Status report, September 23, 2012.

1 Deconvolution

Objective: generate an image with a constant, symmetrical PSF.
(For now, we stay well away from violating the sampling theorem.)

Summary of procedure:

1. Identify the sources in a catalogue of FITS images using SExtractor.
    - SExtractor reads in a FITS image and catalogues all the astronomical sources it contains.
    - My code automates this by generating catalogues of an entire directory of images, using some special settings.
    - These default settings are (mostly) in the files `prepsfex.sex` (governs how the program runs) and `prepsex.param` (governs what information about the sources is recorded in the output file).
    - The program outputs a file with extension `.cat` (by default, if the image is called `image.fits`, the output will be `image.cat`). This contains information like the position, elongation and flux of the source.
2. Find the images' PSFs using PSFEx.
    - PSFEx uses the `.cat` file from SExtractor, and estimates the PSF in the image.
    - There are a considerable number of options for how this estimation is done and what form the outputs take. After flirting with some of the more sophisticated options, it turns out that the key thing for my purpose is an integrated image of the average PSF for the image. This is saved as FITS.
    - I have automated the PSFEx step to process all the image catalogues in a directory with some default settings imposed (these are kept in `default.psfex`).
3. Use these "true" PSFs to generate "ideal" (Gaussian) PSFs.
    - We wish to deconvolve the image such that it has a constant, symmetrical PSF of finite width.
    - My code reads in the FITS file containing the PSF; it calculates the total flux, and the width of the PSF in the y and x directions. Then it generates the image of a 2D Gaussian which shares these properties. This is the "ideal" PSF.
4. Find the convolution kernels which map between these two versions of the PSF.
    - In order to deconvolve the entire image to this "ideal" PSF, we must find an object which, when convolved with the "true" PSF gives the "ideal" one. This object is the convolution kernel, which is we retrieve by casting the deconvolution procedure as a linear algebra problem.

5. Apply this kernel to the entire image to achieve a constant, known PSF.

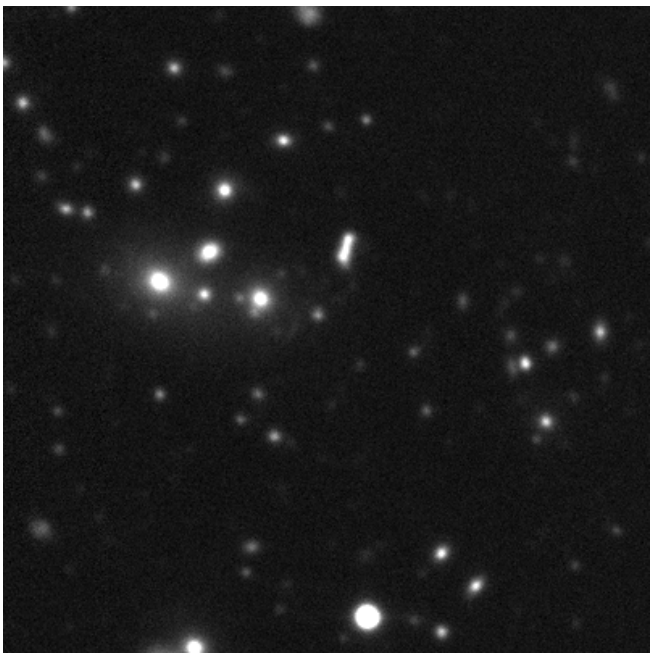
To do:

- Establish a system for documenting everything better – captions and pseudocode &c.

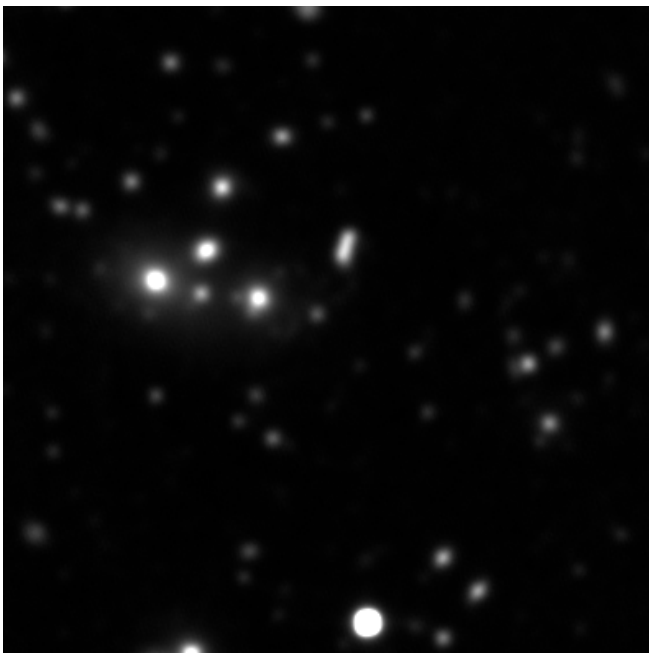
Noise Properties

Here I've applied a few different stretches to the original image and to the deconvolved image. The left-hand side images are all stretched originals and the right-hand side are all stretched deconvolutions. The stertch gets more sever as you go down.

