Attempting to Understand Power Spectra

Lynne Jones, Peter Yoachim, Željko Ivezić

A note on some code and methods around calculating and inverting images and their related power spectral densities and auto correlation functions – including the 1-d PSD and ACF and structure function.

1. Code to calculate and invert the FFT/PSD/ACF and 1-D PSD/ACF

The code developed to calculate the FFT, PSD (Power Spectrum Density), and ACF (AutoCorrelation Function) of an image is available in the github repository at `http://github.com/rhiannonlynne/powerspectrum`. In particular, the basic functionality related to simply calculating (and inverting) these values is in the PImage class (see pImage.py). To supplement the documentation within the class, a short summary of the functionality of this class is covered here, along with some comments on the process of calculating and inverting images/FFTs/PSD/ACFs and the corresponding 1-d PSD/ACF. Plotting functions specifically designed to make it easy to visualize the members of the class are available in PImagePlots, which inherits from PImage (see pImagePlots.py). The plots in this note were generated using functionality from PImagePlots.

1.1. Forward calculation of the FFT, PSD, ACF and 1-d PSD and ACF

First let's look at the forward calculation, from image through FFT, then PSD (and 1-d PSD), then onto the ACF (and 1-d ACF and SF).

The fourier transform of the image (the FFT) is calculated using scipy's fftpack utilities. As we're transforming images, these are evenly sampled, finite inputs to the FFT - so the fast fourier transform code in scipy is well-suited to calculating what is essentially the DFT of the images. The FFT of the image is calculated using scipy.fftpack.fft2, and the related frequencies corresponding to each resulting pixel are calculated using scipy.fftpack.fftfreq.

It's worth bringing up at this point that scipy's fft (and fft2) by default places the smallest frequency components (i.e. largest spatial scales) at the edges of the resulting FFT. To present a more natural image, and for purposes of calculating radial 1-d PSD and ACF profiles, it is useful to use scipy.fftpack.fftshift to bring these small frequencies into the center of the resulting FFT. The PImage class offers the option to choose to create a series of either 'shifted' (i.e. using fftshift) or non-shifted versions of the FFT, 2-d PSD and 2-d ACF, and will keep track of which version you're using, including using the correct version to build the 1-d PSD and ACF profiles. The default is to use shifted versions of the FFT, PSD and ACF.

The 2-d power spectrum density (2-d PSD) is then calculated by squaring the absolute value

of the 2-d FFT (including both the real and imaginary components), as

$$2D\,PSD = \left|(R(u,v)^2 + I(u,v)^2\right| \tag{1}$$

(implemented in PImage as numpy.absolute(FFT)**2, which translates to the same calculation). Note that even the 2-d PSD only captures the amplitude of the FFT variations; this can be generally translated to capturing variations in the intensity of the original image over the range of scales in the PSD. The locations within the image where these variations occurred is captured with the 2-d phase spectrum, which is calculated as

$$\text{Phase spectrum} = arctan\left(\frac{I(u,v)}{R(u,v)}\right) \tag{2}$$

(implemented in PImage as numpy.arctan2(FFT.imag, FFT.real). This 2-d phase spectrum can be highly structured, and without the phase information preserved here, it is impossible to faithfully reconstruct the original image.

The 2-d auto correlation function (2-d ACF) is simply the inverse fourier transform of the 2-d PSD, calculated using scipy.fftpack.ifft2. This relationship is established by the Wiener-Khinchine (or Wiener-Khintchine) theorem. This calculation requires some bookkeeping regarding shifted or unshifted FFT images, but this is handled by the PImage class.

This brings us to the calculation of the 1-d PSD and the 1-d ACF. Much of the calculation of these 1-d versions overlaps: the work is basically creating an azimuthally averaged, radial profile of the 2-d version of the PSD or the ACF. The radial profiles are calculated with r=0 at the center of the image, thus the shifted version of the PSD or ACF is used. The radial profiles are calculated relatively quickly using numpy histograms, which include weighting the radial profile by the number of pixels in each radial bin. In order to reduce noise in (and the size of) the 1-d PSD or 1-d ACF, both the minimum number of pixels in each radial bin (min_npix) and the minimum radial step (min_dr) can be specified in the code. The minimum pixels consideration (min_npix) kicks in near the center of the 2-d PSD, where the frequency scales are smallest and corresponding spatial scales are largest, and similarly at the center of the 2-d ACF, where the spatial scales are smallest, but for min_npix values on the order of 10 or less, this usually only affects the first few radius steps, even if min_dr is as small as 1 pixel. The resulting 1-d PSD or ACF may not have uniform spacing in the radial profile, but the information on the step size is preserved in either frequency or spatial values corresponding to each 1-d radial profile data point.

Finally, the structure function (SF) is generated from the 1-d ACF by simply calculating $(1 - ACF)$. Here, a numpy 1-d linear interpolation is used to create a SF with evenly binned spacing from the center of the image to the edge. In addition, the SF is arbitrarily scaled so that it runs between 0 and 1.

Two other methods are available that can be useful for this calculation: the image can be zero padded at its edges and a hanning filter can be applied. The hanning filter gradually scales the image from its normal peak values down to 0 at $r = r_{max}$, where r_{max} is user definable (the

default is the edge of the image). Without the hanning filter, any discontinuities at the edges of the original image (even if the image is zero padded - there is still a discontinuity between the image values and the zero values in the padding region) will result in noise in the FFT and PSD. This is most noticeable in the PSD, as this is typically viewed on a log scale, and in the phase spectrum. Vertical and horizontal lines in the 2-d PSD aligned with the u/v axis are a typical indicator of this noise. An example of this entire forward process is shown in Figures 1 and 2, both with and without the hanning filter[1].

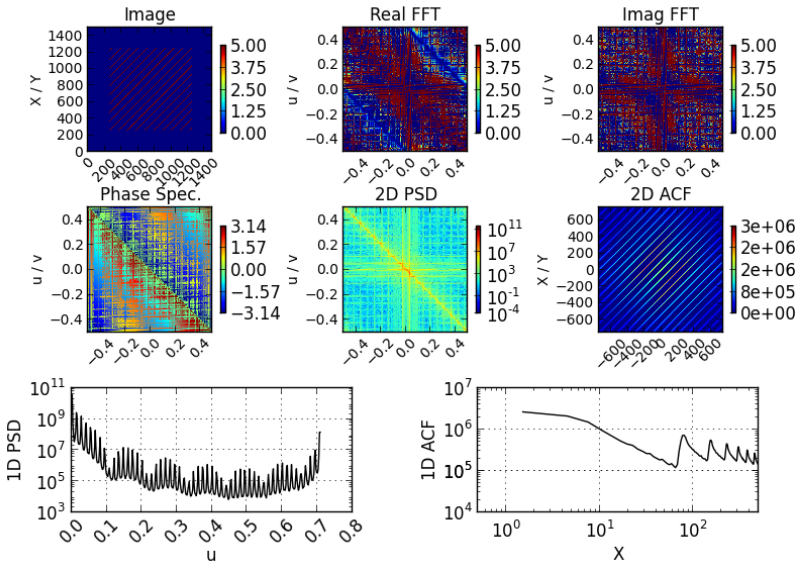


Fig. 1.—: Example of FFT, PSD, and ACF calculation from a simple image, including the 1-d PSD and 1-d ACF. This example starts with a simple image filled with lines at a 45 degree angle, with zero padding around the outside of the image.

1.2. Gaussian example

Walking through the transformations of a 2-d gaussian is illustrative, because the 2-d gaussian has an analytical solution to the FFT and on down the chain to the 2-d ACF. An overview of the 2-d FFT, 2-d PSD and 2-d ACF, along with the 1-d PSD and ACF is shown in Figure 3.

To explore this further, however, we can look at slices through the center of the 2-d images, to compare the expected analytical representations of the gaussian as it translates from original image to FFT to 2-d PSD and to 2-d ACF with the actual calculated widths of the gaussian. This makes

[1]Larger sizes of these images are available in the github repository, along with code (powers_qs.py) which will recreate all figures in this note from scratch.

