py321DIAS: Data Intensity Analysis Software for 2D Powder Diffraction Data

A Thesis Presented

by

Brandon Rhymer

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Master of Science

in

Geosciences

Stony Brook University

May 2016

Stony Brook University

The Graduate School

Brandon Rhymer

We, the thesis committee for the above candidate for the

Master of Science degree, hereby recommend

acceptance of this thesis.

Dr. Lars Ehm – Thesis Advisor
Research Associate Professor, Mineral Physics Institute

Dr. John Parise – Chairperson of Defense
Distinguished Professor, Department of Geosciences

Dr. Sanjit Ghose - Third Reader
Physicist, Brookhaven National Laboratory

This thesis is accepted by the Graduate School

Charles Taber
Dean of the Graduate School

Abstract of the Thesis

py321DIAS: Data Intensity Analysis Software for 2D Powder Diffraction Data

by

Brandon Rhymer

Master of Science

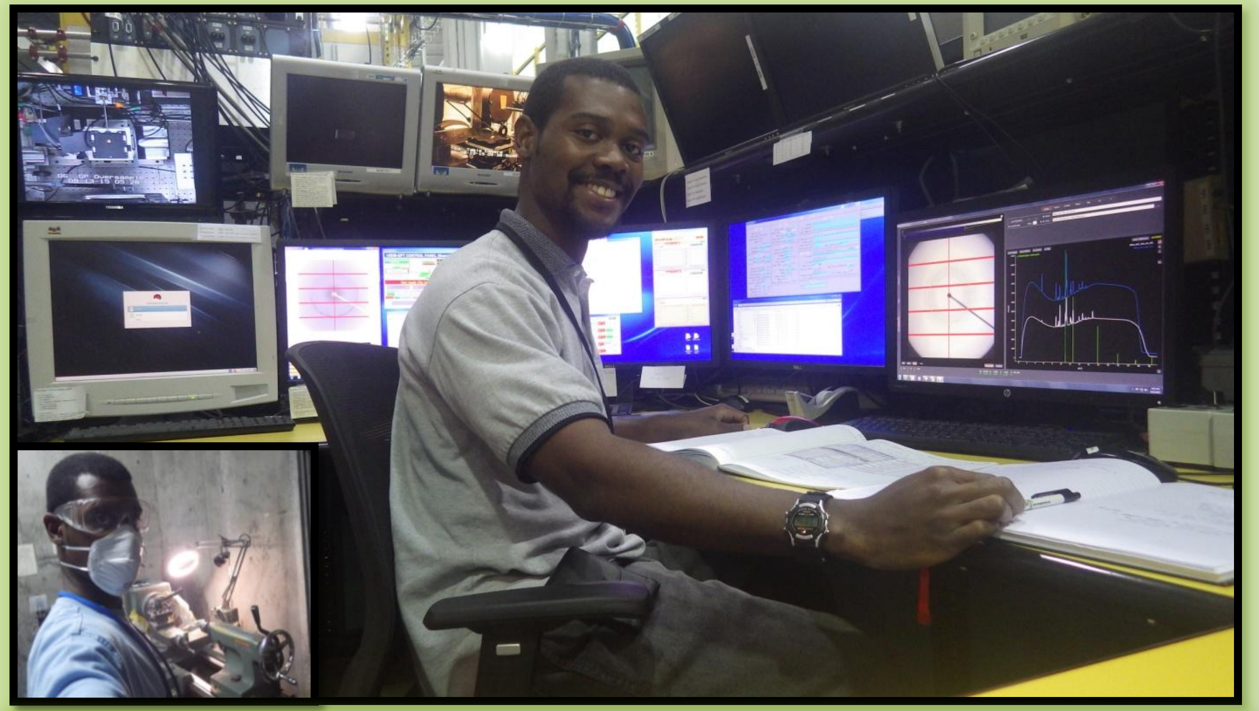
in

Geosciences

Stony Brook University

2016

Obtaining correct intensities and their uncertainties are imperative for acquiring meaningful structural parameters for high-pressure X-ray powder diffraction data that typically have intrinsically low peak-to-background ratios of measured intensities. This is important for full structural determination techniques. Commonly used software packages like Fit2D and pyFAI are useful for reducing 2D X-ray powder diffraction (XPD) images into 1D diffraction plots. However, the default weighting scheme that programs use for fitting diffraction data normally causes unreasonable goodness of fit values, which causes statistical uncertainties to seldom be determined. Hence, we created an integrated open source data analysis software that builds on pyFAI, which was possible because it uses the relatively easy to use open source Python programming language. Our program can covert 2D X-ray powder diffraction images taken from 2D detectors into 1D diffraction data using azimuthal integration with a routine that performs a statistical bin analysis of the collected diffraction intensities. The statistical analysis will be used to provide reliable intensities with calculated intensity errors and any contributions to the diffraction pattern not coming from the sample will be automatically filtered out utilizing this analysis. Due to this, the program can provide meaningful intensity error estimates for high-pressure synchrotron XPD data, which can then be utilized for performing an in-depth structural analysis using Rietveld refinement. This will allow for more precise structure determination and give way for the development of robust structure-property relationships in materials at extreme conditions.



SUMITOMO
HEAVY INDUSTRIES, LTD.
JAPAN

Table of Contents

Chapters

List of Figures/Tables/Illustrations

List of Figures

List of Abbreviations

py321DIAS - Python: 3D sample, 2D diffraction image, 1D diffraction plot, Data Intensity Analysis Software

pyFAI - Python Fast Azimuthal Integration

HP - High Pressure

XPD - X-ray Powder Diffraction

SXPD - Synchrotron X-ray Powder Diffraction

DAC - Diamond Anvil Cell

GOF - Goodness Of Fit

PONI - Point Of Normal Incidence

Acknowledgments