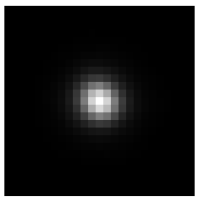
Specific notes:
* I used a 9x9 kernel for these images.
* To change the strength of the stretch, I simply divided the upper bound (you called it "vmax") by 2 and 10.



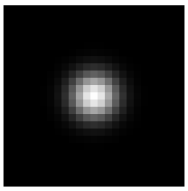
CFHTLS_03_g_sci_rescaled.png



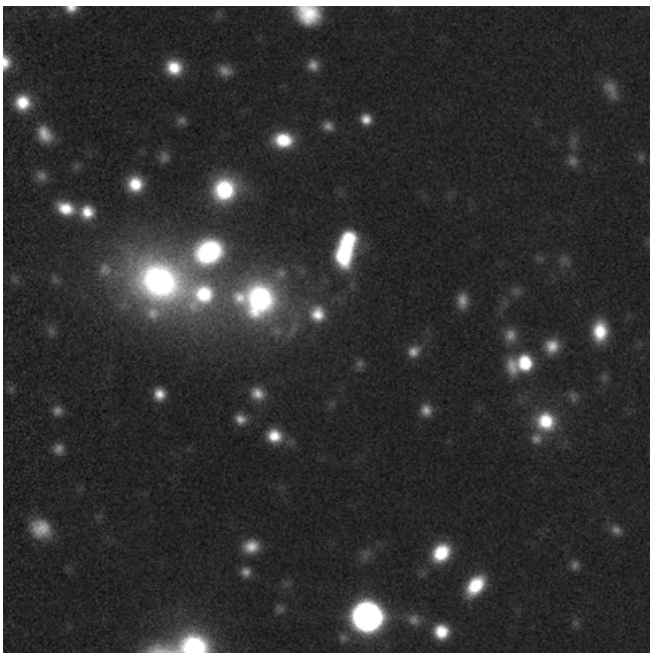
gsc_CFHTLS_03_g_sci_rescaled.png



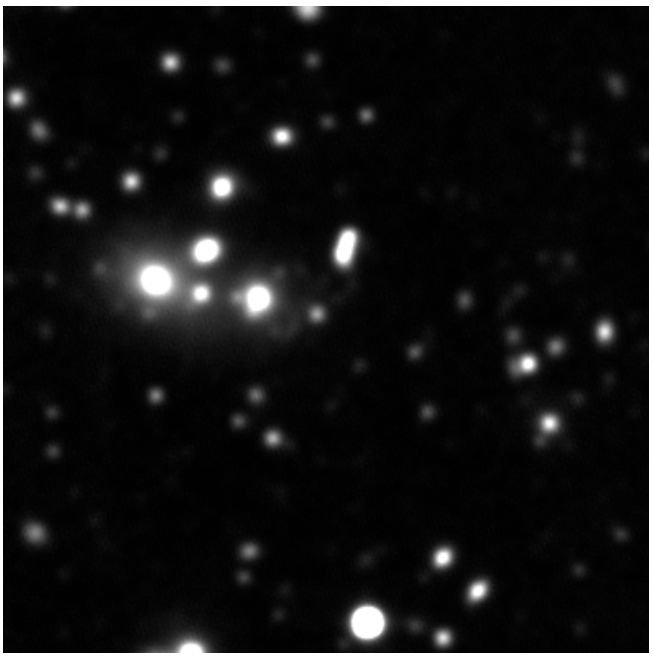
snap_n1_d0_CFHTLS_03_g_sci_z8.png



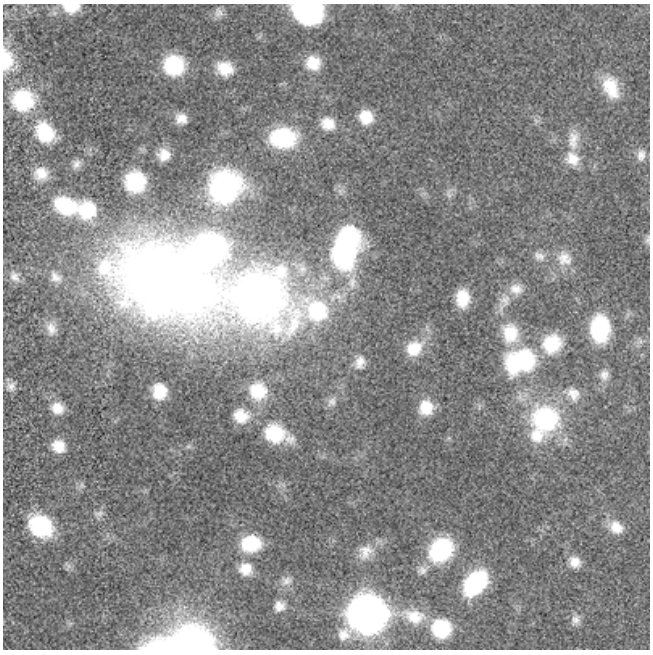
snap_n1_d0_CFHTLS_03_g_sci_Gauss_z8.png



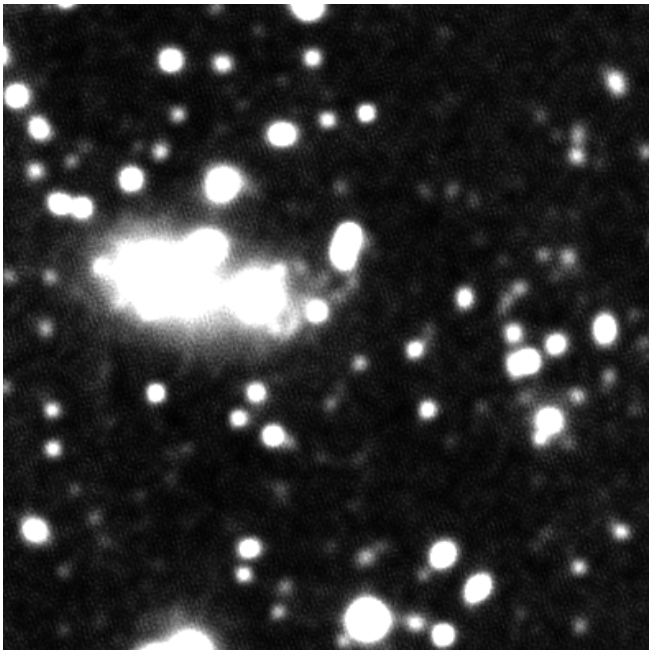
CFHTLS_03_g_sci_rescaledl.png



gsc_CFHTLS_03_g_sci_rescaledl.png



CFHTLS_03_g_sci_rescaledll.png



gsc_CFHTLS_03_g_sci_rescaledll.png

2 Colour composition

Objective: combine images from three band passes to make a colour composite image. Make use of algorithms (Lupton/Wherry stretches) which emphasise faint features while preventing bright objects from dominating.

Summary of progress:

1. Translated (and improved?) the Wherry algorithm from IDL to Python (`wherry.py`). This is what it does:

   (a) `RGB = [readin(R_data),readin(G_data),readin(B_data)] ## *_data can be filename, array of data, or a channel instance`

   (b) `rescale(RGB, scalefactors) ## multiply each band by a given number`

   (c) `rebin(RGB, xrebinfactor,yrebinfactor) ## re-sample images`

   (d) `kill_noise(RGB, cutoff) ## sets all pixels below a threshold to 0`

   (e) `arsinh_stretch(RGB, nonlinearity)`

   - `rad = R_array+G_array+B_array ## collapse images onto each other`
   - `if rad[i,j]==0:  rad[i,j]=1`
   - `factor = arsinh(nonlin*rad)/(nonlin*rad)`
   - `(R_array,G_array,B_array) *= factor`

   (f) `if stauratetowhite==False:  box_scale(RGB)`

