Discovery Sensitivity in Human Viewable Images

Cato Sandford[*]
with Phil Marshall and David Hogg

December 15, 2012

Abstract

There is much scope in the future for "crowd-sourcing" the analysis of astronomical images. But the images will have to be presented in an intuitive and clear way so that non-experts have a shot at making an informed and useful contribution. In a summer project guided by David Hogg and Phil Marshall, I attempted to satisfy this need by devising a pipeline for the production of deconvolved RGB images from photometric telescope data.

Contents

[*]Department of Physics, New York University, USA; `cato@nyu.edu`

1 Introduction

For centuries, astronomers have continued to astonish the world with pictures of a Universe far richer and more magnificent than any ordinary mind could conceive. Perhaps more than any other field of science, astronomy relies on the analysis of images to draw conclusions about the natural world. Indeed, new and unusual astronomical objects, such as galaxies, nebulae, star clusters, supernovae and so on, appear readily to experienced viewers as they inspect new images (a recent example is the amateur discovery of a quadruple star system, reported in Schwamb et al., 2012); our understanding of the structure of, for example, galaxies can be improved by studying their morphology, by eye[1] (e.g. Lahav, 1995; Lahav et al., 1995; Fortson et al., 2011; Buta, 2011)[2]. Image viewing is a form of data exploration, an important step before making quantitative measurements. But both exploration and measurement involve data modeling, or inference: the viewer interprets the image in the context of a model for that image that they hold. By investigating and modeling this viewing process, we can hope to increase the efficiency of such discoveries, increasing the rate at which true detections are made, and reducing the incidence of false detections.

One way to increase the frequency with which discoveries are made could be to employ a larger number of experienced image viewers to look at images. This approach is being taken by "citizen science" projects such as Galaxy Zoo (citation required[3]). In the Galaxy Zoo system, color composite images of galaxies are presented to large numbers of viewers, who carry out visual morphological studies guided by a short questionnaire. The viewers have a wide range of experience with astronomical images, and the majority of viewers come to the site not having viewed many astronomical images before. The model for astrophysical objects that this group has is therefore data-driven: their ability to spot something new depends partly on their pre-conceived notions of what galaxies look like, but to a greater extent it will depend on the images they have seen in the system before. Given an inspection Zoo and a user base like this, how should we prepare and present the color images of galaxies, to increase the probability of an interesting new feature being detected? This is the question we seek to answer in this paper, taking as our archetypal interesting features the faint arcs caused by gravitational lensing.

Gravitational lenses enable a wide range of science projects, providing a means to measure the mass distributions of galaxies, groups and clusters, independent of their luminosity or dynamical state, and giving us a rare magnified view of the distant universe. At present almost all these projects are limited by the small samples of lenses known, but the wide field surveys coming online have the potential to change that. A number of optical and near infra-red imaging surveys are planned for the next decade that together have been predicted to contain over 10,000 new gravitational lenses, an increase in sample size relative to the present day of around two orders of magnitude (see, for example, Oguri and Marshall (2010); Pawase et al. (2012)[4]; LSST Science Collaboration

[1]**Phil:** indeed, is it not true that there is sometimes little alternative if one wants to draw meaningful conclusions about galaxy structure? Perhaps it is beyond the scope here, but it may be interesting to discuss what can be done by computers and what cannot – the question isn't really addressed head-on even in your theory paper, yet it provides a compelling motivation for this work.

[2]**Phil**: We really want evidence of *visual* inspection being useful, hence these references. Possibly too many.

[3]Cato, Phil: Add more zoo citations. In paper X they found Y, in paper Z they found...

[4]The lensing discoveries of Pawase et al. (2012) arose from the dedicated visual inspection of HST images by two astronomers. They note that for the next generation of telescope missions, the time-demands of their method will be (even more) impractical. But their work lends credence to the claim that visual inspection may yet make valuable

et al. (2009, chapter 12); Refregier et al. (2010, chapter 1, page 8)). Most of these surveys will be carried out on ground-based telescopes, and will be both multi-filter, in many cases multi-epoch, and synoptic, in order to meet a wide range of different science requirements. The thousands of square degrees of sky imaged, and the billions of objects catalogued, will provide a huge mine of data to be searched for rare objects like lenses. We expect image viewing to play a role in this search process, most likely in the form of quality control inspection of the outputs of various automated detection algorithms.

For an image to enable discovery, it needs to be *informative*: that is, it must have high quality, so that interesting features are visible to the viewer, and it must not be confusing, so that interesting features are not mistaken for artefacts and ignored. The quality of an astronomical image is only partly determined by the observing conditions, telescope, and camera: the image processing carried out in software is also important. We have additional information about the image that we can use to improve both its resolution and depth. The stars provide images of the imaging system point spread function (PSF), while our understanding of the detector and the sky background provide a model for the noise in the image: we can attempt to use both of these in reconstructing a higher quality image. There is a significant body of literature on this image restoration process in astronomy (e.g. Richardson, 1972; Nityananda and Narayan, 1982; Skilling and Bryan, 1984; Pina and Puetter, 1993; Magain et al., 1998). We might expect algorithms like this to be important for synoptic surveys, whose resolution and depth varies from image to image, and from filter to filter in a partiular field of view. Combining raw images into a color composite will produce colored artefacts due to the mis-matched resolution in the red, green and blue channels; while the resulting composite will have different (mean) resolution than other elsewhere on the sky, leading to an inhomogeneity that will hinder the development of the viewers' internal feature model as they have to allow for the variations in image quality. However, deconvolution is notorious for producing image artefacts if not sufficiently regularized (see, for example, comments at the end of section 1 of Magain et al., 1998): such artefacts could create more problems than the deconvolution solves, reducing the probability of a discovery being made. An experimental test of the sensitivity with which an image set enables discovery is required.

In this paper we investigate a simple model-based deconvolution scheme for resolution matching, combined with a standardised image scaling and stretch, producing homogenized sets of color composites for inspection and then testing their sensitivity for lensed feature detection by viewers in the Galaxy Zoo. Using a set of N realistic simulated test images for the XXX survey containing faint lensed features, we define a set of metrics that quantify discovery sensitivity based on a test group of viewers responses, and ask the following questions:

- Does simple, "light" deconvolution designed for making resolution-matched composite images introduce significant colored artefacts?
- What target PSF-width should be used in order to maximise discovery sensitivity in the color composite images? How does this relate to the input images' resolution?
- What algorithm for choosing the color composite images' stretch and scaling should be used, to to maximise discovery sensitivity?
- Is there significant scatter in viewers' preferences regarding image stretching and scaling, such

contributions to source-discovery.

that viewer control over these parameters is justified?

While we will focus on lensed features as our discovery targets, and the CFHT legacy survey, we hope that our results will be of interest to the astronomical community in general, and in particular to anyone who wants to see what they have found in the object catalog database of a large synoptic imaging survey.

In the following two subsections, we discuss the origins of blurriness in telescope images (1.1) and highlight some issues regarding the combination of band-pass data (1.2). Then in section 2 we outline a scheme for mitigating image bluriness. In section 3 we change tack and discuss a separate issue – that of combining filtered images into coloured composites. These two strands are brought together in section 4.

1.1 The Point-Spread Function (PSF)

An image of the sky taken with a telescope will never perfectly capture the information in the afferent light. One of the main problems is blurring – precise points in the sky become fuzzy, extended objects when measured with any conceivable apparatus. This happens for two reasons: (i) the finite size of camera apertures or telescope mirrors spells trouble for geometrical optics; the resulting diffraction blurs the image formed at the detector. (ii) light wavefronts passing through the atmosphere to the surface of the Earth are inevitably distorted by density fluctuations, which affect the refractive index of the air. This is particularly tricky to correct for because the atmosphere changes in unpredictable ways, on short spatial and temporal scales. The problem can be circumvented to some extent – a space telescope avoids corrupting its data with influence from the messy atmosphere. But putting a telescope in space is expensive and compromises the flexibility of the mission's science goals. Most of the time there is no alternative but to build a ground-based telescope and try to correct the blurriness.

We can talk about "blurriness" in a more quantitive way. An astronomical scene will typically feature a number of point sources (stars) which are bright but should be too small and distant to resolve into more than one pixel. How these point sources map to fuzzy blobs in our recorded image is known as the "point spread function" (hereafter PSF). As noted above, the PSF for a given image will have some contribution from the recording apparatus, and a more complicated time- and space-dependent contribution from the atmosphere.

So each pixel of the scene is replaced by a suitably weighted image of the PSF in the observed image: mathematically, this procedure is a convolution (see figure 1). In order to make precise inferences about the configuration or properties of astronomical objects, it is of utmost importance to find the PSF for a given image (or part of an image) and try to subtract it out as much as possible ("de-convolve" in the astronomy jargon). A framework for estimating the PSF is given in section 2.1, and the de-convolution process is described in the sections following.

1.2 Bandpasses and Colour Images

As mentioned above, astronomical images are often filtered so that only a predetermined broad set of wavelengths will reach the detector. We then make multiple images of the same objects

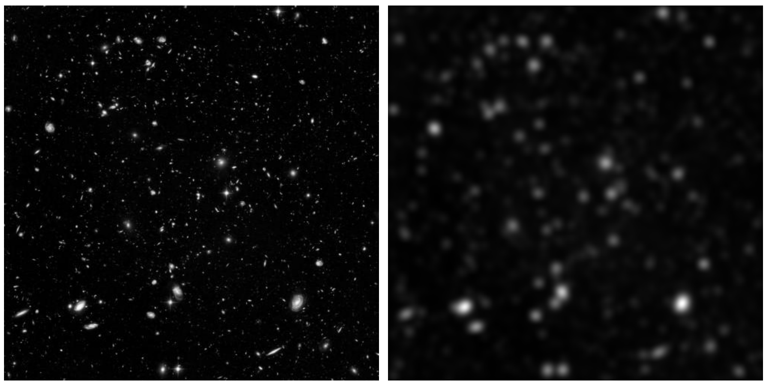


Figure 1: The effect of the PSF: blurring and loss of distinctness.

with different filters in place. This may seem strange: each filter throws away photons, making the resulting image less intense than an unfiltered counterpart. When we're trying to detect faint objects, this is not a good strategy. Furthermore, instead of taking just one picture of an object, we're now taking three or more (depending on the number of filters) – this constitutes a considerable time burden on the telescope.

However, for many applications in the optical or near-optical, the sources of interest have ample brightness for us to use the filters. And the information we gain from having wavelength-dependent flux measurements is well worth the extra time required. With the colour-domain data, we can make all sorts of inferences about the age, activity, distance, chemical composition and temperature of an astronomical source. And these "photometric" images are much cheaper to collect (in terms of time and photon counts) than spectra of each individual object.