It is my great pleasure to extend my sincerest gratitude to my advisor, Dr. Lars Ehm, for your expert guidance, encouragement, assistance, steadfast support and most gracious patience. I would like to thank Melissa Sims for being a great student mentor, friend and support. I'd like to thank Ibrahim Hammould, Sumesh Balan, Ashok Sankar, and Clemens for their time, generosity, willingness to help and invaluable programming aid. I'd like to also thank my officemate, Mohammad Mahdi Davari Esfahani, and all my colleagues in Professors Lars Ehm and John Parise's research group for their valuable and insightful feedback, suggestions and collegial support. This research was only possible due to financial assistance from the National Science Foundation.

Moreover, I'd like to thank Dr. Gabriel Gwanmesia, Dr. Lars Ehm and Dr. Robert Lieberman for creating this exceptional geosciences program for African-American students and for inspiring me to be a part of it. I thank you all for sharing with me a part of your invaluable wisdom and for enlightening me with your love and zeal for the geosciences. You were all extraordinary and special mentors in your own unique ways. Thanks is but a word that cannot fully describe my deepest appreciation for all you'll have done.

Finally, to my family and beloved people who are dearest to my heart, thanks for your support, love and everything that you have done for me. I would like to clarify that friends are more than just friends to me, they are like family. May God bless you all now and always.

Chapter 1: Introduction

1.1 General Introduction

Synchrotron X-ray powder diffraction (SXPD) is a valuable non-destructive experimental technique that is utilized for studying the atomic structure of polycrystalline materials. However, SXPD data acquired from high pressure (HP) experiments produce intrinsically low intensity peak-to-background ratios that make structural data analysis more complex. The diamond anvil cell (DAC) is the most commonly used high pressure device for HP-SXPD research because it is the most experimentally versatile. It can generate extreme pressure and temperature conditions above 600 GPa and 600 K (Dubrovinsky 2012, Heinz 1991), which allows researchers to investigate materials that can inform us about the Earth's core and the interior of other planets. These types of experiments have many other applications ranging from nuclear to material science. Therefore, better understanding the atomic structure and overall properties of these materials are impetrative for discovering new materials and to determine how to use them effectively for future scientific advancement. However, intensity errors that are due to the sample, detector and experimental setup are problematic for obtaining a valuable data analysis. The diamond anvil cell, X-ray source and specimen can all contribute to low peak-to-background ratios. Single crystal reflections due to the diamond, dead pixels due to the detector, beam stop shadow due to the experimental setup, and poor powder statistics due to the sample are all contributors to the integrated intensities and their error uncertainties within diffraction patterns. Hence, a software program must be able to properly account for these errors and remove contributions to the diffraction signal that are not from the sample and then determine accurate errors for the diffraction signal.