   - `maxpixel[i,j] = max(R[i,j],G[i,j],B[i,j]) ## i.e.  find the maximum pixel value of the three arrays`
   - `if maxpixel[i,j] < 1:  maxpixel[i,j]=1`
   - `(R_array,G_array,B_array) /= maxpixel`
   - `(Also translates origin of image if required)`

   (g) `overlay/underlay ## not entirely sure what these are for`

   (h) `scipy.misc.imsave(RGB)`

   - Note: when treating `wherry.py` as a standalone code for making composite images, the user can choose which bands to use for R, G and B.
   - Also, in the IDL version of the code, there is a function devoted to transforming the image data into bytes. When I was translating to Python, I decided that this was an IDL-specific step, and it was unnecessary to implement it as Python does it automatically / there is a way around. Perhaps I didn't fully understand the IDL.

2. Integrated with PJM's HumVI code, so that choice of Lupton/Wherry procedure is an option.
   - Lupton is default. Wherry requires command-line keyword `--wherry` or simply `-w`.

- Again, the user can choose which bands to use for R, G and B – it just depends on the order of the three filenames.
  - After some initial misunderstanding, I now use R=i, G=r, B=g.
- Of course, this bands→colour map is unchanged when when `--wherry` is specified.

To do:

- Noise higher when using arrays than when using channels – investigate.

3 Combining the two strands

The deconvolution and colour composition procedures are not integrated into a pipeline. There are some small challenges in integrating them, but it shouldn't be too difficult. The main danger is that the code becomes too cumbersome and dependent on my directory structure.

4 Notes

- Nothing is checked in to Github.