Once we have our three (say) images of the same patch of sky, with each representing a different colour, we may ask whether we can add them together recover a colour image as shown in figure 2. This seems reasonable, but in fact it can be very problematic. The three images have been made at different times; from section 1.1, we know that they will have been corrupted by different PSFs; they may even have markedly different noise levels; indeed, atmospheric or device conditions may have resulted in spuriously different baseline intensities. As we can see from figure 3, we have to do something cleverer than simply adding three such images to make a nice-looking colour composite. This will be discussed further in section 3

1.3 The Data

So the aim of the project is to start addressing these problems – de-blurring images and combining them in such a way as to capitalise on their information content and emphasise faint features which may be of interest. I test my code on a sample of *irg* images from the CFHT Legacy Survey (CFHTLS). The hope is that the code will eventually be useful for large sets of images in the context described above.

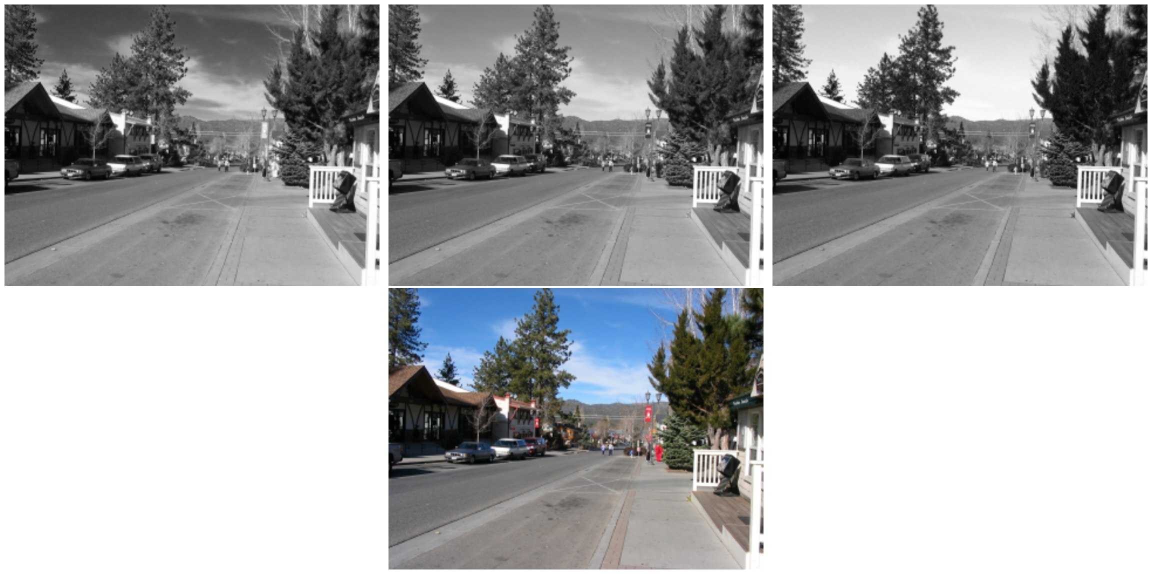


Figure 2: Starting with three greyscale images (our R, G and B channels), we can create a colour composite image.

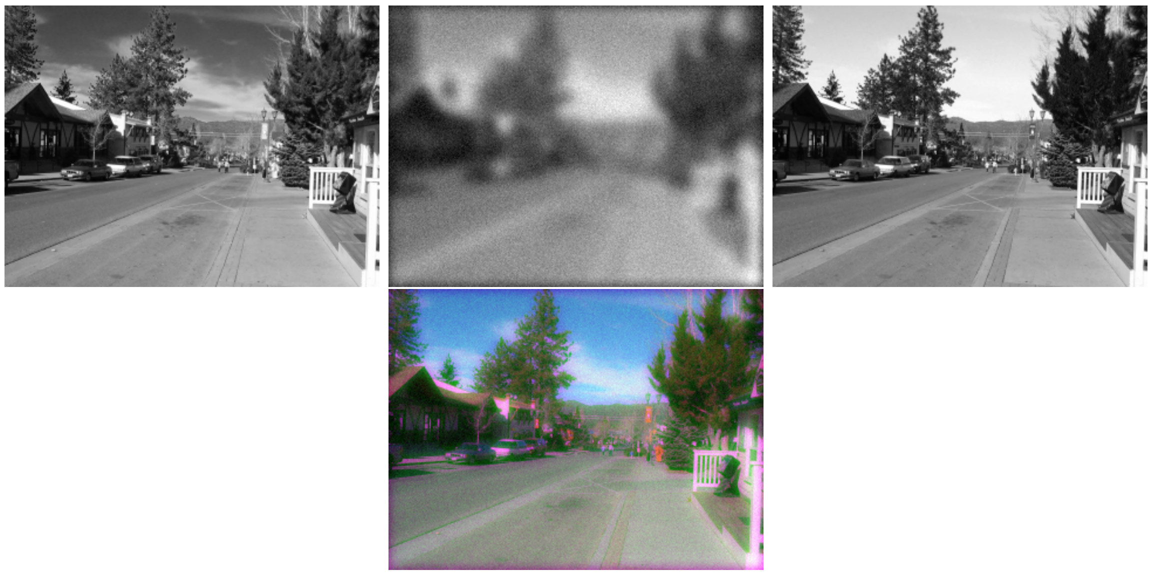


Figure 3: If the images to be combined are of differing quality (the green band now has more noise and larger PSF), adding them as we did before will result in an obviously distorted image. (Ignore the borders of the composite – there is an image processing artefact there.)

2 Deconvolution

As mentioned in the introduction, it is frequently possible and indeed desirable to reduce or control the bluriness (or PSF) of a telescope image. In this section, we describe how one might do this – the procedure is as follows:

1. Identify the astronomical sources in a FITS image.
2. Find the point sources, and from these estimate the PSF of the image. We call this the "observed PSF", or $\mathrm{PSF}_{\mathrm{obs}}$.
3. Generate a symmetrical[5] target PSF for the image, using the dimensions and flux properties of the image PSF from point 2. We call this the "reference PSF" or $\mathrm{PSF}_{\mathrm{ref}}$.
4. Create an object which maps the observed PSF to the reference PSF. This object is called the "kernel" and the mapping procedure is convolution.
5. Apply this procedure to the original image; this should correlate all the pixels in such a way as to rid the image of asymmetrical blurriness and replace it with a blurriness of known properties.
6. Finally, regularise the image by using some smoothing procedure.**MORE**

The reader may be confused at this point as to why, if we can correctly calculate the PSF of an image, we don't simply do away with the blurriness altogether – i.e. find a kernel which maps the PSF to a point. This would be a "hard deconvolution", a procedure which is compellingly discouraged by the work of Magain et al. (1998) (see appendix A for more discussion of this paper). Following this work, we may endeavour to obtain better knowledge of the sky by reducing the PSF, but we must avoid inadvertently violating the sampling theorem, which would certainly happen if we attempted a hard deconvolution.

Once we have made the target PSF as small as possible within this restriction, we can enforce that it be symmetrical uniform throughout the whole image: this is a "soft deconvolution". The result of this will be that all point sources have the same pre-determined shape, and extended sources will be superpositions of this shape.

Mathematically, we can think of $\mathrm{PSF}_{\mathrm{obs}}$ as being composed of the reference PSF convolved with a messy 2D function K:

$$\mathrm{PSF}_{\mathrm{obs}}(\vec{x}) = \mathrm{PSF}_{\mathrm{ref}}(\vec{x}) * K(\vec{x}). \tag{1}$$

In the following sections, we discuss how we determine the observed PSF, construct the reference PSF, and find the convolution kernel K. Crucially for homogenising the image, we must also find the kernel k which governs the inverse transformation, i.e.

$$\mathrm{PSF}_{\mathrm{obs}}(\vec{x}) * k(\vec{x}) = \mathrm{PSF}_{\mathrm{ref}}(\vec{x}). \tag{2}$$

[5] Well, not actually circularly symmetric.

2.1 Source Identification

According to the list in section 2, the first step in deconvolving an image is to pinpoint and characterise all the sources in it. We do this using the SExtractor ("source-extractor") software written by E. Beritn and hosted on the Astromatic website. SExtractor reads in a FITS image and catalogues all the astronomical sources it contains, for each one measuring useful parameters such as position, characteristics of shape and flux distributions. The identification of the distinct sources ("segmentation") is based on a thresholding procedure, which compares regions of the image to the local background; this process is then repeated so that proximate objects can be deblended.[6] For more detail, see the user manual (Bertin and Arnouts, 1996), which can be downoaded from the Astromatic website `astromatic.net`.

All the user really has to do is to decide how the program should be set up, and select – from an impressive list – which quantities should be calculated. I store the default settings in the config files `prepsfex.sex` and `prepsex.param` respectively. The output of the program is 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, which is to be used by the next piece of software to estimate the PSF in the image.

2.2 Determining the PSF of an Image

Another piece of Astromatic software, PSFEx, is used here. From the FITS image and the information in the SExtractor's `.cat` file, PSFEx identifies which objects are actually blurred point sources, and uses their shape to determine the image PSF. The selection criteria for finding the point soures is that they lie in a particular region on a plot of pixel-width versus object intensity. Inside this region, we can confidently say the oject is a star, while inside, the object could be a galaxy or a cosmic ray or noise or anything. More detail is given in the PSFEx manual (Bertin, 2011) – see in particular figure 3 and section 6.2.1.

Again, 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. For now at least, we're considering small images where the variation of the PSF over the field is small. We may have to do something rather more clever if we have to work with larger pictures in the future.

2.3 Generating a Target PSF

We are now equipped with an image which estimates the PSF of our data, PSF_{obs}. We now have to make a decision about what our reference PSF should look like.

In a (misguided) bid to keep things simple, I prescribed "PSF-tidying" rather than a traditional light deconvolution. This means that PSF_{ref} has the same size as PSF_{obs}, but a different shape – specifically, a 2D Gaussian. The procedure was:

[6]There is some discussion of replacing the threshold algorithm with wavelet decomposition in future editions of the software.

1. Find the centre and the two principal widths (variances) of the $\mathrm{PSF}_{\mathrm{obs}}$ image.
2. Use this information to make a 2D Gaussian with the same position and widths.
3. Save as a FITS file.

Unfortunately, after many hours spent on making code compatible with this programme, a belated epiphany on the actual meaning of the Sampling Theorem meant that most of it had to be abandoned in favour of a much simpler procedure: viz., the target PSF should in all cases have a FWHM of two pixels.

My mistake was in applying the Sampling Theorem to the resolution of the data rather than the resolution of the detector. In retrospect, it's fairly obvious that this is wrong: the sampling increment should be twice as fine as the smallest feature in a signal. For an astonomical image, the sampling increment is a pixel and the smallest feature is the PSF – so we should arrange it such that the PSF has a width of about two pixels.

2.4 Mapping from the Target PSF to the Observed PSF

In this section, we outline how to find the "convolution kernel" which maps between the two versions of the PSF, discussed above. This object allows us to regularise the entire image.