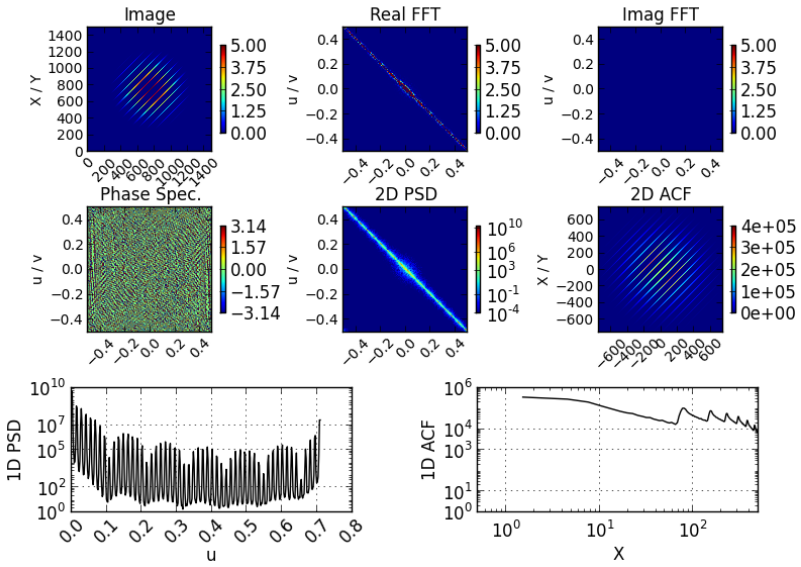


Fig. 2.—: Adding a hanning filter. This example is similar to Figure 1, except a hanning filter has been applied before zero padding the image. The FFT and 2-d PSD are much cleaner (and closer to the true result which would be achieved if the image was infinitely large) with the hanning filter applied.

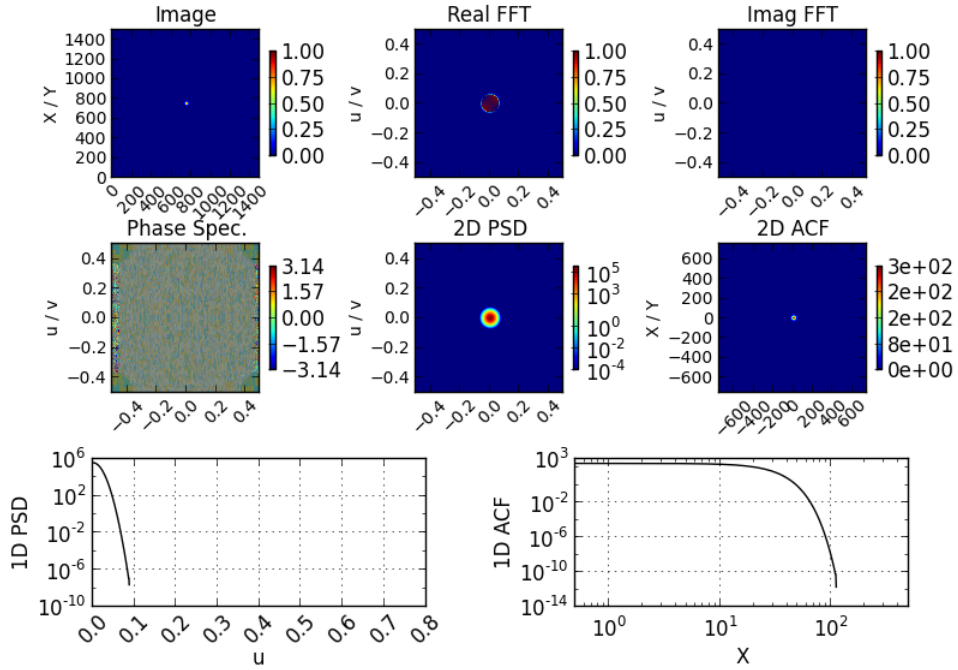


Fig. 3.—: Calculation of the FFT, PSD and ACF for a 2-d Gaussian in the input image.

sure that the scale of our various 'images' (of the FFT/PSD/ACF) are understood, particularly as we translate in and out of frequency space. In theory, the peak of the gaussian in each case should be calculable as well. In practice, I did not try to predict the peaks and instead just used the maximum value of the calculated values as the 'expected' peak. (Predicting this may be a to-do, but understanding the overall structure and spatial scaling is a higher priority).

Following along with the panels in Figure 4:

- Panel a shows a slice through the center of the original image, where we see a gaussian with a width of σ_x.
- Panel b shows a slice through the center of the real part of the FFT, where we see a gaussian with a width of $\sigma_{fft} = 1/(2\pi\sigma_x))$. This is as expected - the FFT of a gaussian is another gaussian, with this predicted width. Note that the x axis units here are in frequency.
- Panel c shows a slice through the center of the 2-d PSD, which is created by squaring the FFT. Since the square of a gaussian is another gaussian with width reduced by $\sqrt{2}$, we expect $\sigma_{psd} = \sigma_{fft}/\sqrt{2}$. Note that this is still in frequency space.
- Panel d shows a slice through the center of the 2-d ACF, which is created by taking the inverse FFT of the PSD. Thus we expect $\sigma_{act} = 1/(2\pi\sigma_{psd}) = \sigma_x * \sqrt{2}$. Note that now we are back in pixel space, where each pixel corresponds to the same spatial scale as the original image.

In each case, the expected width of the gaussian matches the fitted σ values, as can be seen in the plots.

We can also compare the 1-d PSD and 1-d ACF to these expected values, to ensure that the creation of the radial profile was accurate. Figure 5 shows the results of these comparisons, where once again the expected σ values are recovered from the 1-d profiles.

There is one case which is somewhat perplexing, although I think this is actually just related understanding what the 1-d PSD actually means. The 1-d PSD in frequency space matches expectations for this gaussian, but it is also possible to translate this profile from frequency space back to the original (or physical) pixel space. Panel a in Figure 6 illustrates how this is possible, by taking a slice through the center of the 2-d PSD and translating the width of the gaussian in pixel coordinates in the 2-d PSD image to the width of the gaussian in frequency coordinates – $\sigma_{psd(pix)} = \sqrt{n_x\, n_y}/(2\pi\sigma_{psd(freq)}$ (note that this is generally what we would expect from an FFT, with the addition of the $\sqrt{n_x\, n_y}$ term). If we then calculate the 1-d PSD and translate the frequency values into pixel coordinates using a similar formula – $x = \sqrt{n_x\, n_y}/(2\pi freq)$ – then we obtain the 1-d PSD in pixel / physical values, as is shown in panel b of Figure 6. The perplexing part of this is knowing what to expect for this 1-d PSD in pixel values; the result is no longer gaussian. Even trying to match a gaussian distribution with $\sigma_{psd(pix)}$ to this profile and compensating for the shift

