Discovery Sensitivity in Human Viewable Images

Cato Sandford*, Phil Marshall, David Hogg

October 29, 2012

Abstract

A sketch of my project with DWH and PJM: image deconvolution and colour-composition.

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 artifacts 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 artifacts 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 artifacts if not sufficiently regularized (see, for example, comments at the end of section 1 of Magain et al., 1998): such artifacts 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 artifacts?
- 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 focus on lensed features as our discovery targets, and the XXX 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 **??**.

1.1 The Point-Spread Function (PSF)

- What is it?
- Whence?
- Different in every image, sometimes varies within image
- How to identify

1.2 Bandpasses and Colour Images

- Photometric applications – cheaper than spectroscopy (don't throw away photons) but yields information
- Build a full picture by co-adding. But from section 1.1 we know this can be problematic.

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.

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

TODO: Update

[5] Well, not actually circularly symmetric.

2.1 Source Identification

- Use SExtractor to identify the sources in an image
- How does it work?
- What else does it measure / output?

- 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.

2.2 Determining the PSF of an Image

- Use PSFEx to compute $\mathrm{PSF}_{\mathrm{obs}}$ (arbitrary shape, noisy)
- How does it work? Use that graph in the docs.
- What else does it measure / output?

- 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`).

2.3 Generating a Target PSF

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

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

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.

2.3.1 Pseudocode

...

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 $\vec{k}$)[6] 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 9×9 image, as shown in figure 2.4.

FIGURE: Convolution → linear algebra

MORE

[6] 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.

B is then a matrix with size(a) rows and size(k) columns. The vector problem in equation 4 is underdetermined, so we must use some optimisation 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() method constructs rectangular Toepliz matrix of dimensions dim
7. kernel = solve(psf_ref, psf_obs)
8. kernel = reshape(kernel, kernel_dim)
9. save_image(kernel)
10. return kernel
```

In practice, we have some idea of what the kernel should look like: a bright central element... We can use this prior guess to speed up the convergence of the algorithm **MORE** needs pseudocode.

2.4.2 Interlude: Honesty Note

TODO: Cato: this might be confusing – reverse A and B?

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 $\mathcal{A}$ with a 3×3-pixel image to get $\mathcal{B}$. The first (i.e. top-left) pixel of $\mathcal{B}$ that we can properly determine is not the same as the first pixel of $\mathcal{A}$, because **MORE**.

Thus, the honest thing to do is to throw away some information by making image $\mathcal{B}$ smaller than $\mathcal{A}$ – in this example with a 3×3 image, we shave off the four outer edges, so if $\mathcal{A}$ is $N \times M$, $\mathcal{B}$ 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. Hmm...

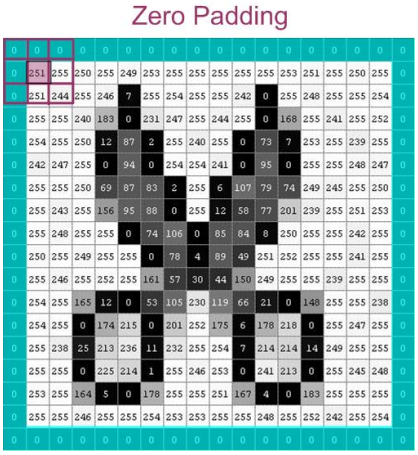


Figure 1: An image padded with zeroes. Convolution with the 3×3 stencil will now preserve image size; but at what cost?

2.5 Full Image Deconvolution

In section 2.4 we found the kernel k which would transform a PSF with known properties (PSF_{ref}) into the observed PSF (PSF_{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 $\mathcal{B}$.

But unfortunately that 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 on page 17.

2. The procedure outlined in this section may be highly over-determined, and so there is a lot of flexibility in the solution. Typically, this will introduce artifacts 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 overkill. Instead, a more modest regularisation scheme is proposed in section 2.7.

2.5.1 Pseudocode