1.2 History of Synchrotrons

Synchrotron radiation was first observed at the General Electric Research Laboratory in Schenectady, New York in 1947, which was 52 years after the 1895 discovery of X-rays by the German physicist Wilhelm Konrad Röntgen, who was awarded the first Nobel Prize in Physics in 1901. Due to this observation and its promising usage, there was an increased interest for creating and operating more advance synchrotron facilities (Robinson 2015).

While the popularity of synchrotron facilities grew, the next major advancement was the development of electron storage rings, which are the basis for all synchrotron sources to present day. In the 1950s, the Midwest Universities Research Association (MURA) was established to produce a proposal for a high-current accelerator for particle physics and as a part of this project, Fred Mills and Ednor Rowe designed a 240 MeV storage ring (Rowe 1973) as a platform for advanced accelerator studies. In 1965, there was an enormous push towards using synchrotron radiation for solid-state research that led to the MURA altering the storage-ring vacuum chamber to provide access to synchrotron radiation without interfering with accelerator studies (Robinson 2015). Shortly after, the MURA was officially disbanded in 1967, but the University of

Wisconsin took on the task of finishing the storage ring, known as Tantalus I (Lynch 1997, Robinson 2015). The 240 MeV synchrotron, Tantalus I, was completed by Rowe and collaborators in 1968, having a few beamlines with monochromators, and became the first storage ring dedicated to synchrotron research, thus making it the main original model for today's multi-user synchrotron radiation facilities (Willie 1991, Robinson 2015).

However, high-energy storage rings are not optimized for experiments with synchrotron radiation because the beam cross sections or beam emittances are too large. Therefore, this led to the development of low-emittance storage rings in the 1970s, which marked the beginning of second generation storage rings that were solely built to perform synchrotron radiation experiments. Due to the advantages and wide range of potential novel research that could be done, the drive for creating these types of facilities was so great that second generation light sources were built in the USA (Aladdin at Madison and the NSLS at Brookhaven), France (SUPER ACO at Orsay), Germany (BESSY in Berlin), Japan (Photon Factory in Tsukuba), etc.

After testing insertion devices like wigglers and undulators at several synchrotron facilities, a huge increase in photon beam density was obtained. This led to the creation of third generation light sources or upgrading older storage rings to become third generation synchrotrons, which are specifically designed to take advantage of these two main insertion devices by constructing several straight sections. Early examples of third generation light source would be ELETTRA in Italy and the Advance Light Source in Berkeley (USA), which had energies up to 2 GeV (Willie 1991). Today facilities like the NSLS II are the most advanced third generation synchrotrons and its beam is about 10,000 times brighter than its predecessor, the NSLS.

Inside a storage ring, powerful bending magnets guide a beam of highly charged electrons moving close to light speed in a circular path, which in turn continuously circulates current at a fixed energy for periods of time. While this occurs, synchrotron beam is produced by passing the electron beam through special magnet setups that have a large number of periodically arranged poles of alternating polarity, mainly wigglers or undulators, and the synchrotron beam undergoes a repeated sequence of injection, acceleration, and extraction at rates up to a few units of hertz (Robinson 2015). Several advantages come about because of this, like a high-duty-cycle when the beam is accessible, higher radiation flux, better beam stability (Singh 2005), a synchrotron-radiation spectrum that remains constant with time, a reduction in radiation losses (Price 1982) and extreme brightness.