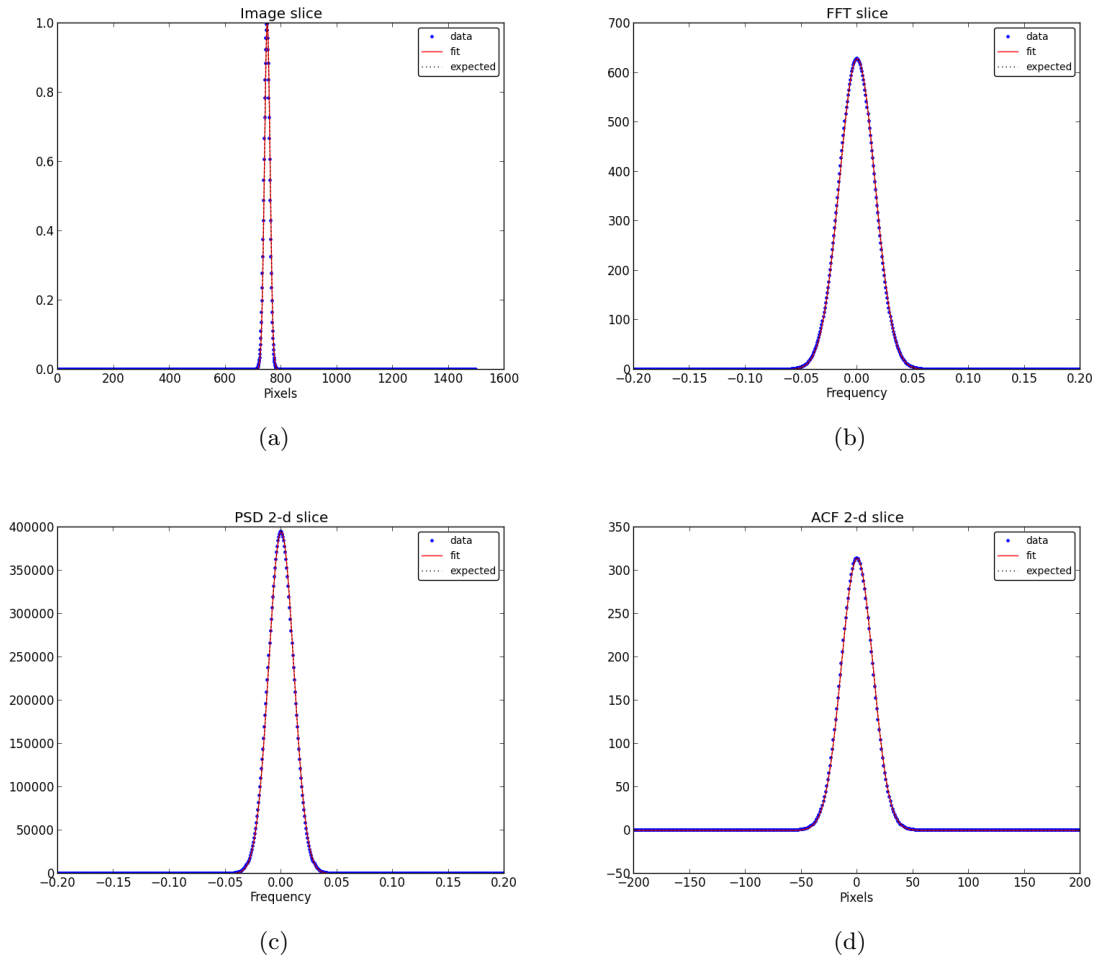
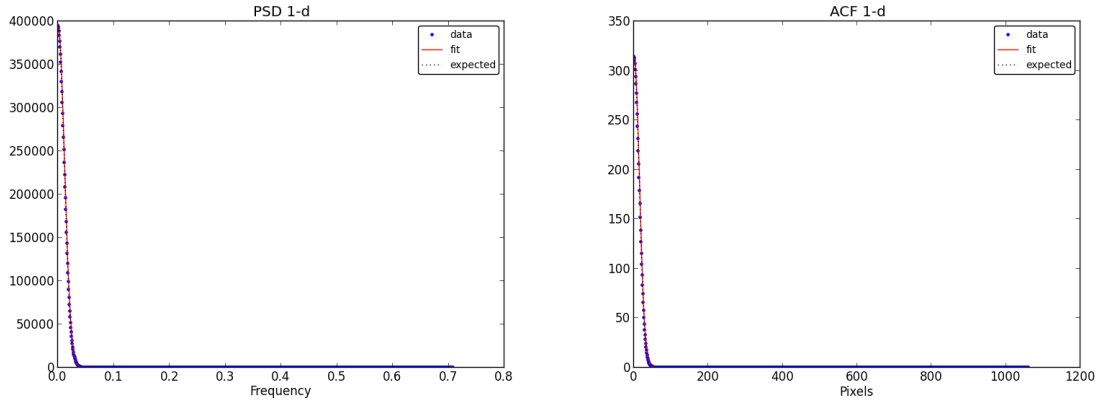


Fig. 4.—: Analyzing the forward transformations of a gaussian image. By taking slices through the center of the 2-d images representing the original image (panel a), the FFT (panel b), the PSD (panel c), and the ACF (panel d), the expected width of the gaussian as translated to the FFT/PSD/ACF can be compared with the actual calculated result. In each panel, the data (the calculated values) are represented by the blue dots, the analytical or expected result is plotted with the dotted black line, and a gaussian fit to the data points is shown with the red line. In all these cases, the fitted gaussian agrees very well with the expected values.



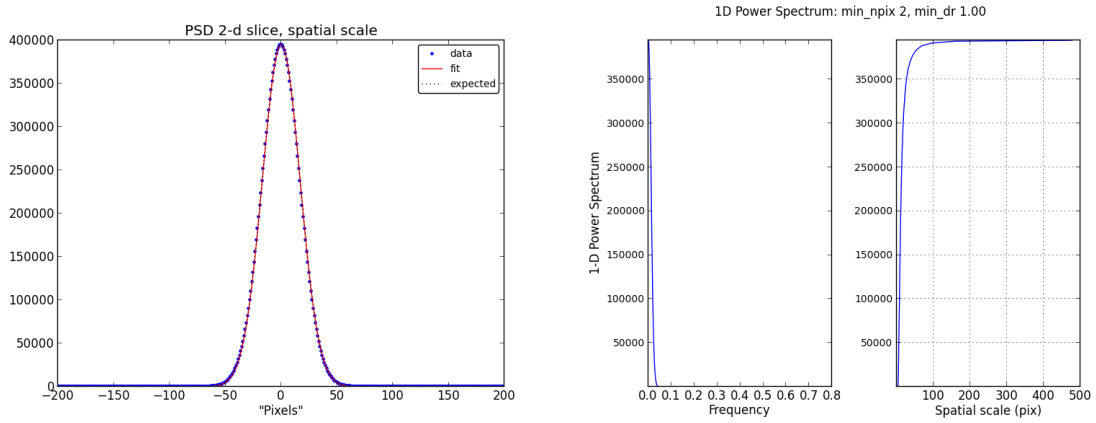
(a) (b)

Fig. 5.—: The 1-d PSD and ACF profiles for a gaussian image. These plots show a comparison of the calculated values of the 1-d PSD (panel a) and 1-d ACF (panel b) with the result expected from the gaussian distribution and the azimuthal symmetry of the 2-d cases of the PSD and ACF. The blue circles represent the calculated values, while the expected result is plotted as a dotted black line, while a gaussian fit to the data points is shown with the red line. This shows that the 1-d calculation preserves the expected radial profile.

from frequency to pixels doesn't match the data points. This raises the question of whether trying to interpret the 1-d PSD in pixel (or original/physical scale) coordinates is meaningful, whether any interpretation attached to the original spatial scales should be done using the 1-d ACF instead (which naturally exists in pixel scales and does match what intuition would suggest), or whether the problem is simply that 'intuition' isn't leading us to the right conclusions here and simply having more experience with 1-d PSD's in physical units would make this understandable. An example of the 1-d PSD in physical coordinates vs. the 1-d ACF in physical coordinates for two comparison input images with a much different but still high degree of structure is given in Figure 7; looking at these two comparisons illustrates the differences between what is represented in the 1-d PSD vs the 1-d ACF.

2. Inverting the FFT, PSD or ACF to reconstruct an image

After explaining the forward calculation of the FFT (real and imaginary), 2-d PSD and 2-d ACF, now let's look at the various possible steps of inverting this process to reconstruct an image. This process is also handled by the PImage class. The general idea is that the inverted (or reconstructed) version of the image (or FFT or PSD or ACF) is stored in the class with a suffix 'I' – thus the original image is 'image' while the image which is reconstructed as a result of inverting the FFT is called 'imageI'. The most complicated part of this inversion process is keeping track of what



(a) (b)

Fig. 6.—: An analysis of the physical scaling of the 1-d PSD. Panel a shows a slice through the 2-d PSD, where the x axis has been left in pixels (pixels in the 2-d PSD image), centered on 0. The gaussian fitted (red line) through the data points (blue circles) has a width $\sigma_{psd(pix)} = \sqrt{n_x\, n_y}/(2\pi\sigma_{psd(freq)})$. Panel b shows the calculated 1-d PSD, in both frequency (u) and equivalently-scaled pixel coordinates (*i.e.* $x = \sqrt{n_x\, n_y}/(2\pi freq)$ units.

you started with – obviously, starting with a 1-d ACF or 1-d PSD (vs. the 2-d PSD or ACF) and with or without the phase spectrum information makes a big difference to the final reconstructed image!

Inverting from the FFT (using both the real and imaginary components) to an image is simple using scipy.fftpack.ifft2, and recreates an almost perfect (within numerical error) image. Similarly, starting from the 2-d PSD or the 2-d ACF plus the phase spectrum results in an almost perfect reconstruction of the original image, including the intensity scales of the original image. No information is lost as long as the phase spectrum *and* the 2-d PSD or ACF is maintained. Note that if you start from the ACF, you must first reconstruct the PSD and then the FFT and then finally the image itself, while if you start from the PSD, you just reconstruct the FFT and then the image. Examples of these 'perfect' reconstructions are given in Figure 8.