```
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)
```

Consider for a moment point number 5. 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.[7]

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`

[7] Details – if the same stopping tolerances atol and btol are used for each system, the number of iterations for the two methods will be similar, but the final solution x0 + dx should be more accurate. The only way to reduce the total work is to use a larger stopping tolerance for the second system. If some value btol is suitable for A*x = b, the larger value btol*norm(b)/norm(r0) should be suitable for K*db = r0.

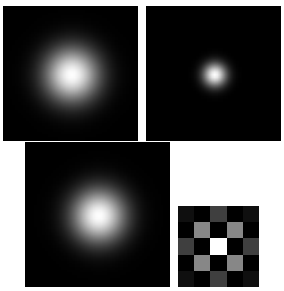


Figure 2: original image, target PSF, deconvolved image and the kernel

8. `refimg_vec = stretch(refimg_vec)`
   `## Ease of viewing`
9. `refimg = reshape(refimg_vec)`
   `## Make into an image-worthy shape`
10. `save_image(refimg)`

2.5.2 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(`$\text{PSF}_{\text{ref}} \rightarrow \text{PSF}_{\text{ref}}$`)`
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 2, 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 (2). 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.

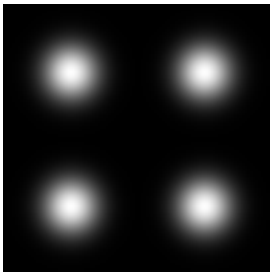


Figure 3

- 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.

—

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 **??**.

Next Hogg emphasised the following (26/10):

- The data image should be bigger than the scene;

i.e. pad the data with zeros. This goes against the discussion of section 2.4.2. The reason for it is that we want to solve an overdetermined system rather than underdetermined (I had previously assumed /tried to solve the underdetermined system and then regularise / choose one solution somehow). Since this only causes problems at the edges, the larger the image, the smaller the problem.

- To speed things 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.

2.5.3 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 4. Simple code is on page 16.

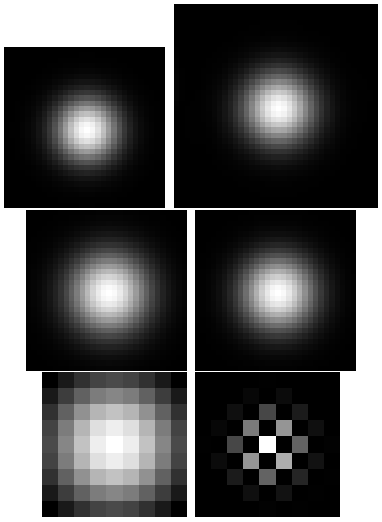


Figure 4: Scene, data and kernel (scaled 2x,2x,6x). On left is the "correct" image (2D Gaussian). On the right is output of program. Note each image has different intesity scale which is unhelpful.