Say we have two similar images: $\mathcal{A}$, which is a picture taken by a real telescope; and $\mathcal{B}$, which represents the same astronomical image, with a smaller, user-specified PSF ($\mathrm{PSF}_{\mathrm{ref}}(\vec{x})$). In order to change image $\mathcal{B}$ into image $\mathcal{A}$, we can convolve it by some kernel k:

$$\mathcal{A} = \mathcal{B} * k. \tag{3}$$

Note that k is merely an image, or a template for how each pixel in $\mathcal{A}$ is a sum of pixels in $\mathcal{B}$.

Since convolution is a linear operation, we can cast our search for the convolution kernel in terms of a linear algebra problem. Specifically, we want to solve (for $\vec{k}$)[7] an equation of the form

$$\vec{a} = \mathbf{B}\vec{k}, \tag{4}$$

where the vector $\vec{a}$ and the matrix $\mathbf{B}$ encode the original and deconvolved image, and $\vec{k}$ represents the convolution kernel (also an image). When written in this form, our problem of finding $\vec{k}$ becomes a traditional vector-equation–solve.

The vectors $\vec{a}$ and $\vec{k}$ are easy to construct – rows of the 2D image are concatenated into a long 1D array (this is called "flattening"). If $\mathcal{A}$ is an image of N pixels, then $\vec{a}$ will have N elements. The kernel $\vec{k}$ has many fewer elements, but the procedure is exactly the same.

The target-PSF image matrix, $\mathbf{B}$ is less strightforward. We must think carefully about the convolution process to understand what's going on here. Consider a uniform 3×3 kernel being convolved with a 5×5 image, as shown in figure 4. The form of the pixels suggests that it may be represented as the linear algebra operation shown in figure 5.

[7] Here, and throughout the document, we abuse the vector/matrix notation. Vectors are traditionally defined by their transformation properties; here we just take them to be carfully-constructed lists of real numbers.

$$\begin{array}{|c|c|c|c|c|}\hline 1 & 2 & 3 & 4 & 5 \\ \hline 6 & 7 & 8 & 9 & 10 \\ \hline 11 & 12 & 13 & 14 & 15 \\ \hline 16 & 17 & 18 & 19 & 20 \\ \hline 21 & 22 & 23 & 24 & 25 \\ \hline \end{array} \ast \begin{array}{|c|c|c|}\hline \mathrm{A} & \mathrm{B} & \mathrm{C} \\ \hline \mathrm{D} & \mathrm{E} & \mathrm{F} \\ \hline \mathrm{G} & \mathrm{H} & \mathrm{I} \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline \begin{array}{c}\mathrm{1A+2B+3C+} \\ \mathrm{+6D+7E+8F+} \\ \mathrm{+11G+12H+13I}\end{array} & \dots & \dots \\ \hline \dots & \dots & \dots \\ \hline \dots & \dots & \dots \\ \hline \end{array}$$

Figure 4: A five-by-five image (left) with pixels numbered from one to 25, is convolved with a three-by-three kernel (right) with pixels labelled from A to I. Notice the final image is smaller than the original – more on this in section 2.4.2.

$$\begin{pmatrix} 1 & 2 & 3 & 6 & 7 & 8 & 11 & 12 & 13 \\ 2 & 3 & 4 & 7 & 8 & 9 & 12 & 13 & 14 \\ 3 & 4 & 5 & 8 & 9 & 10 & 13 & 14 & 15 \\ & & & & \vdots & & & & \end{pmatrix} \begin{pmatrix} A \\ B \\ C \\ D \\ E \\ \vdots \end{pmatrix} = \begin{pmatrix} 1A + 2B + 3C + \\ +6D + 7E + 8F + \\ +11G + 12H + 13I \\ \vdots \end{pmatrix}$$

Figure 5: The convolution procedure of figure 4 is translated into a matrix operation.

The **B** matrix of equation 4 is then a matrix with size(a) rows and size(k) columns. Moreover, it has a skew-diagonal Toeplitz structure, which will help us to constuct it later.

Presently, the vector problem in equation 4 is overdetermined (the kernel is smaller than $\mathrm{PSF}_{\mathrm{obs}}$), so we must use some “best fit” procedure – typically least-squares minimisation – to find $\vec{k}$.

2.4.1 Pseudocode

```
 1. psf_obs = readin(psfex_psf.fits)

 2. psfmoments = moments(psf_obs)

 3. psf_ref = 2DGaussian(moments)

 4. ## Turn these image arrays into linear algebra objects

 5. psf_obs = psf_obs.flatten()

 6. psf_ref = stack(psf_ref, dim=(psf_obs.size, kernel.size))
          ## stack() constructs rectangular matrix of dimensions dim
          ## conformable for multiplication with the flattened kernel image.

 7. kernel = solve(psf_ref, psf_obs)

 8. kernel = reshape(kernel, kernel_dim)

 9. save_image(kernel)

10. return kernel
```

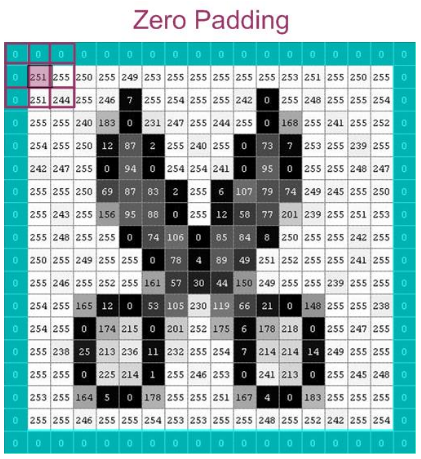
2.4.2 Interlude: Honesty Note

When we convolve two images, we produce a new image pixel by taking a weighted sum of the surrounding pixels. For instance, say we convolve a large image $\mathcal{B}$ with a 3×3-pixel image to get $\mathcal{A}$. The first (i.e. top-left) pixel of $\mathcal{A}$ that we can properly determine is not the same as the first pixel of $\mathcal{B}$, because we require input from the eight pixels around every $\mathcal{B}$ pixel in order to get an $\mathcal{A}$ pixel.

Thus, the honest thing to do is to throw away some information by making image $\mathcal{A}$ smaller than $\mathcal{B}$ – in this example with a 3×3 image, we shave off the four outer edges, so if $\mathcal{B}$ is $N \times M$, $\mathcal{A}$ is $(N-2) \times (M-2)$. A bigger convolving image would require more pixels to be lost from the result.

This contrasts with the traditional solution to the problem, which is to "pad" the original image with zeroes (see figure **??**) such that the final image is the same size as the (pre-padded) original was. This seems to introduce spurious information – how can we possibly know that there are zeros at the border of an image? Often this is patently not the case, even for astronomical images which can be mostly black. With this in mind, it may seem preferable to adopt the "lossy" procedure outlined in the last paragraph.

DWH has argued semi-convincingly that we needn't make such a sacrifice in practice, because there is enough information in the border pixels to mitigate grave data-fabrication. See also the sections below, where this issue is discussed further in the context of a well-posed linear algebra problem.


Zero Padding

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	251	255	250	255	249	253	255	255	255	255	255	253	251	255	250	255	0
0	251	244	255	246	7	255	254	255	255	242	0	255	248	255	255	254	0
0	255	255	240	183	0	231	247	255	244	255	0	168	255	241	255	252	0
0	254	255	250	12	87	2	255	240	255	0	73	7	253	255	239	255	0
0	242	247	255	0	94	0	254	254	241	0	95	0	255	255	248	247	0
0	255	255	250	69	87	83	2	255	6	107	79	74	249	245	255	250	0
0	255	243	255	156	95	88	0	255	12	58	77	201	239	255	251	253	0
0	255	248	255	255	0	74	106	0	85	84	8	250	255	255	242	255	0
0	250	255	249	255	255	0	78	4	89	49	251	252	255	255	241	255	0
0	255	246	255	252	255	161	57	30	44	150	249	255	255	239	255	255	0
0	254	255	165	12	0	53	105	230	119	66	21	0	148	255	255	238	0
0	254	255	0	174	215	0	201	252	175	6	178	218	0	255	247	255	0
0	255	238	25	213	236	11	232	255	254	7	214	214	14	249	255	255	0
0	255	255	0	225	214	1	255	246	253	0	241	213	0	255	245	248	0
0	253	255	164	5	0	178	255	255	251	167	4	0	183	255	255	255	0
0	255	255	246	255	255	254	253	253	255	255	248	255	252	242	255	254	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6: An image padded with zeroes. Convolution with the 3×3 stencil will now preserve image size; but at what cost?

2.5 Full Image Deconvolution

In section 2.4 we found the kernel k which would transform a PSF with known properties ($\mathrm{PSF}_{\mathrm{ref}}$) into the observed PSF ($\mathrm{PSF}_{\mathrm{obs}}$). This same kernel can be used to transform an entire image with the observed PSF, $\mathcal{A}$, into the "same" image with the target PSF, $\mathcal{B}$. To do this, we can perform the operation

$$\mathcal{A} = k * \mathcal{B}, \tag{5}$$

or, in the language of linear algebra,

$$\vec{a} = \mathbf{K}\vec{b}. \tag{6}$$

The image vectors $\vec{a}$ and $\vec{b}$ have the same form as discussed earlier – they are simply flattened versions of the pixel-arrays. The kernel matrix $\mathbf{K}$ is more tricky; but after some refelxion, it transpires that $\mathbf{K}$ is a sparse, upper-diagonal matrix where every row is identical, but shifted right by one element with respect to the row above. (Note that in practice it is unweildy and prohibitively expensive to store all the zero-entries of the $\mathbf{K}$ matrix – we have to be a little bit clever about how to store and manipulate the relevant information.)

Once we have constructed an appropriate kernel matrix, we solve (using `scipy.sparse.linalg`) the linear system to find the pixel vector $\vec{b}$ for the target image $\mathcal{B}$.

Solving an overconstrained problem. If we want to solve the matrix equation 6 and get an unambiguous answer, we need the system to be *overdetermined* – i.e. the inferred scene image-vector $\vec{b}$ should have fewer elements than the data image-vector $\vec{a}$. This goes right against the arguments raised in section 2.4.2, which considers information-transduction in the convolution procedure.

How to resolve the conflict? Notice that in violating the section 2.4.2 mandate, we will introduce problems on the inferred image's boundaries[8]. We can live with this – especially if we are considering large images which have comparatively little information on the boundary. On the other hand, solving an underdetermined system is inherently unstable. So in order to arrive at a robust solution for $\vec{b}$, we should sacrifice our faithfullness to the convolution procedure.

Now there are two possibilities: (i) "pad" the data image $\mathcal{A}$ with zeroes; this will allow us to maintain the size of the inferred image. (ii) Keep $\mathcal{A}$ the same and accept a smaller inferred image $\mathcal{B}$. (In both cases, the kernel matrix $\mathbf{K}$ is taller than it is wide.) The choice isn't particularly compelling – in (i) we are introducing spurious information but this is confined to the edges; in (ii) we maintain the purity of our method, but may introduce problems later on when we try to compare the images. Uncharacteristically, I'll opt for (i), the simpler-but-corrupt choice.