At this point, we can consider how much information is lost by attempting to reconstruct the original image (a) without the phase information but with the 2-d PSD or ACF (keeping any azimuthal information from the PSD or ACF, but losing the location information from the phase spectrum), (b) with the phase information but using only a 1-d PSD or ACF (keeping the location information from the phase spectrum, but losing any azimuthal dependence from the PSD or ACF and replacing this with a radially symmetric function), or (c) without the phase information and using only a 1-d PSD or ACF. Examples of these reconstructions are given in Figure 9 and 10.

To some extent, the amount of information 'lost' will depend on the input image; images which

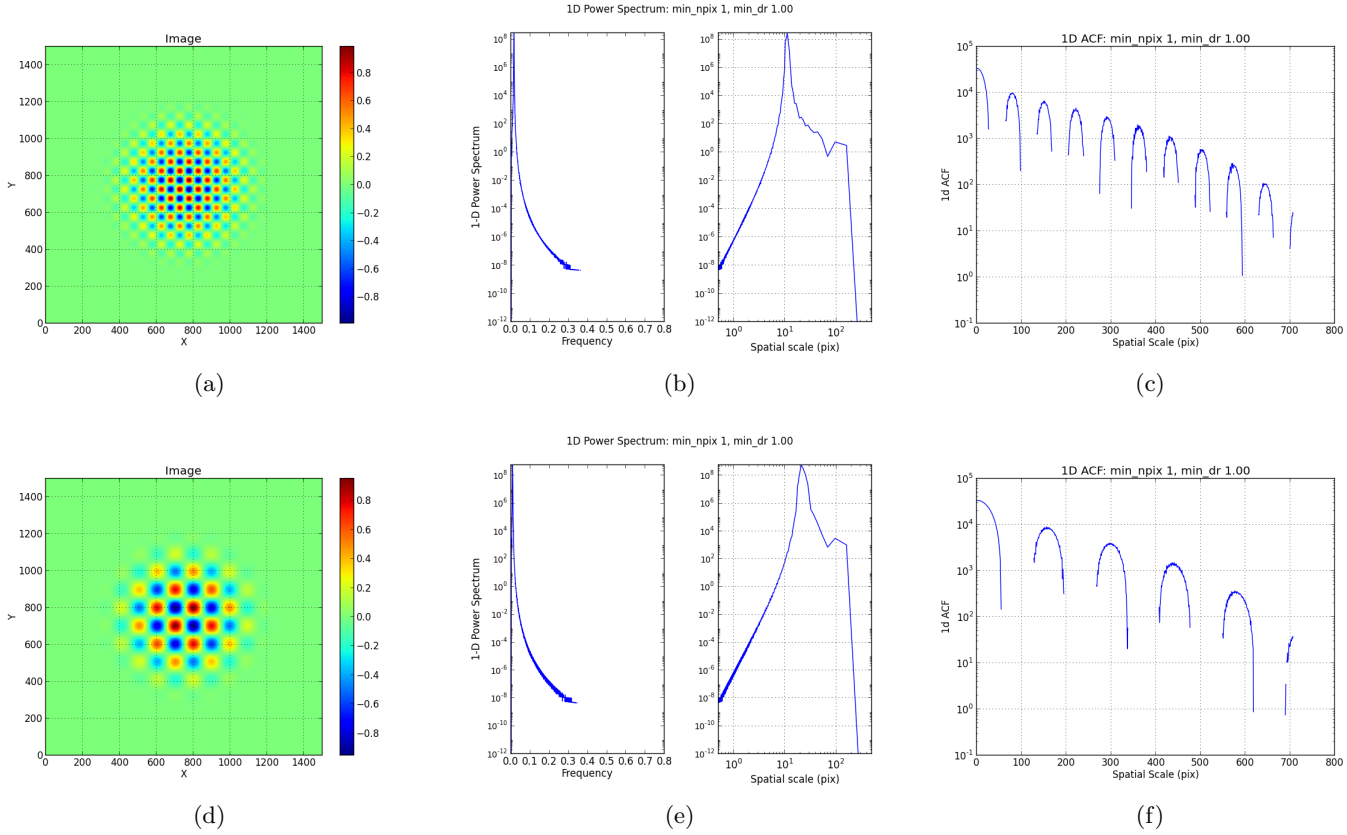
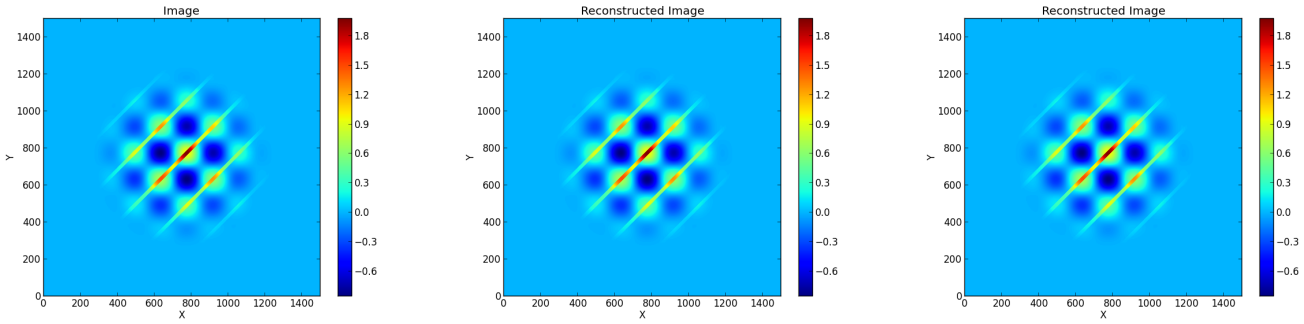


Fig. 7.—: A comparison of the 1-d PSD and 1-d ACF. The top row (panels a, b, c) show the transformation of the image (panel a) into the 1-d PDS (in both frequency and physical pixel space) (panel b), and onto the 1-d ACF (panel c). The bottom row (panels c, e, f) are similar, but start with the image in panel d, where the scale of the varying sinusoidal pattern has been doubled (the 'spots' are twice as big). The result of this scaling can be seen in both the 1-d PSD and 1-d ACF. In the 1-d PSD, the difference shows up only as a shift in the peak, by a factor of two, but the location of the peak is consistent with about half the size of the 'spots' in the original image, and does match the start of the rolloff in the smallest-scale correlations in the 1-d ACF. With the 1-d ACF, the peaks and valleys in the image show up as regions of correlated values, so we see peaks in the ACF at all of the regions we would naively expect from the image (*i.e.* within one spot and from each spot to each of the other spots at varying distances), and we see this pattern become twice as large in the lower row as in the top row.



(a) (b) (c)

Fig. 8.—: Reconstructing an image without losses. Reconstructing an image (panel a) from the 2-d PSD plus phase spectrum (panel b) and reconstructing the same image from the 2-d ACF plus phase spectrum (panel c). By retaining the 2-d PSD or ACF, plus the phase spectrum, the original image can be reconstructed to very high accuracy. A hanning filter has been applied to the original image here, although in practice these images could be reconstructed perfectly well without this filter. However, applying the hanning filter here allows for a better illustration of the recovery of the 1-d PSD and 1-d ACF in Figure 11 below, while still keeping the input images identical.

produce a nearly radially symmetrical 2-d PSD or ACF, for example, will not suffer as strongly when the 1-d PSD or ACF is used instead of the 2-d version. Losing the phase spectrum and replacing it with a uniformly distributed random sample of phases will generally spread the sources randomly around the image. In general starting from the 2-d PSD or ACF with a random phase spectrum yields similar results; however, starting from the 1-d ACF preserves structure at large scales better than starting from the 1-d PSD, while starting from the 1-d PSD preserves structure at small scales better than starting from the 1-d ACF. These are small differences, but are visible in the 1-d ACF or PSD calculated from the reconstructed images. This difference appears to be the result of how the 1-d PSD and 1-d ACF are calculated; because these are radial profiles, and there are only so many pixels near the very center of the image, the resolution of the 1-d ACF is lowest at the center of the 2-d ACF (equivalent to small spatial scales). Conversely, the resolution of the 1-d PSD is lowest at the center of the 2-d PSD, which is equivalent to the largest spatial scales.