```
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
```

2.6 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:

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

TODO: Cato: read up on this.

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 $\mathbf{C}$, that using this matrix equation in 13 yields equation 15.

2.6.1 Pseudocode

...

2.6.2 Noise Model

Having established that our minimisation functional, S, should include the data covariance matrix $\mathbf{C}$, 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 $\mathbf{C}$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[8]. 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".

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 corellated 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 inensity 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

[8] **TODO: Cato: what does "heteroscedastic" mean? Relevant?**

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

...

2.6.4 Precomputed Covariance Matrices

Note that inclusion of this covariance matrix has an additional advantage.

- Weight information from other sources, e.g. telescope operator who knows about bum pixels, saturated pixels
- Block out missing pixels so they aren't involved in inference.

2.7 Regularisation

Occam's razor

FIGURE: Before smoothing, after smoothing

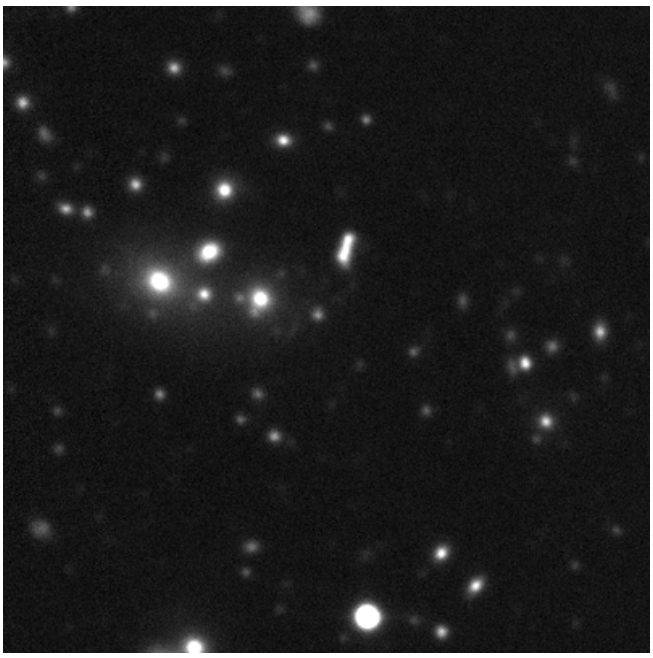
2.8 Image Manipulations

- Making them viewable by humans – stretch. **FIGURE:** Before after stretch
- .psf file conversion
- FITS conversion

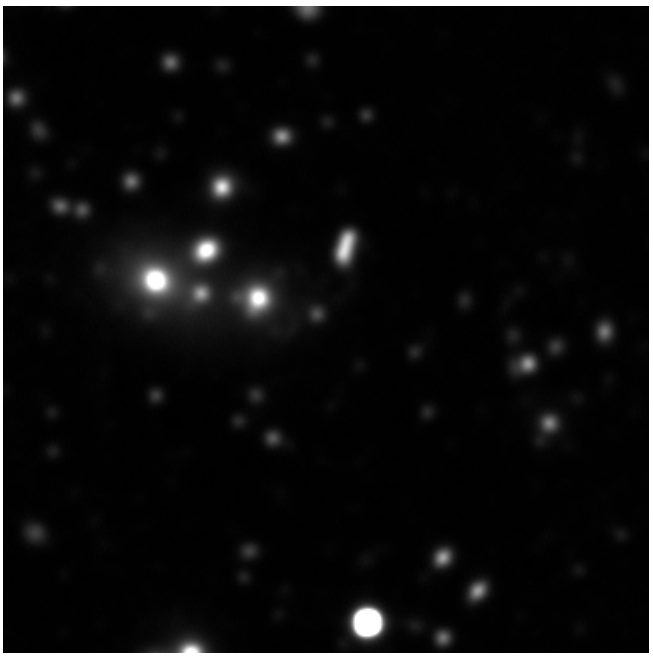
2.8.1 Stretch: Noise Properties

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 severe as you go down.

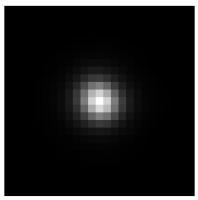
Specific notes:
* I used a 9x9 kernel for these images.
* To change the strength of the stretch, I simply divided the upper bound (you called it "vmax") by 2 and 10.



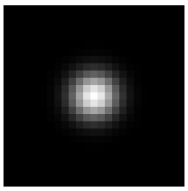
CFHTLS_03_g_sci_rescaled.png



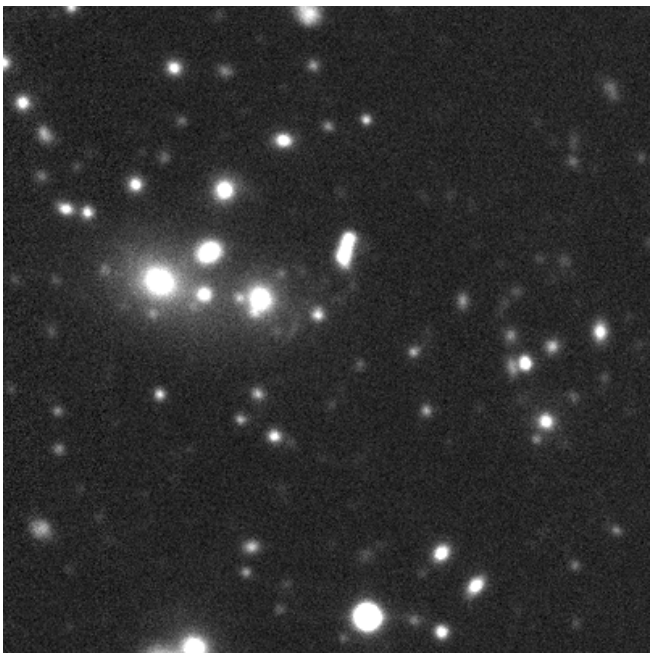
gsc_CFHTLS_03_g_sci_rescaled.png



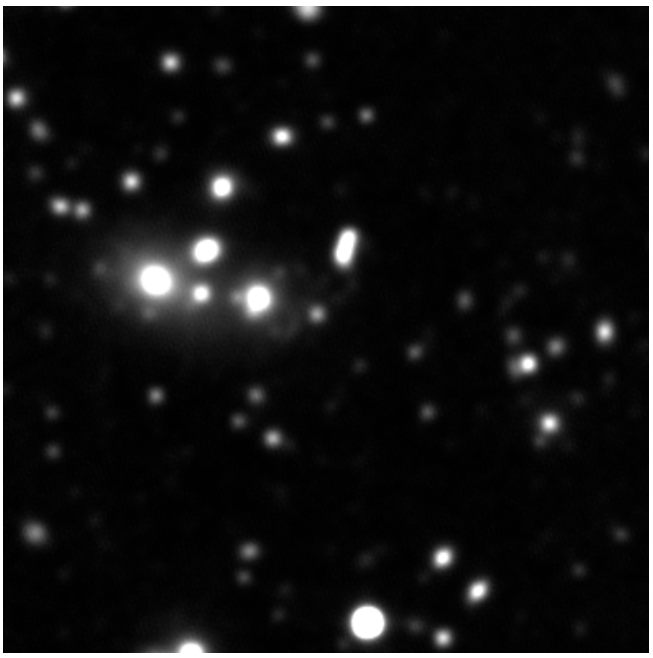
snap_n1_d0_CFHTLS_03_g_sci_z8.png



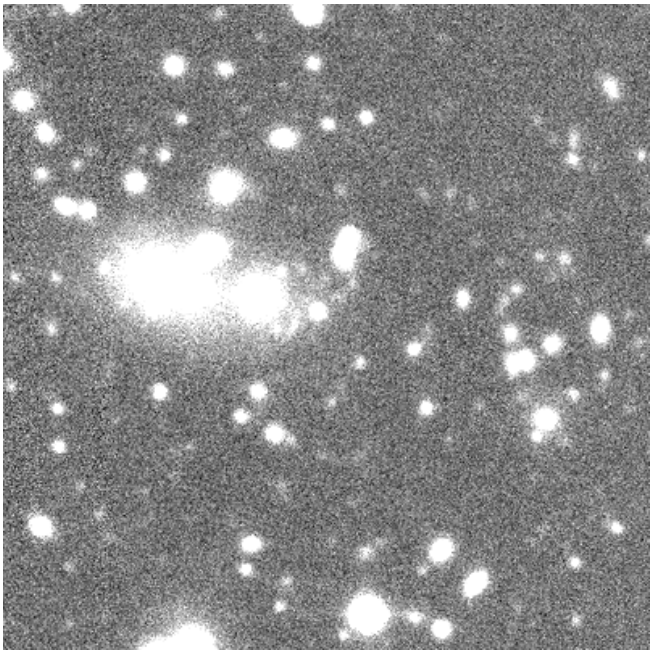
snap_n1_d0_CFHTLS_03_g_sci_Gauss_z8.png



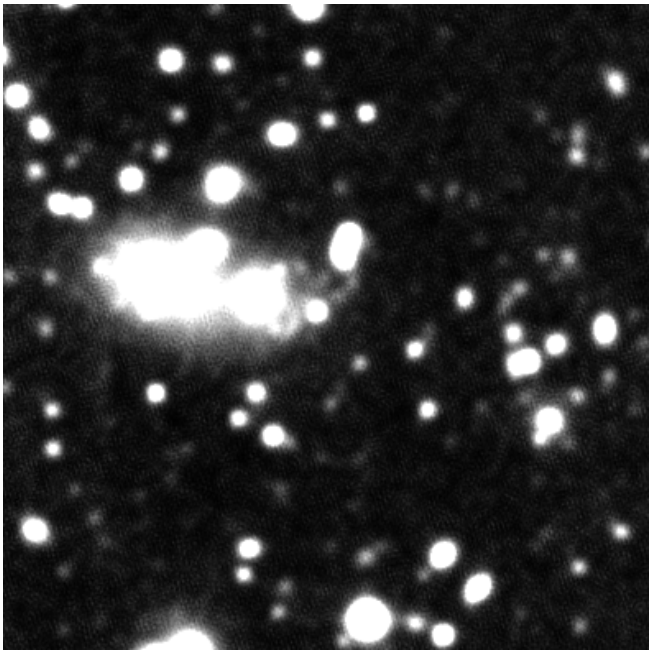
CFHTLS_03_g_sci_rescaledl.png



gsc_CFHTLS_03_g_sci_rescaledl.png



CFHTLS_03_g_sci_rescaledll.png



gsc_CFHTLS_03_g_sci_rescaledll.png

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.