We should choose the padding so that the deconvolved image has the same size as the original data image. This means making a border of black pixels with total width ahnd height equal to the width and height of the kernel image. The changes to how we construct our kernel matrix are less trivial, and require careful thought and much picture-drawing (often it is hard to visualise how image manipulations translate into linear algebra operations). The kernel matrix corresponding to a 2D Gaussian kernel is shown in figure 7.

The kernel matrix has some conspicuous gaps. For the unpadded case, this comed from the kernel overlapping the image boundaries. For the padded case, we also need to factor in the fact that there are many extra zeroes for which we must acconut. It should be clear from the shapes of the images in figure 7 that the unpadded matrix is underconstrained and the padded matrix is overconstrained.

[8]The problem will rach a full kernel-width into the recovered image, though the severity diminishes the further in you go.

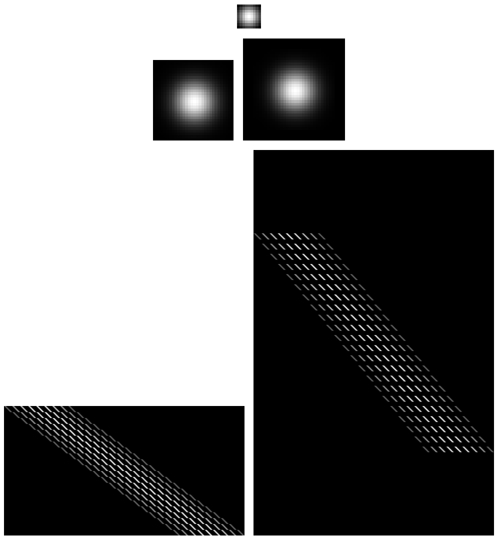


Figure 7: Kernel matrix from the kernel image. Top: the convolution kernel. Middle row: the data (left) and the data padded all around with zeroes. Bottom: the matrix constructed from the kernel image in the case of the unpadded data (left) and the padded data (right).

2.5.1 Pseudocode

This should illustrate how the general structure of the deconvolution procedure.

```
1. img_array = readin(imagefile)
         ## Or can be given as straight array

2. img_vec = img_array.flatten()

3. krn_array = readin(kernelfile)
         ## Or can be given as straight array

4. krn_mat = sparse_diag_matrix(krn_array.flatten(), offsets=[0,1,2,...])
         ## general gist

5. refimg_vec = sparse_lsq_solve(krn_mat, img_vec)
         ## System is underdetermined → iterative solution

6. refimg_vec = stretch(refimg_vec)
         ## Ease of viewing

7. refimg = reshape(refimg_vec)
         ## Make into an image-worthy shape

8. save_image(refimg)
```

2.5.2 Incorporating an Initial Condition

In practice, we can do better than this, since we have some prior information on what the deconvolved image should look like: viz. the original image. Using this guess could dramatically reduce the number of iterations required for convergence. This is how we do it.

We wish to solve an equation of the form

$$\vec{a} = \mathbf{K}\vec{b} \tag{7}$$

for $\vec{b}$. We have an initial guess, $\vec{b_0}$; from this we compute a residual vector,

$$\vec{r_0} = \vec{a} - \mathbf{K}\vec{b_0}. \tag{8}$$

If our initial guess is good, the elements of this vector should be small. Now we can find the correction to $\vec{b_0}$ by solving the matrix equation

$$\mathbf{K}\vec{\Delta b} = \vec{r_0}, \tag{9}$$

using least-squares or something similar. Then our final estimate for the solution is

$$\vec{b} = \vec{b_0} + \vec{\Delta b}. \tag{10}$$

This should be more accurate and less computationally intensive than doing it without the initial condition.

For our purposes, the initial guess will just be the observed image; or in the language used above, $\vec{b_0} = \vec{a}$. However, we can't simply plough ahead with this, because the original image and the target image will have different dimensions ($\mathbf{A}$ is not square). This is easily rectified by augmenting the original image vector $\vec{a}$ with zeroes and using this padded version as $\vec{b_0}$; thankfully there is no "dishonesty" here (see section 2.4.2), since $\vec{b_0}$ is only a guess in the first place.

We still have to be a little careful about how we achieve this padding: we shouldn't put zeros in the wrong places. If the original image was a square of side n pixels, and the kernel image a square of side m pixels, then the initial guess image should have side $n+m-1$ pixels. The vector $\vec{b_0}$ must therefore be padded with $(n+m-1)^2 - n^2$ zeroes, distributed equally at the beginning and end of each stride. This could be achieved by doing the following:

```
img_vec_padded = img_vec.append(zeros(floor(m/2)), [[i*stride, -i*stride]
                                              for i in range(img_vec.height)]),
```

or by embedding the original image into a larger array (this is what I actually do).

With all this in mind, it seems we modify our algorithm above:

```
1. img_array = readin(imagefile)
        ## Or can be given as straight array

2. img_vec = img_array.flatten()

3. krn_array = readin(kernelfile)
        ## Or can be given as straight array
```

```
4. krn_mat = toeplitz_matrix(krn_array.flatten())
        ## general gist
5. krn_mat = sparse_diag(krn_mat)
6. Dim = sparse_lsq_solve(krn_mat, img_vec - krn_mat*img_vec_padded)
        ## Find correction from residual
7. refimg_vec = img_vec + Dim
8. refimg_vec = stretch(refimg_vec)
        ## Ease of viewing
9. refimg = reshape(refimg_vec)
        ## Make into an image-worthy shape
10. save_image(refimg)
```

Having said all this, it is unclear how much benefit we actually get from including our prior information. The least-squares algorithm is convex, and has been well-optimised so it is also very fast. Should test this, but it's not central to the project so maybe some other time.

2.5.3 Testing

Now This is what I'm doing at the moment to test my deconvolution code (both with and without the initial condition).

```
1. $PSF_{obs}$ = Gaussian(bigwidth)
2. $PSF_{ref}$ = Gaussian(smallwidth)
3. kernel = get_kernel($PSF_{ref} \rightarrow PSF_{ref}$)
4. decon = deconvolve($PSF_{obs}$, kernel)
5. saveimage(decon)
```

This finds the convolution kernel which takes a small Gaussian to a large Gaussian. Then it finds the image, `decon`, which must be convolved with that kernel to give the large Gaussian. `decon` *should* be the small Gaussian, right?

So far this has not been the case; see figure 8, which shows the original image, the target PSF, the deconvolved image and the kernel (calculated according to the prescription above). The deconvolved image looks exactly like the original image, not the reference PSF as hoped.

Other than the results being wrong, here are some other things I've noticed:

- For Gaussian target and data, the kernel does not come out to be a Gaussian (8). What's going on?
- It takes a long time to solve for the image. Even something of 100x100 pixels takes ten minutes on my computer.

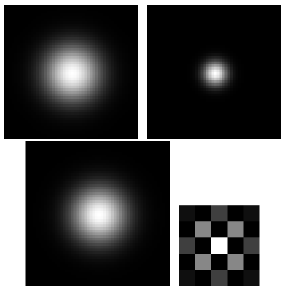


Figure 8: original image, target PSF, deconvolved image and the kernel

- It takes longer to solve the sparse system using Scipy's sparse linalg module than using Numpy's straight linalg module. Weird.
  - (`linalg.lstsq` uses householder bidiagonalization to decompose. For an m by n matrix, the complexity will be $\mathcal{O}(\max(m,n) * \min(m,n)^2)$.)
- My "initial guess" procedure doesn't seem to speed up convergence at all – even when I give the exact solution as my initial guess! Is there some large overhead for using least-squares?
  - Hogg says that the initial guess shouldn't help much because least squares is so fast. That doesn't explain why it still takes a long time when given exact solution.

Constructing the kernel matrix K from the kernel image. Figure 9 shows an image of the kernel matrix, constructed from the image in figure 8, to be used in equation 6 for inferring the reference image.

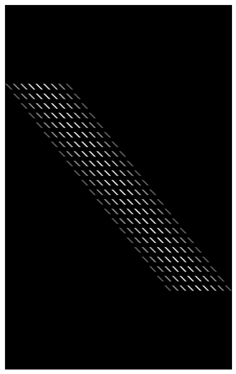


Figure 9: Kernel matrix used in linear algebra.

The form of this matrix is governed by the inclusion of padded black-space in the observed image.

Deconvolving a larger image. Once this works, I'll apply the same procedure to a larger image with several objects of size $\mathrm{PSF}_{\mathrm{obs}}$; e.g. figure 10.

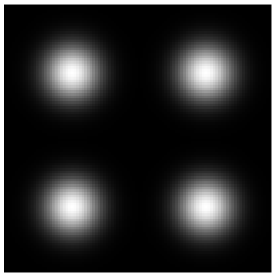


Figure 10: Bigger image to deconvolve

Speeding it up. It may be possible to do the linalg-solve without computing the kernel matrix and performing numerous matrix products. The syntax is something like `solve(function, b)`, where `function` would be the operation I use to get the kernel matrix. Investigate this.

2.5.4 hoggtest

A good test problem to pose is one where we know the answer in advance. An example for us to try could be a system where each image (data, kernel and scene) is a 2D Gaussian of some width. This relies on the (remarkable) property that the convolution of two Gaussians is another Gaussian:

$$\mathcal{G}(r, \sigma_1^2) * \mathcal{G}(r, \sigma_2^2) = \mathcal{G}(r, \sigma_1^2 + \sigma_2^2). \tag{11}$$

To be totally explicit: if we pick our scene image to be a Gaussian with width σ_1 and our kernel to be a Gaussian with width σ_2, the data image will be a third Gaussian with the quadrature-sum width.

The results are in figure 11. Simple code is on page 19.

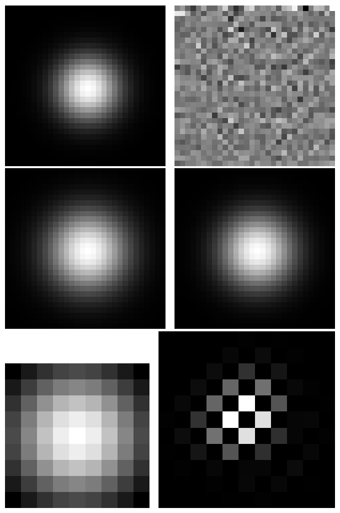


Figure 11: Scene, data and kernel (scaled 2x,2x,6x). On left is the "correct" image (2D Gaussian). On the right is output of program. Note each image has different intesity scale which is unhelpful.

Note the obvious problem (identified earlier) that the kernel looks funny: clearly not a Gaussian. This is strange, because the kernel calculation is a highly over-constrained linear algebra problem, so there should be no ambiguity from that end.

It turns out that for a size of 9x9 pixels, the kernel image looks pretty plausible. This, along with a number of other simple tests, lead me to believe that the code was working properly. But if we select a larger kernel size, say 15x15, the image looks like a mess and not at all what we expect.