In addition to evaluating the reconstructed image, we can examine the FFT, PSD or ACF resulting from calculating these values from the reconstructed image. In particular, comparing the final 1-d PSD and 1-d ACF is interesting as these are quantities that are often used as a summary measurement in astronomy. Figure 11 shows a comparison of the 1-d PSD and 1-d ACF calculated from the original image compared with these quantities calculated from images reconstructed using only the 1-d PSD or 1-d ACF with a random phase spectrum. As seen above in Figure 9 and 10, the reconstructed image in these cases does not look much like the original image. However, a 1-d

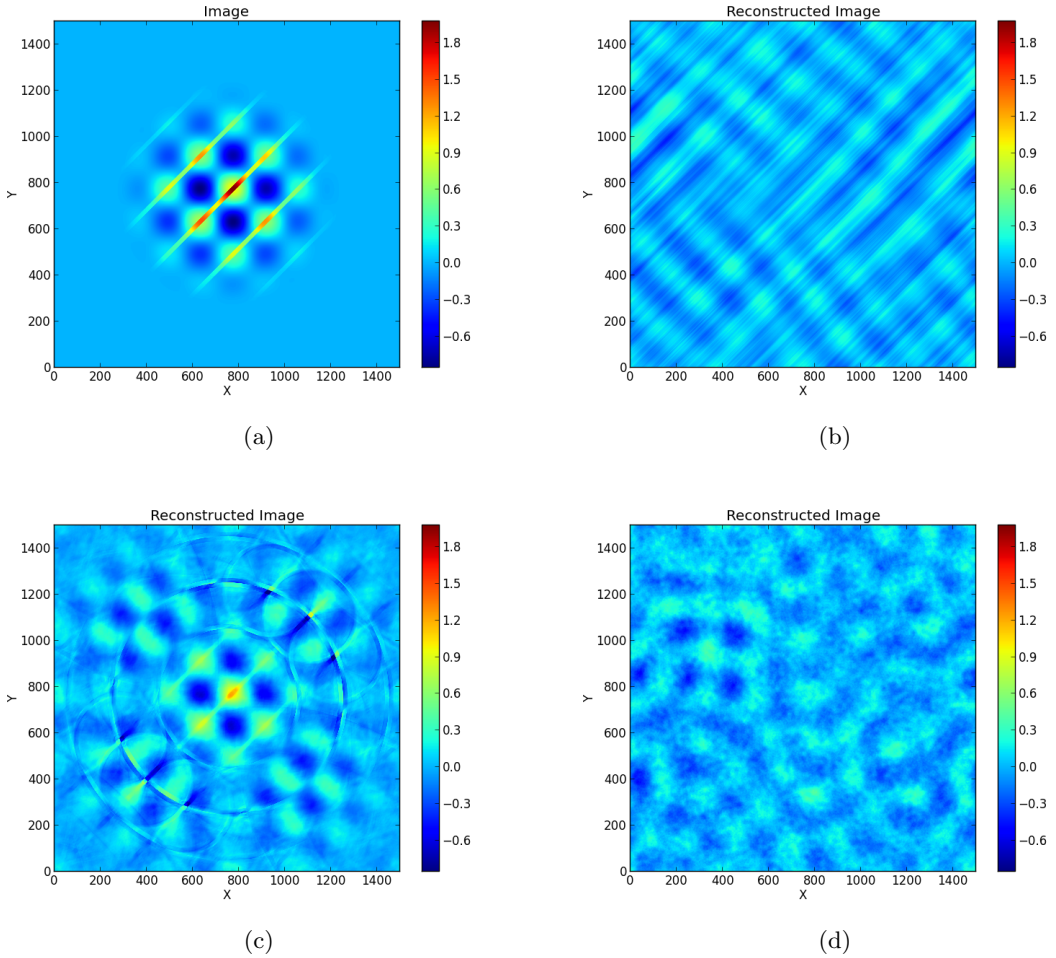


Fig. 9.—: Reconstructing an image after losing information, using the PSD. Panel a shows the original image. Panel b shows the image as reconstructed using the 2-d PSD but a uniform random distribution of phases in the phase spectrum. Panel c shows the image as reconstructed using the 1-d PSD but keeping the phase spectrum. The influence of the hanning filter is visible in the circular pattern seen in this reconstructed image – it seems that since the function of the hanning filter is known, this should be possible to remove during the reconstruction (perhaps when doing the inverse FFT to the image), but I did not investigate this possibility. Panel d shows the image as reconstructed using the 1-d PSD and a uniform random distribution of phases (the most information loss). In all cases, the generation of the random distribution of phases was started with the same random seed.

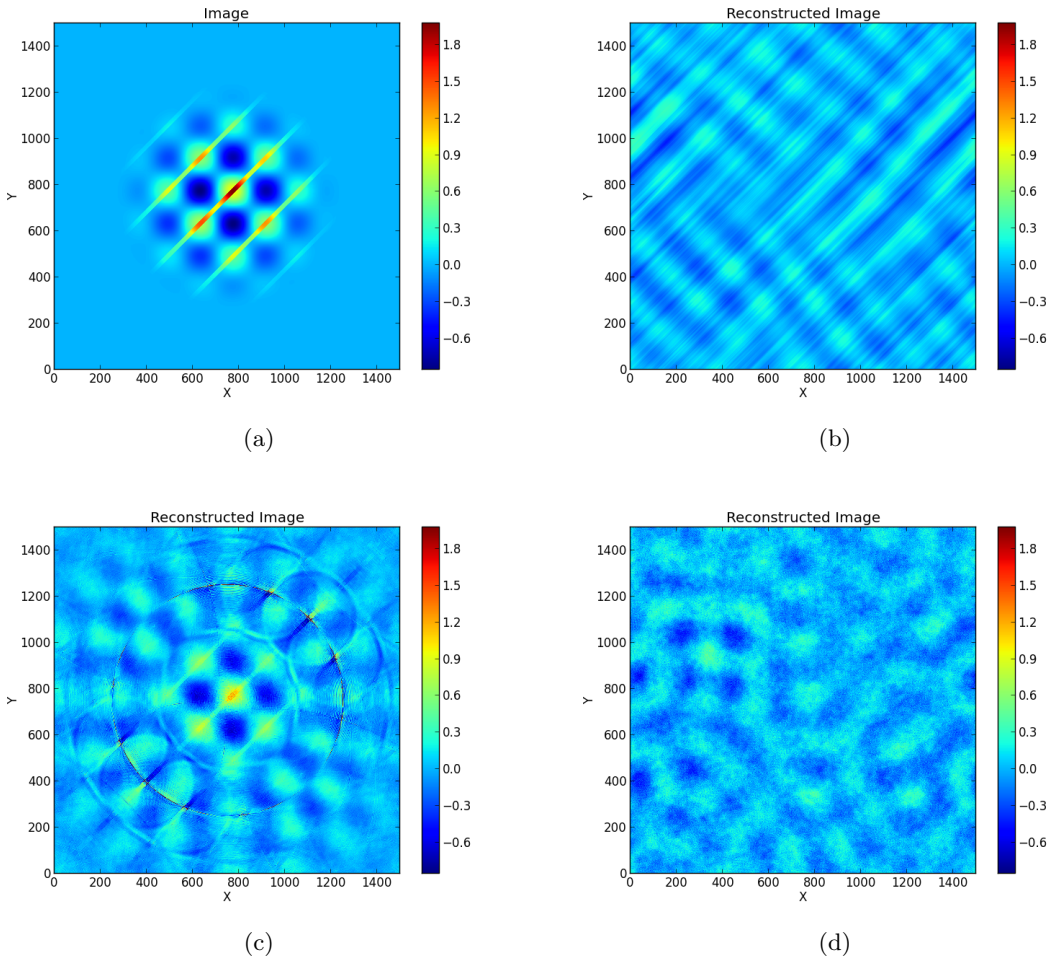


Fig. 10.—: Reconstructing an image after losing information, using the ACF (this is similar to Figure 9, but using the ACF instead of the PSD). Panel a shows the original image. Panel b shows the image as reconstructed using the 2-d ACF but a uniform random distribution of phases in the phase spectrum. Panel c shows the image as reconstructed using the 1-d ACF but keeping the phase spectrum. Panel d shows the image as reconstructed using the 1-d ACF and a uniform random distribution of phases (the most information loss). In all cases, the generation of the random distribution of phases was started with the same seed value. Comparing panel b of this figure to panel b of Figure 9 shows that when using the same random phase information, the reconstructed image is identical. However, comparing the bottom rows of this figure with Figure 9 shows how the 1-d PSD better preserves small spatial scale information while the 1-d ACF better preserves large spatial scale information (this is most noticeable in how panel d here is grainier than panel d in Figure 9, while panel c here better preserves the larger scale 'spots' when compared to panel c in Figure 9.