The brightness of a synchrotron X-ray source is also known as brilliance (Eq. 1). It takes into account the number of photons produced per second, the angular divergence of the photons (speed of beam spreading outward), the cross-sectional area of the X-ray beam and the photons falling within a bandwidth (BW) of 0.1% of the central wavelength or frequency.

$$brilliance = \frac{number\ of\ photons}{(sec)(mrad^2)(mm^2)(0.1\%BW)} \qquad \text{(Eq. 1)}$$

The maximum brightness that can be obtained from a conventional laboratory source is limited to about 10^8 photons s^{-1} mm^{-2} $mrad^{-2}$ (0.1% bandwidth)$^{-1}$ because of the heating limitations of their rotating-anode X-ray source. This amount of intensity was inadequate for numerous important experiments and applications. Therefore, an alternative method for increasing the brightness was sought after, which is another reason that led towards the support of synchrotrons. For example, when the third generation synchrotrons were built, they could reach a brightness around 10^{20}, which was approximately 10^4 higher in brightness than the previous generation of synchrotrons, which were already much brighter than the conventional laboratory sources.

Moreover, with the developmental improvements of synchrotrons over time, the experimental setup and detectors for experiments have changed. Because of these changes, data collection is more complex and analyzing it requires improvements in computer software. See Chapters l.6 and 2.1 for more details.

1.3 Synchrotron Powder Diffraction

The expression “powder diffraction” denotes the phenomenon of any electromagnetic waves or particles diffracting on or through polycrystalline (thin film, bulk or powdered) materials which are used for an extensive variety of experiments. The conception of powder diffraction came four years after the 1912 discovery of the single-crystal diffraction phenomenon by Walther Friedrich, Paul Knipping and Max Laue in Germany, which was developed from 1912 to 1913 by William Henry Bragg and his son William Lawrence Bragg. The X-ray powder-diffraction method was pioneered and developed during World War I in 1916 by the Dutch-American physicist/chemist Peter Debye and Swiss physicist Paul Scherrer while in Germany. Hence, the observed diffraction rings from powdered X-ray diffraction experiments were then termed Debye-Scherrer rings.

For a powder diffraction experiment, a bright X-ray beam is used to penetrate a polycrystalline sample for the purpose of fulfilling Braggs law (Eq.2).

$$n\lambda = 2d\sin\theta \qquad \text{(Eq. 2)}$$

Braggs Law was derived in 1913 by William Henry Bragg and his son to determine why crystals seem to reflect X-ray beams at certain angles of incidence. This observation is a case of X-ray wave interference, which is generally known as X-ray diffraction. Their experiment also served as direct evidence for the periodic atomic structure of crystals, which led to the Braggs being bestowed the Nobel Prize in Physics in 1915 for their work in determining crystal structures. For clarity, crystals are defined as solids that have an atomic structure with long-range, 3-dimensional order. This long-range order can only be absolutely confirmed by using a diffraction technique. Even though Bragg's law was exploited to explain the interference pattern of X-rays scattered by crystals, diffraction has been developed to study the structure of all states of matter (solid, liquid and gas) with any beam (photon, ion, electron, proton or neutron beams) with a wavelength that is comparable to the distance between the inter-atomic or inter-molecular structures of interest.

In equation 2, n is an integer that represents the "order" of reflection, λ is the incident beam's wavelength, d is the spacing between crystal planes and θ is the angle of the diffracted incident X-ray. In these types of SXPD experiments hundreds of randomly oriented crystallites that make up the polycrystalline sample diffract the X-ray beam when atomic crystal planes in the crystallites are properly aligned. To fulfill Bragg's Law, monochromatic light is used, where the wavelength is fixed and each crystallographic d-spacing (Figure 1) produces a cone (Figure 2) of diffracted X-ray photons at a fixed 2θ angle. A 2D area detector is used to collect the circular X-ray photon intensity pattern such that the intersection of the detector plane with the cone of diffracted X-rays produces a diffraction ring for each d-spacing. The angle around the ring is the azimuth. Computer software is then used to integrate the intensity over azimuth and the d-spacings are calculated by fitting intensities of the Bragg reflections versus 2θ (Figure 3). Intensities around the ring vary depending on the orientation of the crystallites in the sample.