FIGURE: kernel image 9x9, 15x15

```
def hoggtest():

t0 = time.time()

## Define true scene, kernel, and data
## Arguments are image size, width of Gaussian (x,y), centre (x,y), total flux
scene_t  = DT.Gauss_2D(*[30, 4., 4., 15, 15, 1.])
kernel_t = DT.Gauss_2D(*[9, 3., 3., 4, 4, 1.])
data_t   = DT.Gauss_2D(*[30, 5., 5., 15, 15, 1.])
scipy.misc.imsave("hoggtest_scene_true.png", scene_t)
scipy.misc.imsave("hoggtest_kernel_true.png", kernel_t)
scipy.misc.imsave("hoggtest_data_true.png", data_t)

## Data: convolve the scene with the kernel
data = scipy.signal.fftconvolve(scene_t, kernel_t, "same")
scipy.misc.imsave("hoggtest_data.png", data)

## Kernel: data = scene * k --> kernel
kernel = DT.get_kernel(scene_t, data_t, [9,9], False)
scipy.misc.imsave("hoggtest_kernel.png",kernel)

## Scene: data = scene * k --> scene
scene = DT.deconvolve_image(data_t, kernel_t, False)
scipy.misc.imsave("hoggtest_scene.png",scene)

## Moments
print DT.moments(scene)
print DT.moments(kernel)
print DT.moments(data)

print "Total",round(time.time()-t0,3)

return
```

2.5.5 Shortcomings

That still isn't the end of the story, as equation 6 has some important shortcomings:

1. In using our recorded image to recover a "better" one (which bears more resmblance to the scene), we have ignored the existence of *noise* in the data. This problem will be addressed in section 2.6.

2. The procedure outlined in this section may not be well-determined, so the solution may be flexible. Typically, this will introduce artefacts into the image, which, given the goals outlined in the introduction, would be highly unwelcome.
   The reference image we have obtained at this point must be "regularised", or smoothed according to some well-motivated principles. One can find a good discussion of these principles in Magain et al. (1998): this paper is outlined in appendix A. Some comments on their mathematical procedure can be found in appendix B.
   Though it would be entirely possible to implement this for the current project (indeed, we have already generated much of the necessary data), it may well be more effort than necessary. A more modest regularisation scheme will eventually be proposed in section 2.7.

2.6 Pixel Variance and Noise

Here we outline how one may address the existence of noise in our original image – this has been hitherto ignored.

First, consider how the solution $\vec{y}$ to the matrix equation

$$\mathbf{M}\vec{y} = \vec{b} \tag{12}$$

is found. Typically we strive to minimise the sum of the squared-residuals (call this S) of the solution: that is, find successive values of $\vec{y_i}$ such that

$$S_i = (\mathbf{M}\vec{y_i} - \vec{b})^{\mathrm{T}}(\mathbf{M}\vec{y_i} - \vec{b}) \to 0, \tag{13}$$

where i labels the iteration number. Clearly, if we find a vector $\vec{y}$ which exactly solves equation 12, the square-residuals will sum to zero (moreover this solution is unique). In practice, we choose some small number ϵ such that once $S_i \leq \epsilon$, we accept the corresponding solution $\vec{y_i}$.

Now if our data are noisy, or worse if there are correlations between data points, we must accomodate this into our residual-minimisation procedure.

Say we know or can calculate the variance of each pixel, σ_j^2, and the covariance between pixels, σ_{jk} (j and k label the pixels). We express this information as a covariance matrix $\mathbf{C}$:

$$\mathbf{C} = \begin{pmatrix} \sigma_1^2 & \sigma_{12} & \sigma_{13} & \dots & \sigma_{1N} \\ \sigma_{21} & \sigma_2^2 & \sigma_{23} & \dots & \sigma_{2N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sigma_{N1} & \sigma_{N2} & \sigma_{N2} & \dots & \sigma_N^2 \end{pmatrix} \tag{14}$$

(note that $\mathbf{C}_{jk} = \mathbf{C}_{kj}$).

Including this information in the solution amounts to modifying the expression for the summed residuals, 13, in the following way:

$$S_i = (\mathbf{M}\vec{y_i} - \vec{b})^{\mathrm{T}}\mathbf{C}^{-1}(\mathbf{M}\vec{y_i} - \vec{b}). \tag{15}$$

A good trick for achieving this result without re-writing our minimisation code is to trivially modify equation 12:

$$\mathbf{M}\vec{y} = \vec{b} \quad \longrightarrow \quad \mathbf{C}^{-\frac{1}{2}}\mathbf{M}\vec{y} = \mathbf{C}^{-\frac{1}{2}}\vec{b}. \tag{16}$$

It is easy to show, using the symmetry of **C**, that using this matrix equation in 13 yields equation 15.

2.6.1 Pseudocode

We modify the procedure of section 2.5.1 to accomodate the covariance matrix (we only really modify part 5, but for completeness here is the whole thing):

```
1. img_array = readin(imagefile)
        ## Or can be given as straight array
2. img_vec = img_array.flatten()
3. krn_array = readin(kernelfile)
        ## Or can be given as straight array
4. krn_mat = sparse_diag_matrix(krn_array.flatten(), offsets=[0,1,2,...])
        ## general gist
5. Chalf = sqrt(inverse(covariance_matrix( img_array )))
6. refimg_vec = sparse_lsq_solve(Chalf * krn_mat, Chalf * img_vec)
        ## System is underdetermined → iterative solution
7. refimg_vec = stretch(refimg_vec)
        ## For ease of viewing
8. refimg = reshape(refimg_vec)
        ## Make into an image-worthy shape
9. save_image(refimg)
```

2.6.2 Noise Model

Having established that our minimisation functional, S, should include the data covariance matrix **C**, we must now discuss how to calculate this matrix.

The most trivial thing to do is what we have done thus far: neglect all the noise in the data. This corresponds to taking

$$\mathbf{C} = \mathbb{I}, \tag{17}$$

such that all **C**s in the above equations disappear.

If we actually want to take the noise into consideration, the easiest thing to do would be to assume (i) that each pixel represents an independent measurement (i.e. $\mathbf{C}_{jk} \propto \delta_{jk}$: there are no off-diagonal elements); and (ii) that each pixel in the image has the same uncertainty σ associated with it (this uncertainty σ is simply the root-variance of all the pixel values). Thus,

$$\mathbf{C} = \sigma^2 \mathbb{I}. \tag{18}$$

Increasing the sophisitcation of our model further, we could relax condition (ii) above and let different pixels have different noise levels[9]. This makes a lot of sense, since we'd expect brighter pixels to have higher noise. To get an idea of why this might be, consider photons from the sky arriving on a (perfect) telescope detector. This is a prototypical Poisson process, and in accordance with Poisson statistics, the mean count for a particular pixel will be equal to the variance in that pixel. In situations where there are many photons arriving at our detector (e.g. looking at bright optical sources), our Poisson distribution will become a Normal distribution; but the equality of mean and variance wil still hold. These considerations suggest the following covariance matrix:

$$\mathbf{C}_{jk} = \frac{n_j}{n}\sigma^2 \delta_{jk}, \tag{19}$$

where n_j is the number of counts in pixel j, n is the total number of counts in the entire image. Note that this matrix is still diagonal, and the overall noise properties of the image are preserved.

This is a pretty rudimentary model for the pixel uncertainty. A more methodical / unprejudiced procedure might be to use "feasible generalised least squares". But that's the general gist.

Now consider condition (i) above: is it possible that the noise in pixel j is correlated with the noise in pixel k? The answer is yes, for two reasons. First, a well-resolved source will be spread over several pixels. These pixels will have similar (high) noise levels when compared to the background, so neighbouring pixels have correlated noise. The second reason comes from considering the limitations of the detector. Take for example a CCD grid: very bright sources may saturate pixels if the exposure time is long. Charge, or photon counts, will overflow into neighbouring pixels, coupling the intensity and the noise.

A covariance matrix which refelcts these considerations will have off-diagonal elements which appear in diagonal stripes directly either side of the main diagonal (representing sideways-neighbours of pixel j) and other diagonal stripes around w elements from the main diagonal, where w is the width of the image. A calculation of how these off-diagonal stripes are related to the diagonal value j will depend to some extent on the intensity profile of the source which appears in j and on the properties of the detector.

Still more advanced noise models will exist. But for our treatment, we will content ourselves with a covariance matrix of the form given in equation 19.

2.6.3 Pseudocode

This pseudocode illustrates how we compute the covariance matrix in equation 19.

[9] I believe one would describe this variable noise model as "heteroscedastic".

```
1. img_array = readin(img_file).flatten()
2. flx_tot = sum(img_array)
3. flx_var = variance(img_array)
4. covmat = zero_array( shape = [img_array.size, img_array.size] )
5. covmat[i,i] = img_array[i] for i in img_array.size
6. covmat *= ( flx_var / flx_tot )
```

2.6.4 Precomputed Covariance Matrices

The necessity for locally calculating covariance matrices as in section 2.6.2 is sometimes avoided if the data come packaged with precomputed covariance matrix. Rather than inferring the image properties solely from the image itself, it is preferable to trust the knowledge of the people who produced the data, who are more familiar with the properties (and possible malfunctioning) of the instrument. In the case of faulty or missing pixels, they can also provide "masking arrays" which block out the relevant pixels so they aren't involved in any inference downstream.

2.7 Regularisation

After deconvolution, there will sometimes be image artefacts, or ambiguity as to the "best" solution (e.g. in the case of an underdetermined problem). We can reduce these using a regularisation technique, which typically involves minimising a cost function associated with the distirbution of intensity in the image. Choosing the cost function will be to some extent a matter of preference, since it encodes many of our assumptions about what the image ought to look like.

Magain et al. (1998) propose an effective regularisation scheme (see appendix A), but, as noted above, what they do may be overkill for our purposes. As of writing, this project hadn't reached the stage of incorporating a robust regularisation scheme; so this section is a placeholder for further work.

2.8 Image Stretch

A final comment about image viewability. The CFHTLS images with which we've been working have low photon counts, and the brightest sources often dominate to such an extent that other sources are invisible. To amend this, we've been applying a "stretch" to the image – a re-scaling of intensities such that faint objects are made brighter without saturating bright pixels. Much of the work that went into the next section was concerned with how best to do this, but for now we just use a simple linear stretch:

```
1. a = median(img)
2. b = precentile(0.975, img)
3. sigma = 0.5*(b-a)
```

```
4. scale_min = a-5.0*sigma
5. scale_max = a+10.0*sigma
6. img = (img-scale_min)/(scale_max-scale_min)
7. img.clip(scale_max-scale_min)
8. save(img)
```

Figure 12 shows the results of the procedure performed on two images. Notice that the stretch also brightens the sky-level, and enhances noise.

