f_{\rm f105w} < 0.4$$
$$f_{\rm f105w}/f_{\rm f125w} > 0.75$$
$$S/N(f_{\rm f435w}) < 2 \wedge S/N(f_{\rm f606w}) < 2 \wedge S/N(f_{\rm f775w}) < 2$$

Task 3b *Identifiy high-redshift galaxy candidates*

Add the above flux-ratio criteria to your plot from 3a (either as lines or a shaded region). Apply the criteria to your catalogue of objects and highlight any objects meeting the criteria on your plot.

Task 3c *Detection image thumbnail*

Using `example7.py` as a guide make detection image thumbnail of your candidate(s), if you have any.

Task 3d *More thumbnails*

Following on from 3c also make thumbails in each band (**Hint:** use `subplots` for ease) in addition to an RGB thumbnail.