There are several benefits when performing X-ray diffraction experiments with a synchrotron light source. Since the main synchrotron beam is white light, X-rays are produced over a vast range of energies. Hence, users can extract a preferred single wavelength from the intense white light using a variety of X-ray optics and also benefit from the synchrotron's high polarization. Due to the high flux or number of photons per area per unit time that a synchrotron's can produce, brightness is at least 5 orders of magnitude greater than the finest conventional laboratory X-ray sources. This allows for quickly analyzing unknown materials and characterizing them while obtaining better resolution. Its tunablility, highly collimated beam, extreme brightness and broader range of available wavelengths allows it to be use for a lot more applications compared to normal laboratory based sources.

Compared to conventional detectors, the integrated data of 2D-diffraction data provides better intensity and statistics for phase identification and quantitative analysis, even for samples with texture, large grain size, or small quantity. For textured samples, you can observe intensity variations in the 2D-image (Figure 4) and for samples of large grain size you can witness spotty diffraction rings (Figure 5). 2D-detectors can also collect diffraction data in a large 2θ-range without sample or detector movement. Nonetheless, for conventional 1D detectors, counts of photons follow a Poisson distribution which has a standard deviation that is equivalent to the square root of the intensity. However, though the Poisson distribution may hold true universally when using 2D detectors, the method for calculating the standard deviation must slightly change. This is because 2D detectors are integrating detectors, where the number of photon counts recorded in a pixel is not equal to, but proportional to the number of the detected photons. Hence, the amount of photons contributing to a pixel induces a probabilistic situation along with the fact that now multiple intensity measurements are allocated to a given 2θ bin value. Therefore, the uncertainties are not as straightforward as the square root of the intensity. Nevertheless, programs assume that the square root intensity method is correct and thus the estimation of the uncertainties on raw intensities in 2D imaging are often disregarded. It is also known and easy to see that low intensities would give unreasonably low uncertainties, which will cause the goodness of fit value (see Chapters 1.5.4 and 1.6) to be unreasonably high. This is why it is important to understand that the beginning point for estimating the precision of parameters in refinement models is knowing the uncertainties of diffraction data. Chapter 1.6 goes into more details.