PSD or 1-d ACF calculated from these reconstructed images does still match the original 1-d PSD or 1-d ACF fairly well (although an image reconstructed from the 1-d PSD has a better final 1-d PSD match than the image reconstructed from the 1-d ACF, and vice versa).

3. More simple images

More simple images are available to play with using the TestImage class (testImage.py). This class gives the user the ability to add lines of varying widths, spacing and angle to the images, as in Figure 1, or to add a Gaussian of varying sigma, or to add random noise, sine functions, or elliptical shapes. An example of the latter is shown in Figure 12. These simple test images can be used to gain some intuition on what the FFT/PSD/ACF should look like, and the effects of inverting these values with and without the phase spectrum on the reconstructed image and/or using the 1-d PSD/ACF. A walk-through of this process (with comments in the code) is given in example.py.

4. Cloud Images

This brings us to analyzing clouds. The French calibration simulation group have provided code to generate clouds using two methods - one directly from a 1-d structure function ('old clouds') and one which is based on an image[2] ('new clouds'). Figure 13 shows what clouds generated with each of these methods looks like. The 'new' clouds end up being much more realistic looking, but we currently don't understand the physical scale related to each image. I've proceeded here under the assumption that the 'windowsize' specified by the cloud generation code is in degrees, thus setting a physical scale to the overall image (in this case, the image covers 4.0 degrees), and this seems to generally agree with the overall form of the structure function (see Figure 15).

Figure 14 shows the FFT/PSD/ACF of the old and new clouds (after applying a hanning filter). The old clouds have some unusual structure in the FFT and 2-d PSD which is not well understood. The new clouds have a much smoother 2-d structure in the FFT and PSD. The phase spectrum in both cases appears to be uniformly random, with no structure. A histogram of the pixel values is shown in Figure 16 and reinforces that the distribution appears to be uniform between $-\pi$ and π.

However, with the tools available and described in the earlier parts of this note, I decided to attempt again to construct cloud images from the 1-d PSD used for the 'old' style French clouds. I was able to determine the physical scale which seemed appropriate for the structure function (see Figure 17), and then translate this SF into a 1-d ACF. Although at this point I lost the information on the overall intensity scale appropriate for the 1-d ACF, since we eventually wish to scale the

[2]I believe this is a RASICAM image, from which has been extracted a 2-d PSD. It's unclear if the phase spectrum was preserved, but it does appear uniformly random.

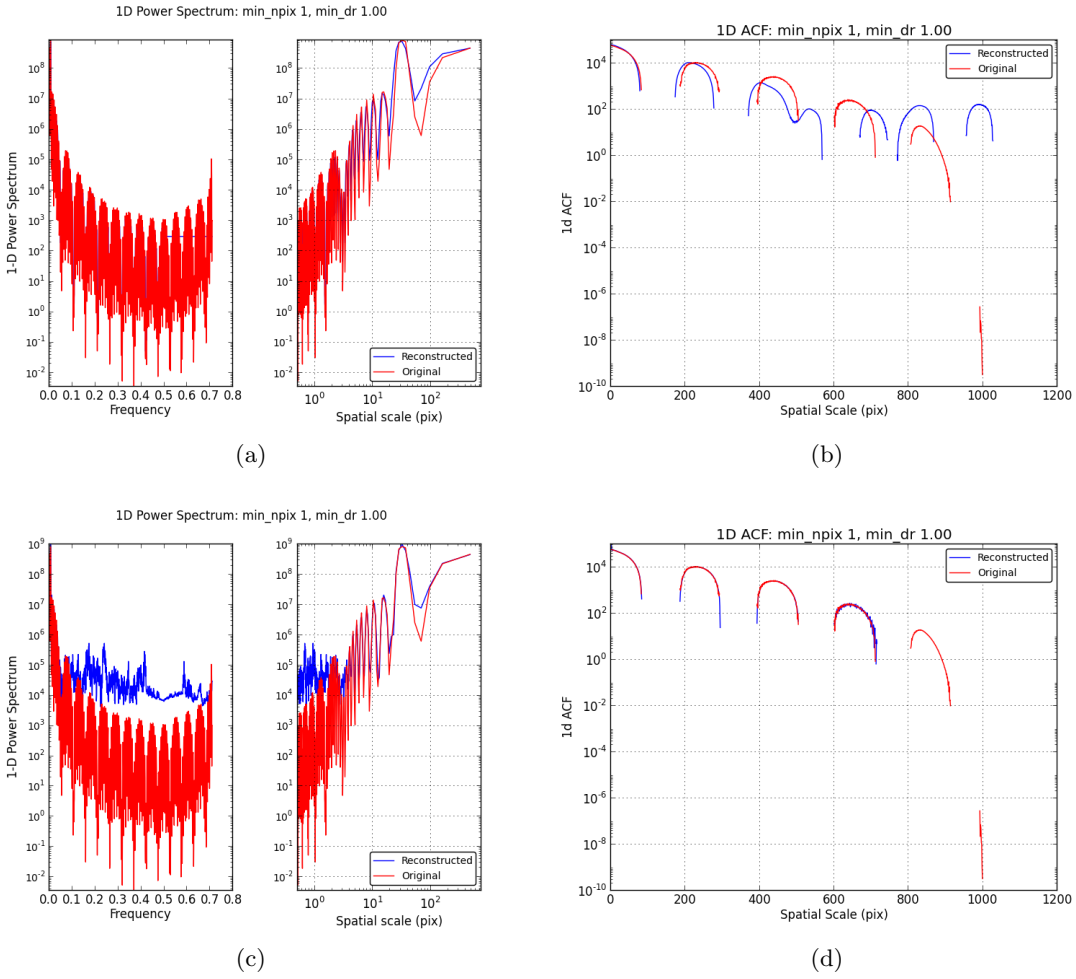
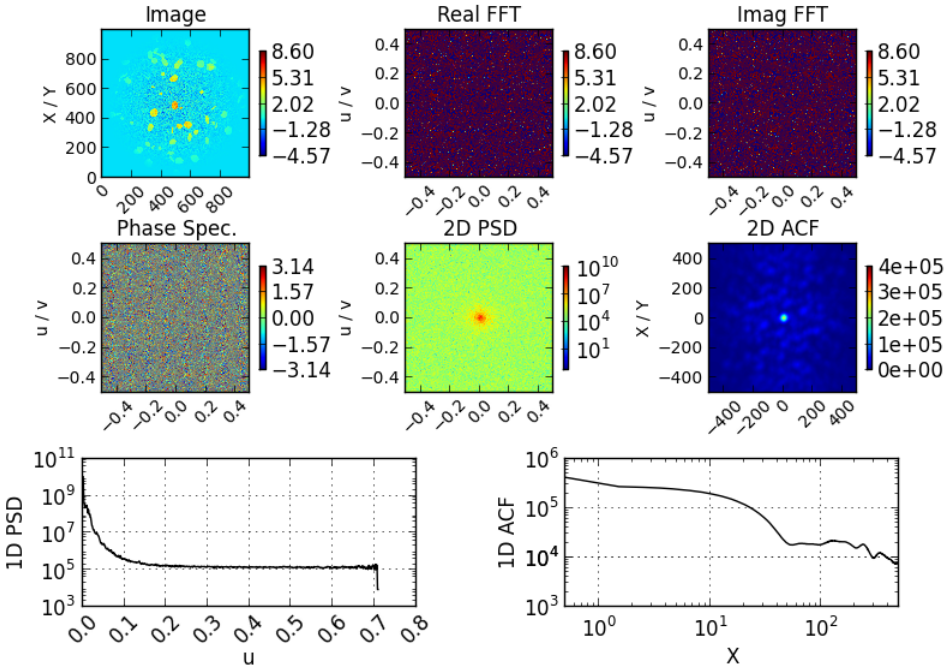
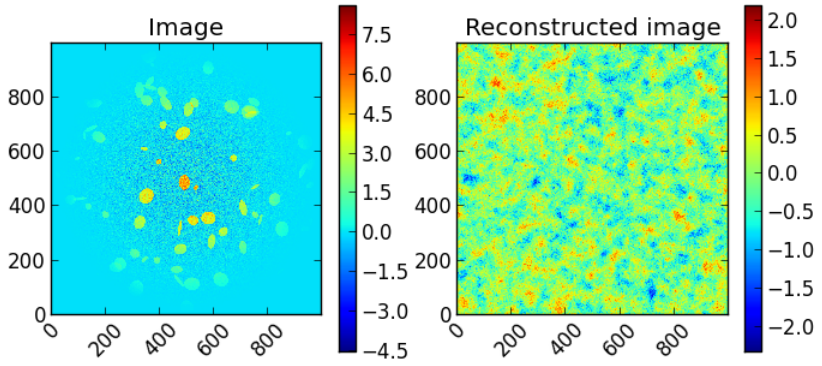


Fig. 11.—: Calculating the 1-d PSD and 1-d ACF from reconstructed images. These figures demonstrate the difference between the 1-d PSD and 1-d ACF calculated from the original image and the 1-d PSD and 1-d ACF calculated from the reconstructed images. In the top row, the image has been reconstructed from the 1-d PSD and a random phase spectrum, while in the bottom row, the image has been reconstructed from the 1-d ACF and a random phase spectrum (in both cases, the original image was the same as in Figures 8, 9, and 10). In each case, the 1-d PSD (or 1-d ACF) calculated from the reconstructed image is most accurate when the 1-d PSD (or 1-d ACF) was used to reconstruct the image.

