Chapter 1

License

Chapter 2

Code Manifesto

This code (theta.py) is designed to take images of diffraction data and, using the geometrical conventions of the 'LP-Diffract' code written by Andrew Higginbotham, map the images into theta-phi space.

The code is also capable of compensating for sources of intensity attenuation as a function of gsqr and phi. The sources of attenuation considered are:

- Thomson polarisation factor
- filters placed in front of imaging media
- different path lengths of rays through the diffracted sample

These intensity corrections allow the output to be used for intensity sensitive calculations, such as temperature measurement.

Chapter 3

Setup and Metadata

3.1 Libraries Used

- Pillow 3.3.1
- numpy 1.11.1
- math
- matplotlib 1.5.3

Chapter 4

Code Structure Overview

The code is composed of three different types of files. These are:

- input file
- spine file
- function files

The code works by first running the spine file 'theta.py'. The spine file then reads in the input file 'in_theta.py' as python variables. The spine file then sequentially feeds these variables into the function files in the correct order. In other words, the spine file links all of the functions together, and brings the process from beginning to end.

From a users perspective, the only file that requires editing is the 'in_theta.py' input file. If the user wishes to modify the code, they are welcome to do so.

Chapter 5

Methods

5.1 Overview

5.1.1 Inputs

Instructions to the code are provided through the "in_theta.py" file. Each of these values is read into the code as a python variable. The parameters are:

- **image_filename** - a string containing the name of the image file to be read in. The image file should have single values for intensity at each point; RGB values are not supported. For the purposes of accurate intensity measurement, the code assumes the intensity values are given in PSL.
- **source_position**

5.1.2 Outputs

5.1.3 Method

The code has several distinct sections: calculate some common results needed for later functions; calculate $|G^2|$, phi, angles relevant to intensity corrections for each pixel in the image; apply intensity corrections due to filter attenuation and thomson polarisation; group pixels into bins of theta and phi; make plots, including integrated intensity across phi. The first section of code flows like this:

- Find the central pixel in the image.
- Determine the width and height of each pixel in mm.
- Renormalise the vectors view_x and view_y (the vectors which translate movement in x and y of the image into the 'canonical' vector space). This renormalisation exists because sonOfHoward will sometimes give slightly wrong normalised vectors (eg. the vector (-0.01, 1.00, 0.00) might be given by sonOfHoward).

- Convert the 'offset' input variable to mm. The offset input variable tells us how many view_x and view_y we should move to get the image plate in the correct position.
- Calculate the adjustment required to move the position vector of each pixel from the top-left corner to the centre of the pixel.
- Create a vector from the origin (i.e. target position) to the centre of the central pixel, using the above adjustments for offset. By now, all inputs from sonOfHoward have been converted to mm.
- Create a vector and unit vector from the source position to the origin. This is done by taking the negative of the input source position.
- Define phi = 0 degrees. This is done by taking the normal of the plane containing the vector (0, 0, 1) and the vector from the origin to the centre of the image.

The code has now formulated all of the components required to calculate $|G^2|$ and phi for each pixel. To calculate $|G^2|$ for each pixel:

- Calculate the vector and unit vector between the origin and the current pixel.
- Using the unit vector from origin to current pixel (u), and the unit vector from source to origin (v), calculate G by the equation $G = \frac{u-v}{\lambda}$, where λ is the wavelength of the x-rays given in the input file 'in_theta.py'.
- Convert G into units of Å^{-1}, by multiplying by the lattice constant of the probed material.
- Calculate the length of G to get $|G|$, then square the result to get $|G^2|$.

To calculate phi for each pixel:

- Calculate the normal to the plane containing the vector (0, 0, 1) and the vector between the origin and the current pixel.
- Calculate the angle between the normal of the plane, and the normal of the plane previously defined to represent phi = 0. If the cross product of these vectors (in the same respective order) points in the (0, 0, 1) direction, the angle phi is said to be negative, and vice versa.

While looking at each pixel in an image, the code will also calculate the angles needed to compensate for filters and polarisation. For the filter correction, the angle α between the normal of the image plane and the vector connecting the origin to the current pixel is required. The filter correction proceeds like so:

- The attenuation length ϵ is defined such that $I = I_0 e^{-\frac{x}{\epsilon}}$. From this, the attenuation factor is defined as $e^{-\frac{x}{\epsilon}}$.

- x is calculated by taking the thickness of the filter, and dividing by $cos(\alpha)$, to give an effective thickness.
- The inverse of the attenuation factor is multiplied by the intensity of each pixel so as to reproduce an approximation of the intensity, had the filters been absent.

After the filter correction is applied, the correction due to thomson polarisation must also be included.

- The angle ψ, between the vector connecting the source to the origin and the vector connecting the origin to the current pixel, is calculated.
- The correction to intensity due to this polarisation effect is given by $\frac{1+cos^2(\psi)}{2}$. The intensity of each pixel is multiplied by the inverse of this value to eliminate this effect.

Once corrections to the intensity have been implemented, the pixels are then binned according to their $|G^2|$ and phi values. Once binned, a theta-phi image (technically, a G^2-phi image) is constructed. The intensities are then integrated over phi.

5.2 Detailed Structure of Functions

work_out_common_results - This function works out results that don't need to be recalculated for each pixel of the image, but are necessary to calculations performed on each pixel. It's flow is: find image central pixel, calculate mm per pixel in both width and height, renormalise view_x and view_y numbers from LP-diffract, convert the 'offset' input variable into mm, calculate the adjustment needed for a vector to point at the centre of a pixel rather than the top-left corner, calculate the vector connecting the origin to the pixel in the centre of the image, calculate the vector between the x-ray source and the origin, convert the soruce-origin vector to a unit vector.

compensate_for_filters -

Chapter 6

Code tests

In order to validate the code, some test features are included. They can be run using the bash command "pytest" in the main directory. Note that the library 'pytest' must be installed for this to work. Once "pytest" has been called, a number of tests will be undertaken in order to verify that the code is operating as expected. Note also that the variable "test_mode" must be set to "True", like this: "test_mode = True".

6.1 Unit Tests

6.2 Full Code Test

This test will make use the code as if it were being used by a normal user. By setting the variable "test_mode = True", the code will use the file "test_in_theta.py" as the set of input variables, rather then the usual "in_theta.py" file.

The test case is set up as described hence. There are two images of 2x3 pixels (image width is 2 and height is 3). The pixel values are set to 1.0. These images have the following LP-Diffract settings:

	Image 1	Image 2
Source	100, 0, 100	100, 0, 100
Normal	0, -20, 20	20, 0, -20
Offset	0, 0, 0	5, -2, 0
Width	20	20
Height	30	30
View_x	-1.00, 0.00, 0.00	0.71, 0.00, 0.71
View_y	0.00, 0.71, 0.71	-0.00, 1.00, 0.00

Other input parameters are: wavelength = 1.378, filter thickness = 10.0 and 6.0 for each image, filter attenuation length = 34.1 and 109.7 for each image, "phi_0_definer = [0.0, 0.0, 1.0]", "phi_limit = [-180.0, 180.0]", "gsqr_limit = [0.0,

100.0]", "num_gsqr_bins = 10", "num_phi_bins = 10", "num_width_subpixels = 2", "num_height_subpixels = 3".