SKA School 2014: Fourier Transforms Lab

In this lab we will use the `numpy.fft` module (which implements the Fast Fourier Transform algorithm) to perform a number of exercises to show the properties of the Fourier transform, and see how and why moving from the time (or spatial) domain to the frequency domain is useful. As you will see on subsequent days, the Fourier transform plays an essential in radio astronomy. Note that it saves a bit of typing to use `import numpy.fft as fft` to include the FFT routines in your programs.

Fourier transforms of 1d functions

We'll begin by looking at how to perform a Fourier transform in python. An example code is provided here:

http://www.acru.ukzn.ac.za/~ska2014/materials/fourierlab/1d/exampleFFTs1D_template.py

Run the script to see what it does, and load the script into a text editor to see how it works. You should see that it performs the FFT (and inverse FFT) of a delta function and a Gaussian; two functions shown in the lecture this morning. All of the action takes place in the `makeFFTPlot` routine – there you can see that `fft.fft` is used to do the Fourier transform, and `fft.ifft` performs the inverse Fourier transform. Note that when plotting the Fourier transformed image, the `fft.fftshift` routine needs to be used to put the frequency components where we expect to see them on the plot (this is just convention; without the shift `numpy` puts the zero frequency component at the edges rather than in the centre).

Modify the code to add a function that generates a boxcar signal, and feed that signal into the `makeFFTPlot` routine. If done correctly, the Fourier transform of the boxcar is a sinc function. Perhaps experiment with taking Fourier transforms of other signals, and/or change the widths of the boxcar or Gaussian signals...

Convolution

Now let's take an image and convolve it with a Gaussian kernel to blur it out. Start with this script:

http://www.acru.ukzn.ac.za/~ska2014/materials/fourierlab/convolution/convolution_template.py

You'll also need to download the image `MeerKAT.jpg` which is in the same directory on the webserver.

The template script loads in the colour MeerKAT image, converts it to grayscale, and then applies the `ndimage.gaussian_filter` routine to blur the image. This is the easy way to blur an image in python, but your task is to modify the script to do the convolution using `fft.fft2` and `fft.ifft2` instead (the '2' here indicates these routines work in 2d rather than 1d). The code already generates the Gaussian kernel you need to do this (since this is a little tricky) – you just need to work out what to do with the arrays `kernel2d` and `gray` to make the blurred image. You can also experiment with changing the kernel size: you will probably find that for very large kernels (`kernelSize = 80` or higher) that your FFT method is faster than `ndimage.gaussian_filter`.

Fourier transforms of images

Now let's look at magnitudes and phases. Download the script

http://www.acru.ukzn.ac.za/~ska2014/materials/fourierlab/magsAndPhases/magsAndPhases.py

Running this, you should see it produces three plots: one of the input image (a circle or a box), one of the magnitudes (this shows the power on different spatial scales in the image), and one of the phases. Experiment with commenting in/out the routines that makes the input array `arr`, and change the parameters (location, radius or box dimensions, rotation) to see their effect on the output magnitudes and phases after Fourier transforming the image.

Swapping phases and magnitudes

If we look at an image of a face, do the magnitudes or phases contain the most information? Let's check this by taking two images of faces, Fourier transform them, swap their magnitudes or phases, and then inverse Fourier transform them. Download the script and images from here:

http://www.acru.ukzn.ac.za/~ska2014/materials/fourierlab/phaseSwap/

The script `phaseSwap_template.py` already contains routines to load and save the colour images from the `images/` directory as gray scale (look in the previous `magsAndPhases.py` script for how to use these). Your task is to write the code that calculates the magnitudes and phases, swaps the phases between the images, and saves the results as images (there are some comments in the template script to guide you). Once you have done this, swap the magnitudes instead of the phases and compare. What do you conclude?

Filtering

Converting an image from real space to frequency space makes it easy to do low-pass or high-pass filtering. In low-pass filtering, we keep only the longest wavelengths (lowest frequencies), which correspond to large-scale features in an image. In high-pass filtering, we keep only the shortest wavelengths (highest frequencies), i.e., the small scale features of an image.

Download the script:

http://www.acru.ukzn.ac.za/~ska2014/materials/fourierlab/filtering/filtering_template.py

and the associated images (these are the same as the ones from the previous phase-swap exercise, but you could a different .jpg or .png image if you like). If you run the script, you should see that it produces a low-pass filtered image by masking out the magnitudes corresponding to the smallest scales. Your task is to adapt the script to make a high-pass filtered image (you can do this immediately below the code that is there in the script). Once you have done this, experiment with changing the filter size (change `maskRadius`) and see the effect this has on the image. You will see some artefacts in the images – what causes this? If you have time, investigate how to mitigate this.