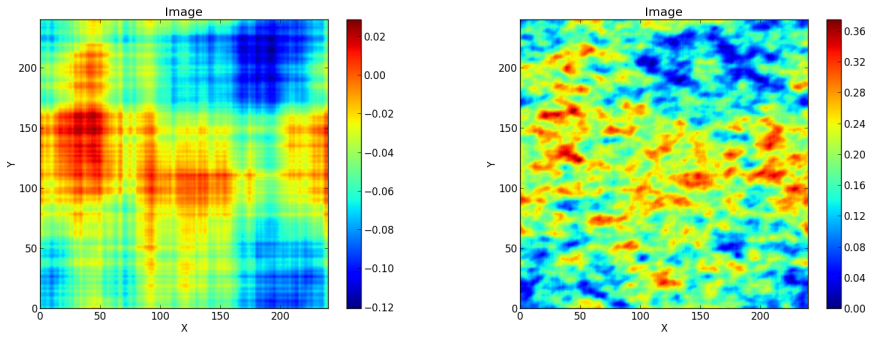


(a)



(b)

Fig. 12.—: An example of generating a random series of elliptical shapes, with random locations and angles, and adding gaussian white noise to the background. Panel a shows the FFT/PSD/phases/ACF and 1-d PSD/ACF calculated from the original image. Panel b shows the original image and the image reconstructed using only the 1-d ACF and a random phase spectrum.



(a) (b)

Fig. 13.—: Cloud images generated from the atmosphere_clouds package. The images are assumed to be 4 degrees across (resulting in a pixel scale of 60"/pix). When using the images to simulate cloud extinction, the intensity variation in these images is scaled by the overall cloud extinction. Panel a shows the 'old' style clouds generated from the SF only, using the code in the atmosphere_clouds package, while panel b shows the 'new' style clouds generated from a 2-d PSD generated from a RASICAM image (as well as the atmosphere_clouds package code).

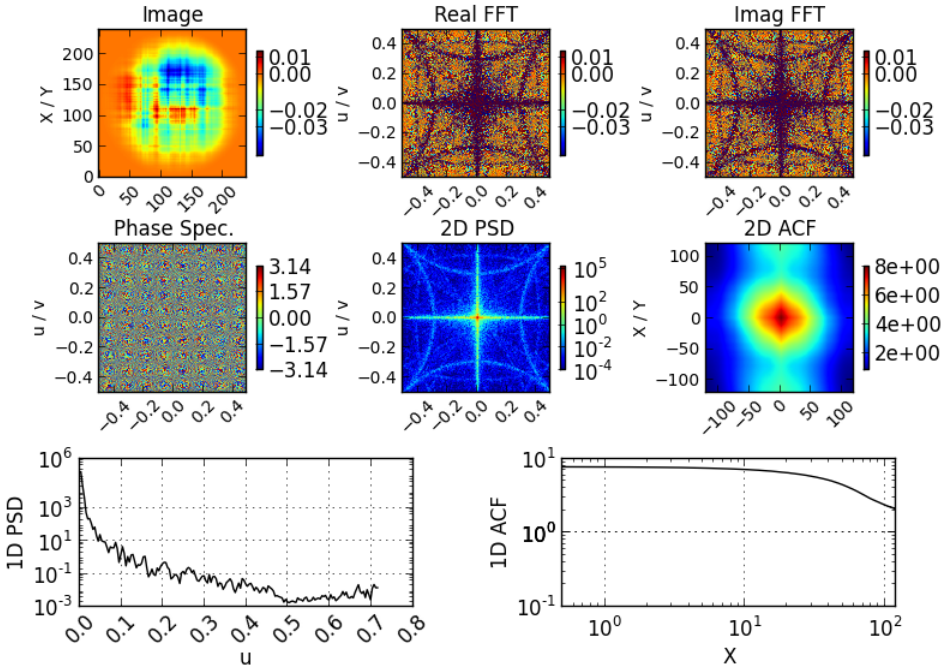
cloud extinction variations across the field of view by the overall cloud extinction, I believe this is inconsequential (I tested this by multiplying the 1-d ACF by a factor of 100 in the second cloud image I reconstructed; it did not appear to make a significant difference).

I added a uniform random phase spectrum, then reconstructed an image - a cloud extinction image, where I now know the pixel scale used to generate the image (and thus can place a physical scale on the resulting cloud image). I generated several reconstructed images, using a different random phase distribution each time, and find distinct 'cloud images' as a result. Some examples are shown in Figure 18. Calculating the structure function observed in each new cloud image, I compared this with a scaled version of the SDSS structure function, compensating for the fact that the SF calculated for the cloud images is scaled from 0-1. The resulting structure functions are shown in Figure 19, where it's clear that the final structures are close to but not identical with the original SDSS SF. The code to generate these clouds can be found in moreClouds.py.

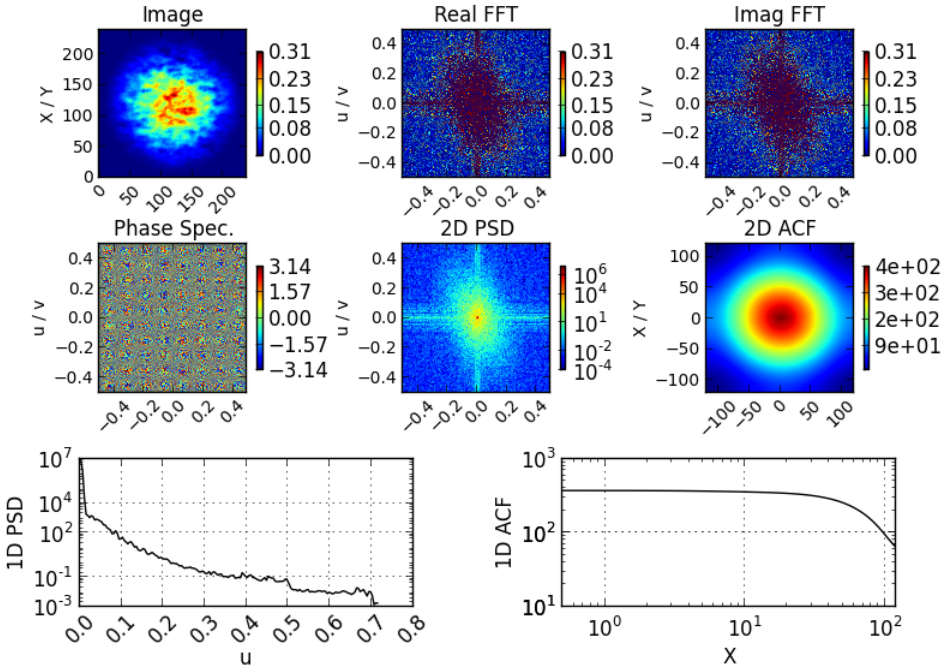
5. Outstanding Questions

1. Does it make sense to place a physical scale on the 1-d PSD, and is our lack of understanding here just related to a lack of intuition on what this should show? Is a 1-d ACF a more appropriate thing to use? (See discussion in text at 1.2).

2. Do these new reconstructed cloud images seem reasonable to use, given that almost anything can 'kind of look like a cloud' once a random phase spectrum is added? Is it significant that



(a)



(b)

Fig. 14.—: Cloud images and their FFT/PSD/ACF transforms, as generated from the atmosphere_clouds package. Panel a refers to the 'old' clouds, while panel b refers to the 'new' clouds. The old clouds show structure in the FFT and 2-d PSD which we do not understand.