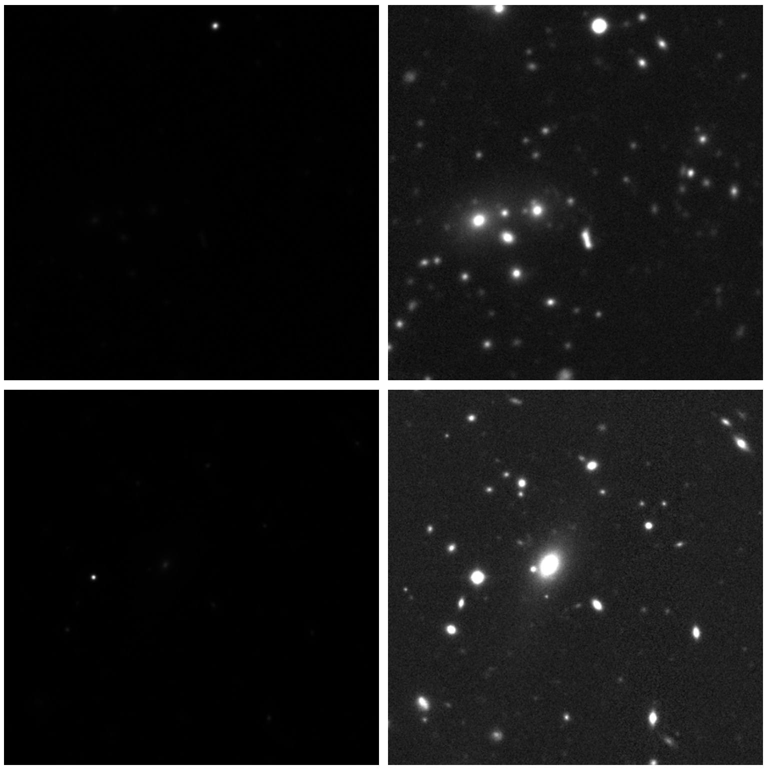


Figure 12: The effects of a simple linear stretch on the images CFHTLS_03_g.fits and CFHTLS_12_i.fits. Left is the raw data; the stretched version is on the right.

2.8.1 Noise Properties Post-Deconvolution

We use stretches to make images more readily interpretable, and we will perform such procedures on original and deconvolved images alike. Since the overarching framework of this project is about pattern recognition, it is necessary to analyse the effects of such manipulations on the noise properties of the image. Below we show stretches of increasing severity on an original image and its deconvolved counterpart.

TODO: Right column should be deconvolved image

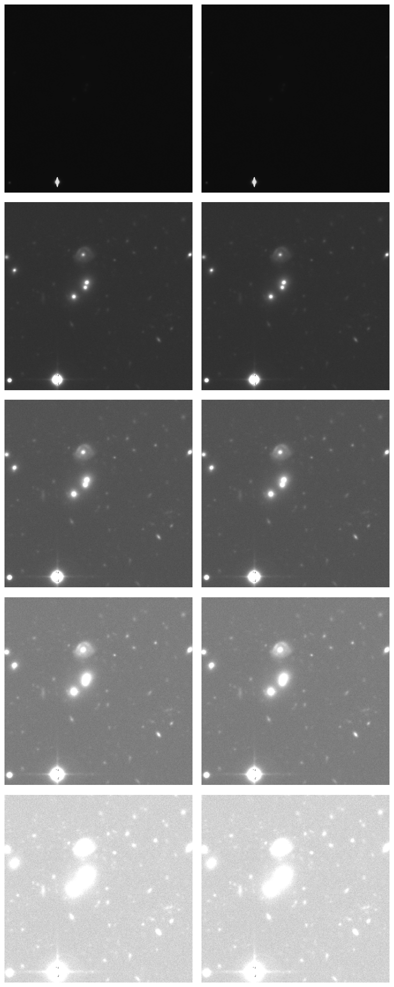


Figure 13: The effects of a linear stretch on the appearance of the image CFHTLS_06_r. The stretch gets more severe as you go down, starting with the unstretched image. The image in the left column is the original data, while the right column is the deconvolved image.

3 Bandpass Composition

Telescope data is often given in “bands” – we do not record the total flux coming from a point in the sky, but the flux in a certain wavelength range. This information allows us to investigate the properties of astronomical objects, such as temperature and chemical composition. However to make a readily-interprable image which contains maximum information in one hit, we need to combine the bands to make a colour-composite image. In all that follows, I use the mapping R=i, G=r, B=g.

Many methods had been developed to do this in accordance with researchers’ individual aesthetic preference. But the full richness of possibility was still unexploited, until Lupton et al. (2004) demonstrated just how much detail one could extract from data when colour was given its deserved treatment (see appendix C for more detail). Using their method, we can combine images from different bands in a way which enhances hidden or faint features without allowing bright objects to dominate. It has the further advantage that the brightness of an object in the image is decoupled from its colour. The method is implemented in PJM’s *HumVI* (formerly *ColorJPG*).

A further improvement to the method was implemented by Wherry. Whereas Lupton’s algorithm is appropriate for tweaking the look of a single image to bring out interesting features, Wherry lets us standardise this manipulation across many images so their properties are comparable. We translated Wherry’s improvements from IDL to Python, and integrated this module with the HumVI code. Some additional improvements are also made, such as increased versatility with rebinning.

3.1 Comparison of Composition / Stretch Procedures

The difference between the two procedures mentioned above (we’ll call them “lupton” and “wherry”) is subtle: the algorithms appear to be almost identical in how they combine and stretch the three input images. And though the methods appear to require different input parameters for the nonlinear stretch and colour scalings, this is deceptive.

Whereas lupton uses two explicit parameters, Q and α, to stretch an image (α is supposed to set the low-intensity scaling and Q controls the saturation behaviour), wherry only takes one user-selected parameter to achieve this. However, this apparent distinction is spurious because the lupton parameters always appear together in the code as the product αQ.

The other input is the list of three scalings, or weights, to be applied to the R, G and B channels of the image. Both methods give us control over these scalings – a fact most important for our purpose, because we want to compare many composites against each other: a pixel of certain colour and intesnity in our final image should be representing the same raw data, regardless of the rest of the image.

The code for the lupton procedure (originally written by Patrik Jonsson) appears to be more versatile with the treatment of these scales. Default behaviour is for the code to choose the best scales for the image, by ensuring some threshold proportion of pixels which are saturated in each band. Or we can demand that the scale is the same for all three channels, so that the overall colour is preserved. Or we can select the scales manually and feed them into the program. This last behviour is the default (and only) for the current wherry implementation. It seems to me that this degree control is desirable, but we should defer conclusion until we see enough images: how much work is it for the user to select their own scales, and does the automatic selection of the lupton code produce “natural”-looking images whose features are clear?

The most obvious distinction between the methods (rather than the implementation) is the inclusion in wherry of additional options on how to tweak the image: for example rebinning and modifying saturation levels. Beyond that, I can’t tell. Maybe there’s something subtle happening too.

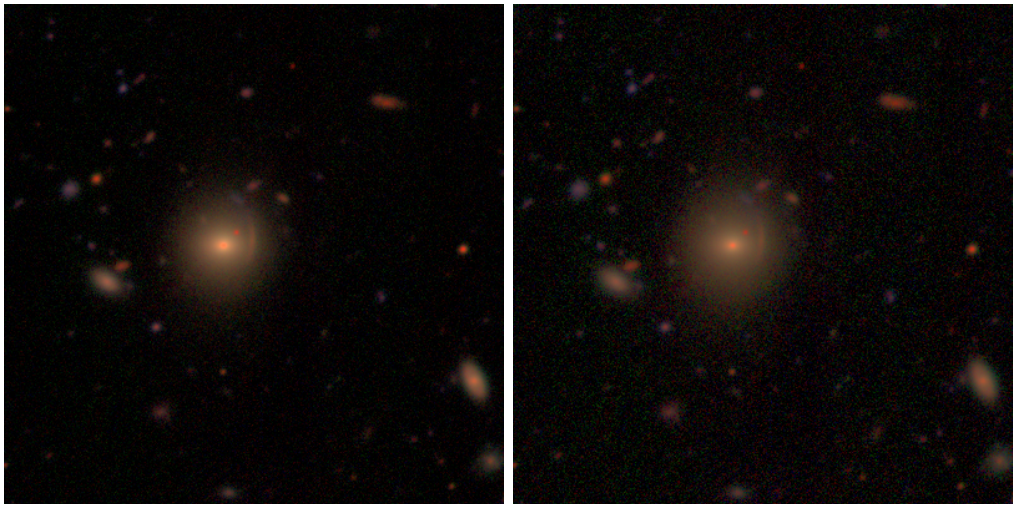


Figure 14: Comparing the results of combining the i, r and g bands of CFHTLS_27_sci using lupton (left) and wherry (right). The same nonlinearity parameters and colour scalings were used for both images.

Figure 14, we see that as expected the methods produce very similar results in terms of colouration and and intensity. However, the wherry image clearly has much higher noise levels, which sensibly degrade the image quality.

3.2 Pseudocode for “wherry” Composition

Here are the general steps in the code for a run of the wherry procedure.

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

2. rescale(RGB, scalefactors) ## multiply each band by a given number

3. rebin(RGB, xrebinfactor,yrebinfactor) ## re-sample images

4. kill_noise(RGB, cutoff) ## sets all pixels below a threshold to 0

5. arsinh_stretch(RGB, nonlin)