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.

A summary of this part of the project (with pseudocode) follows.

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)`

   - `pixtot = R_array+G_array+B_array ## collapse images onto each other`
   - `if pixtot[i,j]==0:  pixtot[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 found that this was an unnecessary IDL-specific step, and it was unnecessary to implement it.

2. Integrated with PJM's HumVI code, so that choice of Lupton/Wherry procedure is an option.
   - Wherry is default. Luption can be invoked with command-line keyword `--lupton` or simply `-l`.
   - 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 `--lupton` is specified.

3.1 Finding Optimal Parameters for the Image Manipulation

'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. Probably we would want to supply a few different standard forms as well (and by "form" I mean scales,nlin). 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?" I have no idea where one sends paper like this - Ill ask the zooniverse team.'

4 Combined Deconvolution and Colour

??

Magain et al. (1998) – must deconvolve all bands to target PSF before combining.

5 To do

Document

- Invert figure colours to save ink
  [Latex can't do this, I'll have to make them myself.]

Colour

- Make some side-by-side composites showing Wherry and Lupton
- Fail to compute scales → error
- Should work with one or two FITS images.

Deconvolution

- Put in covariance matrix stuff
- How does kernel size affect execution time?
- Regularisation method
- Invent quality metrics: each one represents the quantitiave answer to a particular question about some aspect of the reconstructed image – its "noise" level, the correlated nature of that noise, the number of false SExtractor detections (in the reconstructed image) generated, the SExtractor flux of certain objects of interest, the number of inspectors who identify feature X, etc etc etc. We can propose and discuss these aspects after seeing some images.
- arcsec→ pixels method; will need header information.
- Assert target PSFs of width 0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0 arcsec FWHM, and make deconvolved images for each value. Then plot your quality metric(s) against FWHM.

Combining C&D

- Determine some metrics for successful combination / sensitivity to interesting features. Modify stretch / $\mathrm{PSF}_{\mathrm{ref}}$ appropriately.