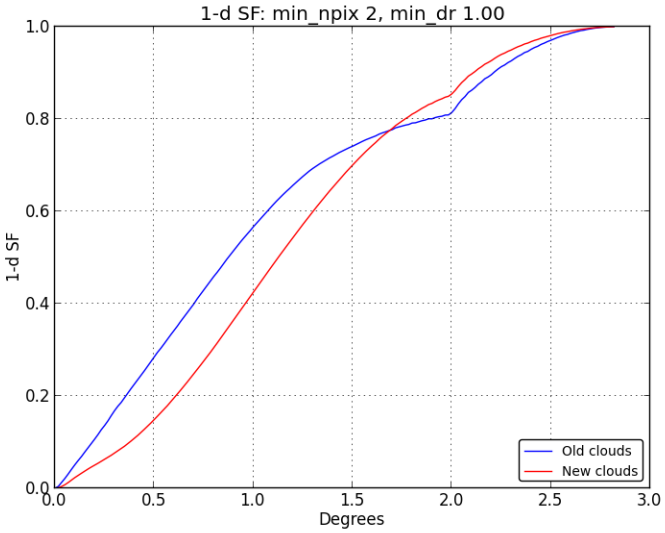


Fig. 15.—: Structure function from the 'old' and 'new' clouds generated from the atmosphere_clouds package. Note that these SF's include a hanning filter on the input cloud images, which influences the final result somewhat.

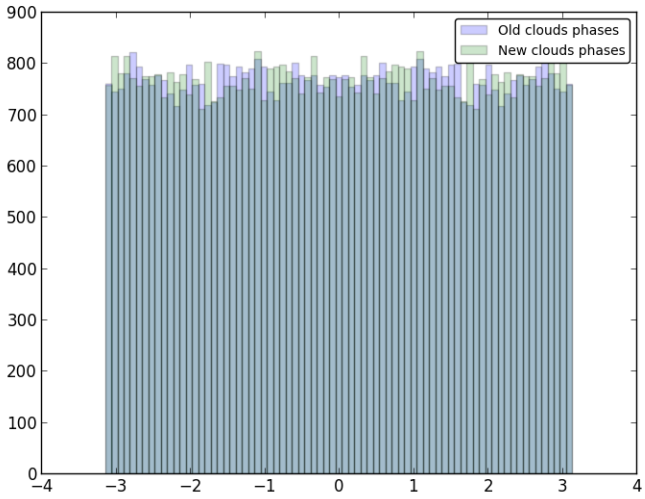


Fig. 16.—: Histogram of the phase spectrum, from both the old and new clouds in the atmosphere_clouds package.

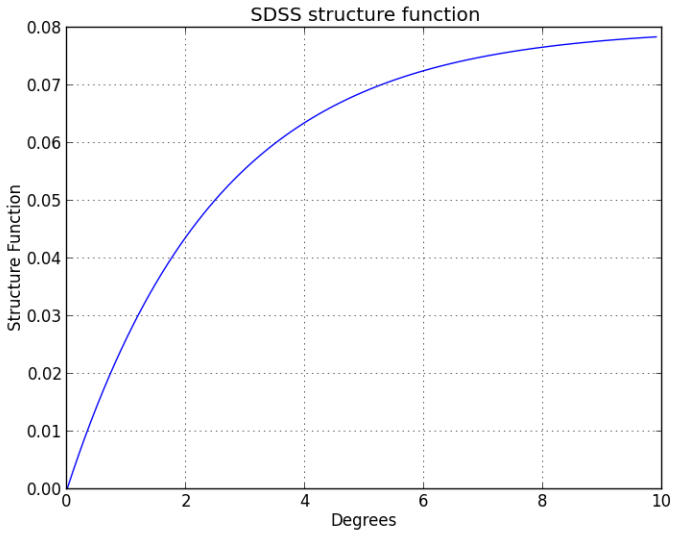
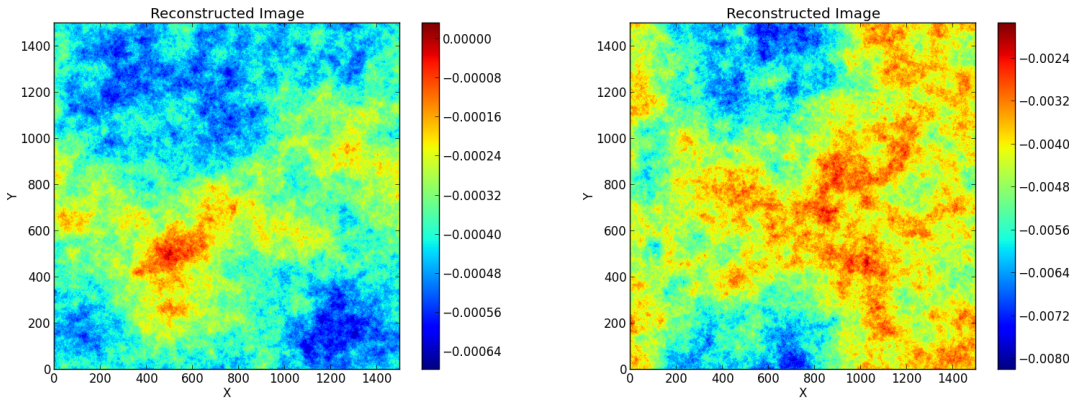


Fig. 17.—: Input structure function based on basic structure function of 'old' atmosphere_clouds code, which seems to be from the SDSS SF.



(a) (b)

Fig. 18.—: Cloud images generated from the SDSS structure function implemented in the old style clouds above, but with an image reconstructed using the code in PImages. Panels a and b show two different realizations of clouds, where the structure function remains the same but the phase spectrum is randomly drawn each time. These images are also 4 degrees across, I understand how the physical scale of these images translates all the way from the SF, and here the resulting pixel scale is 0.00266 degrees per pixel (9.6 ″/ pix) (and is easily choosable by the user).

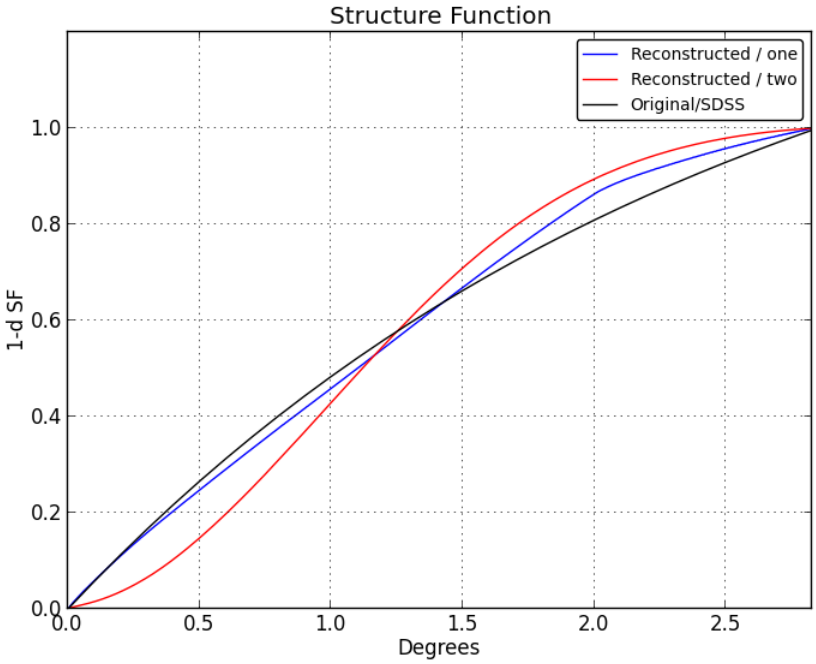


Fig. 19.—: Structure function from SDSS, compared with SFs calculated from reconstructed cloud images in Figure 18. The results are close to the original SF, but not entirely the same. Some of this is due to adding a hanning filter in calculating the reconstructed image SF (which was done for the Reconstructed/two line but not for Reconstructed/1), but it is not identical even with this filter removed.

the SF is not exactly recreated? Can we obtain some cloud extinction data to evaluate the phase spectrum of real clouds?

3. How can one go gracefully go from doing regular FFTs on a tangent plane-projection to using full spherical harmonics on the sky (for the purposes of looking at photometric residuals in self-calibration, etc.)?