     • pixtot = R_array+G_array+B_array ## collapse images onto each other
```

- if pixtot[i,j]==0: pixtot[i,j]=1
- factor = arsinh(nonlin*pixtot)/(nonlin*pixtot)
- (R_array,G_array,B_array) *= factor

6. if stauratetowhite is 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)
7. overlay/underlay ## not entirely sure what these are for
8. scipy.misc.imsave(RGB)

4 Combined Deconvolution and Colour

One of the first questions we asked was how to combine images of different qualities and still make the most of the information in each one; how do we prevent the low-quality images from poisoning the combination? For instance, say we have i-, r-, and g-band images of a galaxy, but by chance atmospheric fluctuations the r-band is very blurred. If we simply add the images together pixel by pixel, we'll end up in a situation like figure 1, where the colours were distorted. Since colour is an important astronomical diagnostic, if this happens we'd be in trouble.

We now have the tools to begin solving this problem. Clearly we want to homogenise the blurriness of each of our R, G and B channels; that is make sure they have the same PSF. With these deconvolved raw imagess in hand, we can combine them à la section 3 to get a clean-looking result.

TODO: Examples? Don't have good data to do this? Can't deconvolve yet?

4.1 Finding Optimal Parameters for Image Manipulation

Recall that the overarching goal into which this project fits is to make certain features in astronomical imaged readily apprent to human veiwers. At some point, the free parameters of our procedure must be decided, in line with this aim; the best way to achieve this is to test the results of different stretches, scales and deconvolution properties on subjects who can rate the images on their clarity and the distinctness of fine structure.

PJM has this to say on the subject: 'Anupreeta More has made a very nice set of test CFHTLS images for the Lens Zoo, and I discussed with her a bit your work: she thinks we should do a blind taste test at some point, showing the lens zoo dev team her images that have been a) displayed in standard form with HumVI and b) deconvolved and then displayed in standard form with HumVI, and ask for them to be graded for arc visibility. We can also do image testing at Adler Planetarium on the willing public there. The question we want to answer is: "how should we display an image to maximize the likelihood of an untrained human seeing interesting feature X?" '

4.2 Quality Metrics

But before troubling astonomers or members of the public with such requests, we should first try to rate the processed data with some objective measures of their quality. I haven't yet explored any of this to the extent of implementation, but here are a few ideas which might be a good starting point.

- There will always be error in the minimisation step when we're solving the equation $\mathbf{A}\vec{x} = \vec{b}$. We can calculate the residual $\chi = \mathbf{A}\vec{x} - \vec{b}$, which will tell us how well our scene matches up with the data after convolution with the PSF kernel. The metric χ^2 will provide an unambiguous indication of the "goodness-of-fit" of our model to the data.
  - We take this further, by devising a protocol for splitting the image up into "meaningful" regions and comparing the χ^2 values in each one.

- As mentioned in section 2.8.1, we're also interested in determining the noise properties (e.g. distribution, level and spectrum) of the image; and whether this chances as we modify our deconvolution parameters or procedure. As mentioned in section 2.6.2, distinguishing noise from features of the scene can be a tricky thing to accomplish. There are some prescriptions in the literature (e.g. Louchet and Moisan (2008); Moisan (2007)), but not all of them are applicable to this field. More investigation is needed.
- We also want to keep track of image-processing artefacts, as these can masquarade as astronomical features. The archetypal artefact is ringing, where band-limited images pick up oscillations in the spatial domain. Again, this is an issue of wide concern, so there are many existing procedures for identification of ringing. All we have to do is pick one that we like and implement it. From some cursory research, I like the approach of Blanchet et al. (2010); but there may well be better methods out there.
    - I had an idea to turn the images into vector fields and calculate their curls and divergences, with the vague assumption that high-curl means more ringing. Of course, this is a highly-flawed approach because there are many good features which may produce high curl; but perhaps there's something in it.
- For our application, the distinctness of complex features is paramount. Some way of quantifying whether RGB image A of a galaxy has more fine detail then image B would be good.
- We get a lot of information from the colour differences between sources. Quantifying how the colour changes as we change the RGB scales may enable us to find some optimum scales which are slightly different from the automatically-calculated ones.

APPENDIX

A Magain et al. (1998) – “Deconvolution with Correct Sampling”

Key Ideas – We shouldn't pretend to derive infinite resolution images from discrete data. A more honest approach can mitigate the appearance of artefacts.

– Bear in mind that correlations in astrnomical images are local. Global treatments and techniques are inappropriate.

Background – Ground-based telescopes suffer from aperture diffraction and from atmospheric inhomogeneities which distort light. One (post hoc) way of correcting for this is to infer the point spread function from a puative point source in your image; if we consider the data to represent ”reality” or the ”scene” convolved with this PSF, then we can in principle deconvolve the data from the PSF to retrieve the scene.

– There will be many scenes compatible with the (uncertain) data, so we must then pose the problem as an optimisation problem: we wish to find the scene, compatible with the data, that minimises some cost function to be devised. A typical procedure is to minimise the chi-squared function (between data and model).

– Also want solution to be smooth, so introduce a Lagrange function which enforces this. A common procedure is to maximise the entropy of the image (using the flux distribution as the information). This has the benefit of requiring positive flux values.

– So far we ignore noise in the image.

Problems – Two problems emerge with this way of doing things: 1) often find image artefacts (from improper sampling, as we shall see); 2) it doesn't preserve the global intensity scale.

– In practice, telescope cameras are constructed so that their data just satisfy the sampling theorem – the pixel-spacing is 2x the maximum frequency expected from objects. Upon deconvolution, where the fuzziness is taken out, the sampling theorem will be violated. Theoretically, deconvolution can introduce point-sources/Dirac-deltas (i.e. stars), so an infinitely small sampling interval would have to be used.

– Deconvolution therefore leads to artefacts when there is a sharp discontinuity in the scene – e.g. a star on a black background shows ringing. (Can think of this as a window in frequency space (i.e. a cutoff at some maximum frequency) leading to a sinc function in position- space: the result of deconvolving a point source will be delta*sinc.)

– In traditional methods, riniging is mitigated by the positivity constraint, which damps down the lobes of oscillations. But this depends crucially on the zero-level, and accurate subtraction of sky noise is necessary for the methods to wrok well.

– Image artefacts steal flux and bias photometry. Also, maximising entropy makes the image as smooth (uniform) as possible, which tends to spread out point sources; peak intensity is thus undersetimated.

Proposal – Do not do a full deconvolution: do a "light" deconvolution where point sources are given as extended objects of know size and flux distribution. These objects are chosen such that they satisfy the sampling theorems. In other words, reconstruct the image you would get if you had a better instrument (rather than a perfect instrument).

– So now the image has a constant PSF, which MCS call r(x). This introduces a length scale over which the image must be smooth (?). This applies to point sources (which have shape r(x)) and extended sources. From the solution space of lightly-deconvolved scenes, we should choose the one which gives maximum smoothness on this local scale.

– Specifically, for each pixel we take the difference of the "background" (everything which isn't delta) from the "reconstrcuted background" (the fixed PSF convolved with the scene); then sum over pixels and minimise (equation 7). This procedure discards high-frequency information, but is consistent with the adopted sampling and the frequencies of r(x).

– Artefacts not stealing flux AND no smoothing of point sources -¿ photometry possible.

– Requires no positivity constraint.

Usage – Using simulated and real astronomical images, with finite resolution and noise, the new procedure is compared with other standard procedures and does (stupidly) well. They are able to recover fluxes and positions to high accuracy, and they avoid exacerbating noise / artefacts in the image.

– Image combination is also demonstrated – deconvolution of many images to the same PSF before combining them yields high-resolution final image.

Further Work – Devise a more robust optimisation that finds minimum even in populated images.

B Light Deconvolution Procedure According to Magain et al. (1998)

In section 2.5, we described our procedure for deconvolving an image to a pre-set target PSF. Here we outline a more robust and powerful method due to Magain et al. (1998). Instead of finding the deconvolved image, $\mathcal{A}$, by minimising the residuals for equation 6, we split the final image into its point sources – amplitude-α δ-functions at (2D) position c –, and a smooth background, h; then we minimise the functional $S\left[\{\alpha\},\{c\},h \,|\, d,\sigma,r,k,\lambda\right]$ with respect to its left-hand arguments:

$$S = \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \left[\sum_{j=1}^{N} k_{ij} \left(h_j + \sum_{k=1}^{M} \alpha_k r(x_j - c_k) \right) - d_i \right]^2 \; + \; \lambda \sum_{i=1}^{N} \left(h_i - \sum_{j=1}^{N} r_{ij} h_j \right)^2 \tag{20}$$

(equation (7) of their paper). This requires some explanation. First, what do all the symbols denote?[10] This is listed in table 1.

Variable	Description	Role
S	A functional	For minimisation
σ_i	Standard deviation of the image at pixel i. Has N elements.	Calculated from the data.
k_{ij}	Deconvolution kernel. Has a user-specified number of elements.	Calculated here from PSFEx…
h_j	The pixels of the "true" image or scene which describe everything except the point sources. N elements.	This is an object to be found via the minimisation procedure. Requires an intial guess, h^0.
α_k	Enodes the intenitisties of the image's point sources. Number of elements, M obviously depends on the image.	This is an object to be found via the minimisation procedure. Requires an intial guess, a^0.
c_k	Enodes the positions of the image's point sources. Since each source has two coordinates, c has $2M$ elements.	This is an object to be found via the minimisation procedure. Requires an intial guess, c^0.
$r(x_j)$	The target PSF. Size is set by user.	From PSFEx…
d_i	Original image, or data.	What we start with.
λ	A lagrange multiplier.	Set by the user to ensure the deconvolved image has the right statistical properties.

Table 1: Listing the variables in equation 20, from left to right.

Now we know what all the symbols mean, we can begin to see some structures.

$$\left(h_j + \sum_{k=1}^{M} \alpha_k r(x_j - c_k)\right) \tag{21}$$

is simply our reconstructed image, with background and point sources, while

$$s_{ij}\left(h_j + \sum_{k=1}^{M} \alpha_k r(x_j - c_k)\right) \tag{22}$$

"re-blurs" the reconstruction for comparison with the data, d_i. The first term of equation 20 is therefore a χ^2 term for our model and our data, albeit with the model decomposed into two pieces.

The second term is included to ensure smoothness of the background… **MORE**

[10]Note that in the argument of S above, we've dropped the vector/matrix notations. Indeed, the form of the variables is somewhat elastic, depending on how we choose to set up the problem. So although it might be tempting at first glance to assume the single-index variables of equation 20 are like vectors and the double-index variables are like matrices, we actually have some freedom in how to express them. For instance, the "vector" d_i represents the original *2D* image. So in order to understand what's going on in the equation, we should keep in mind the *number* of independent elements each object has.

We can re-express the minimisational functional in terms of a linear algebra operation. The most immediate way is

$$S\left[\vec{\alpha}', \vec{h}\right] = \left\{\frac{1}{\vec{\sigma}}\left[\mathbf{k}\left(\vec{h} + \vec{\alpha}' * r\right) - \vec{d}\right]\right\}^2 \; + \; \lambda\left(\vec{h} - \mathbf{r}\vec{h}\right)^2, \tag{23}$$

where, to be totally explicit, we've listed all the variables again in table 2.

Variable	Description	Comments
S	A functional	For minimisation
$\vec{\sigma}$	Standard deviation of image at each pixel. A vector of length N.	Calculated from the data.
$\mathbf{k}$	Deconvolution kernel. Has a user-specified number of independent elements, but is here represented by an $N \times N$ matrix.	Calculated from PSFEx. . .
$\vec{h}$	The pixels of the "true" image or scene which describe everything except the point sources. Vector of length N.	This is an object to be found via the minimisation procedure. Requires an intial guess, $\vec{h}^0$.
$\vec{\alpha}'$	Enodes the intenitisties of the image's point sources. Number of elements, M obviously depends on the image.	This is an object to be found via the minimisation procedure. Requires an intial guess, $\vec{\alpha}'^0$.
r	The target PSF. Size is set by user.	From PSFEx. . .
$\vec{d}$	Original image, or data. N-element vector.	What we start with.
λ	A Lagrange multiplier.	Set by the user to ensure the deconvolved image has the right statistical properties.

Table 2: Listing the variables in equation 23, from left to right.

We can still readily see the structure of what's going on (see comments following equation 20). One issue that immediately presents itself is how very expensive the minimisation procedure is. Each step will require the evaluation of S at least *five* **TODO: Cato: is this true?** times (for calculating S and its numerical derivatives); while each step requires three matrix-vector multiplications, which require N^2 operations (N, remember, is the number of pixels, which could easily be more than a million). So at the end of the day, we are left with a minimisation procedure which needs around $15N^2$ ops per iteration. Using the conjugate gradient method, we may be looking at up to N iterations, giving us a grand total of $\mathcal{O}(10) \cdot N^3$ operations per image. **TODO: any improvement on this would be good**.

Another obvious question which must be answered is whence the initial guess for the point and extended sources, $\vec{\alpha}'$ and $\vec{h}$, come. Note that it is not enough to simply use the original image, since our algorithm needs the point sources to be distinguished from everything else. We have already generated a catalogue of all the sources (along with their coordinates) in our image using SExtractor. We've also used PSFEx to determine which of the sources were point sources – if we can extract this information from the software **TODO: Cato: look into this, otherwise whole algorithm will be much slower**, it should be possible to make a very good estimate for the initial image.

——

Now we outline the steps required to minimise the functional S from equation 23.

1. Generate an initial guess for $\vec{\alpha'}$ and $\vec{h}$ using output from SExtractor and PSFEx.
2. Calculate S from equation 23 with these values (pseudocode below).
3. Calculate also the derivatives of S with respect to the parameters of interest, using the conjugate gradient method (pseudocode below).
4. Apply steps 2 and 3 iteratively until the minimum of S is found.
5. Compute the residual, that is the "reconvolved" image ($\mathbf{k}\left(\vec{h} + \vec{\alpha'} * r\right)$) minus the original image. Check whether this is roughly equal to N.
   - If so, we are done.
   - If not, we must allow λ to vary over the image, i.e. $\lambda \to \lambda(\vec{x})$. Recompute residuals until their sum $\simeq N$.

——

Here is some pseudocode for calculating S.