Maths

- PHIL: 'I would like to see a derivation of this procedure, which involves a matrix acting on a noisy vector. Do you start by writing down the principled probabilistic inference of a model image given noisy data, and end up showing that this boils down (under certain assumptions) to the matrix operations you perform? Any improvements we make to your code will probably be of the form "assign a different prior to the pixel values of the kernel/final image", so it'd be good to see how that propagates through into new matrix operations.'
- Prior on pixel values → Lagrange term added to the summed squared residuals.

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:

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

(equation (7) of their paper). This requires some explanation. First, what do all the symbols denote?[9] 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**

[9]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).

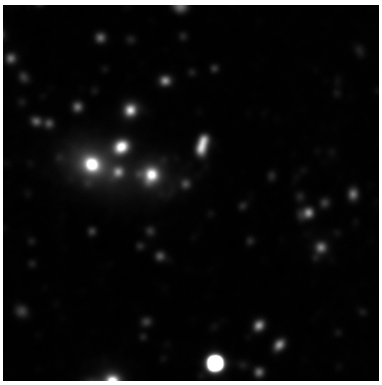
Examples – Some examples are given where the standard technique loses an embarrassing amount of detail compared with the new idea. By eye, we clearly distinguish differences between objects which are otherwise just rendered as white blobs.

D Home-Brew Convolution Code

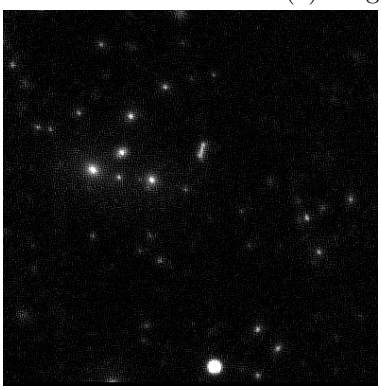
TODO: Cato: Needs tidying / deleting

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.

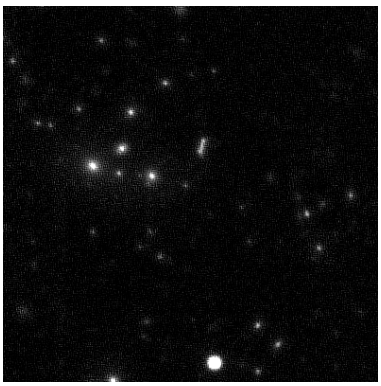
However, it would be naïve to simply place our trust in SciPy – their convolution method may be subtly different from the one we wish to use for scientific inference. Thus, we cannot escape having to implement the concolution procedure of equation 6. Figure 5 on the following page shows a comparison, with residuals. Leaving aside the image artefacts 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.



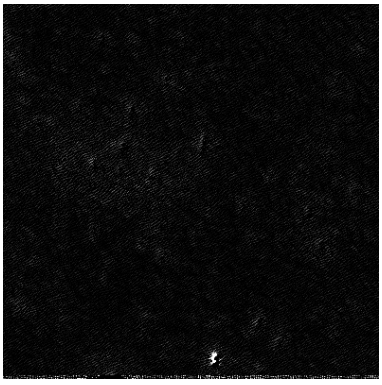
(a) Original image



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



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



(d) Residuals.

Figure 5: 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 artifacts introduced around bright points of the image.

References

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

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, 4, A, B

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