```
1. Do it
```

——

Here is pseudocode for calculating the derivatives of S.

```
1. Do it
```

TODO: Cato: read about conjugate gradient method: Shewchuk (1994)

C Lupton et al. (2004) – "Preparing Red-Green-Blue Images from CCD Data"

Quote 'sheer drama and beauty of the night sky'

Key Idea – There is a lot of information in the colouration of an image. This often helps us distinguish features / phenomena and classify astronomical objects.

– Hitherto, focus has been on *intensity* differences.

Background – We apply stretches to images in order to coax faint objects into observability. But we must strike a balance between this objective and the saturation of bright parts.

– Stretch is a re-scale, bringing all objects to within a brightness cutoff range. Re-scale can be linear, ln, sqrt, depending on preference and the diversity of images.

– Tuning parameters is not always straightforward. Any object above maximum brightness ends up bleached and obese.

– Furthermore, there is degeneracy between brightness and colour in traditional stretching procedures.

Solution – Using a different stretching procedure, (equation 2), can discard uninteresting intensity information in favour of colour information. This works by comparing the individual colour-intensities to the total intensity (i.e. compare the colours amongst themselves), and comparing the total instensity to the two cutoff intensities which define the brightness scale.

– NB the colours are unique – no degeneracy with intensity. So we can draw unambigouous conclusions from looking at colour differences.

– arsinh stretch magnifies faint objects (linear regime) and avoids bleaching bright objects (logarithmic regime). (But this could be achieved with other functions too).

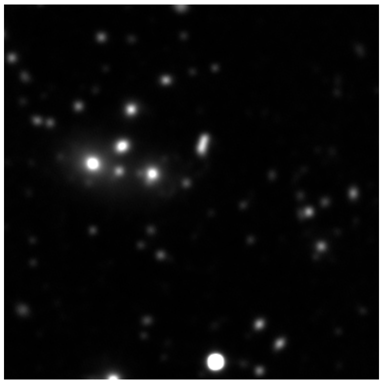
Examples – Some examples are given where the standard technique loses an embarrassing amount of detail compared with the new idea. By eye, we clearly distinguish differences between objects which are otherwise just rendered as white blobs.

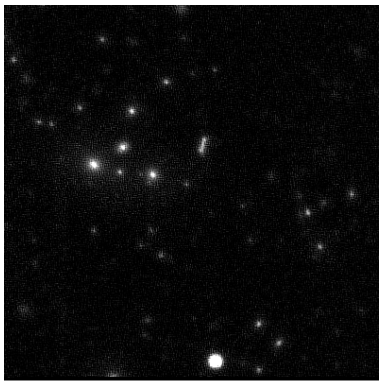
D Home-Brew Convolution Code

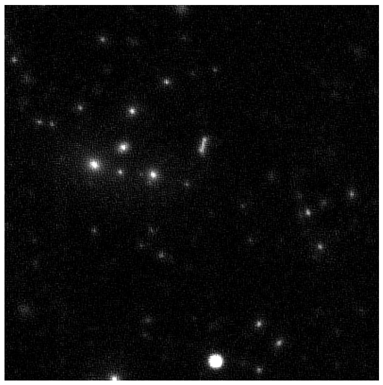
TODO: Cato: Needs tidying / deleting

Alternatively, we can simply use a pre-existing module which performs the convolution for us: `scipy.signal.fftconvolve` does the job nicely. We can even tell this module to reduce the size of the convolved image (`mode="valid"`), in keeping with the concerns raised in section 2.4.2.

However, it would be naïve to simply place our trust in SciPy – their convolution method may be subtly different from the one we wish to use for scientific inference. Thus, we cannot escape having to implement the concolution procedure of equation 6. Figure 15 on the following page shows a comparison, with residuals. Leaving aside the image artefficts from using a small convolution kernel (**MORE** FIX!), we can see that the two methods do not agree completely. This is worrying, but according to DWH the problem is almost certainly with my implementation rather than SciPy's, and I am inclined to agree (it might be a simple indexing error). Combined with the fact that SciPy is around 15 times faster, it makes little sense to labour on with my code; so we adopt the SciPy convolution module.


(a) Original image


(b) Deconvolved using my procedure and a 3x3 kernel.


(c) Deconvolved using SciPy's procedure and a 3x3 kernel.

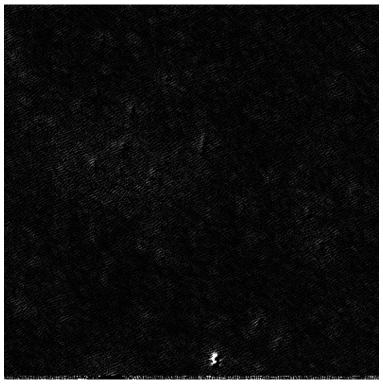

(d) Residuals.

Figure 15: Deconvolution procedure – a comparison of my home-made convolution algorithm and SciPy's optimised one. SciPy takes about 1/15 of the time. Note that a 3x3 kernel was used for this test, and there are obvious artefacts introduced around bright points of the image.

References

E. Bertin. Automated Morphometry with SExtractor and PSFEx. In I. N. Evans, A. Accomazzi, D. J. Mink, and A. H. Rots, editors, *Astronomical Data Analysis Software and Systems XX*, volume 442 of *Astronomical Society of the Pacific Conference Series*, page 435, July 2011. 2.2

E. Bertin and S. Arnouts. SExtractor: Software for Source Extraction. A&AS, 117:393–404, June 1996. 2.1

Gwendoline Blanchet, Lionel Moisan, and Bernard Rougé. Automatic detection of well sampled images via a new ringing measure. pages 1030–1033. IEEE, 2010. ISBN 978-1-4244-4296-6. 4.2

R. J. Buta. Galaxy Morphology. *ArXiv e-prints*, February 2011. 1

L. Fortson, K. Masters, R. Nichol, K. Borne, E. Edmondson, C. Lintott, J. Raddick, K. Schawinski, and J. Wallin. Galaxy Zoo: Morphological Classification and Citizen Science. *ArXiv e-prints*, April 2011. 1

O. Lahav. Galaxy classification by human eyes and by artificial neural networks. *Astrophysical Letters and Communications*, 31:73, 1995. 1

O. Lahav, A. Naim, R. J. Buta, H. G. Corwin, G. de Vaucouleurs, A. Dressler, J. P. Huchra, S. van den Bergh, S. Raychaudhury, L. Sodre, Jr., and M. C. Storrie-Lombardi. Galaxies, Human Eyes, and Artificial Neural Networks. *Science*, 267:859–862, February 1995. doi: 10.1126/science.267.5199.859. 1

C. Louchet and L. Moisan. Total variation denoising using posterior expectation. In *European Signal Processing Conference (Eusipco)*, 2008. 4.2

LSST Science Collaboration, P. A. Abell, J. Allison, S. F. Anderson, J. R. Andrew, J. R. P. Angel, L. Armus, D. Arnett, S. J. Asztalos, T. S. Axelrod, and et al. LSST Science Book, Version 2.0. *ArXiv e-prints*, December 2009. 1

R. Lupton, M. R. Blanton, G. Fekete, D. W. Hogg, W. O'Mullane, A. Szalay, and N. Wherry. Preparing Red-Green-Blue Images from CCD Data. PASP, 116:133–137, February 2004. doi: 10.1086/382245. (document), 3, C

P. Magain, F. Courbin, and S. Sohy. Deconvolution with Correct Sampling. ApJ, 494:472, February 1998. doi: 10.1086/305187. (document), 1, 2, 2, 2.7, A, B

L. Moisan. How to discretize the total variation of an image? *PAMM*, 7(1):1041907–1041908, 2007. 4.2

R. Nityananda and R. Narayan. Maximum entropy image reconstruction - A practical non-information-theoretic approach. *Journal of Astrophysics and Astronomy*, 3:419–450, December 1982. doi: 10.1007/BF02714884. 1

M. Oguri and P. J. Marshall. Gravitationally lensed quasars and supernovae in future wide-field optical imaging surveys. MNRAS, 405:2579–2593, July 2010. doi: 10.1111/j.1365-2966.2010.16639.x. 1

R. S. Pawase, C. Faure, F. Courbin, R. Kokotanekova, and G. Meylan. A seven square degrees survey for galaxy-scale gravitational lenses with the HST imaging archive. *ArXiv e-prints*, June 2012. 1, 4

R. K. Pina and R. C. Puetter. Bayesian image reconstruction - The pixon and optimal image modeling. PASP, 105:630–637, June 1993. doi: 10.1086/133207. 1

A. Refregier, A. Amara, T. D. Kitching, A. Rassat, R. Scaramella, J. Weller, and f. t. Euclid Imaging Consortium. Euclid Imaging Consortium Science Book. *ArXiv e-prints*, January 2010. 1

W. H. Richardson. Bayesian-Based Iterative Method of Image Restoration. *Journal of the Optical Society of America (1917-1983)*, 62:55, January 1972. 1

M. E. Schwamb, J. A. Orosz, J. A. Carter, W. F. Welsh, D. A. Fischer, G. Torres, A. W. Howard, J. R. Crepp, W. C. Keel, C. J. Lintott, N. A. Kaib, D. Terrell, R. Gagliano, K. J. Jek, M. Parrish, A. M. Smith, S. Lynn, R. J. Simpson, M. J. Giguere, and K. Schawinski. Planet Hunters: A Transiting Circumbinary Planet in a Quadruple Star System. *ArXiv e-prints*, October 2012. 1

Jonathan R Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Pittsburgh, PA, USA, 1994. B

J. Skilling and R. K. Bryan. Maximum Entropy Image Reconstruction - General Algorithm. MNRAS, 211:111, November 1984. 1