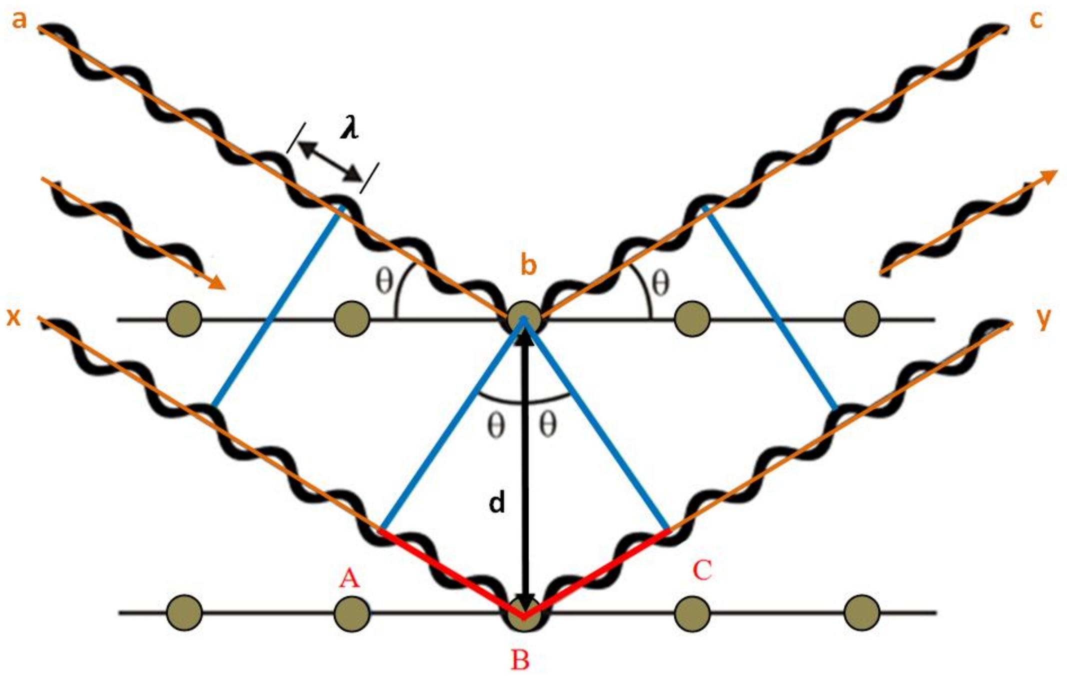


Figure 1: Bragg's Law is shown, where X-rays that are diffracted display constructive interference when the distance between paths abc and xBy vary by an integer number (n) of wavelengths.

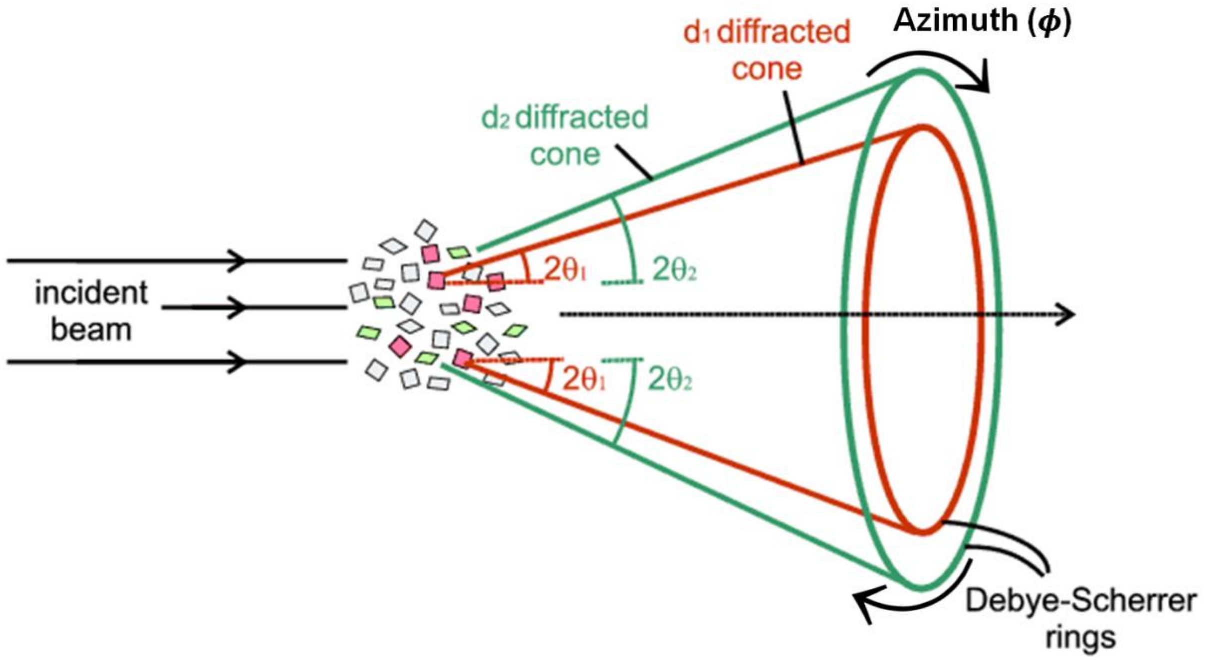


Figure 2: When Bragg's Law is fulfilled, diffracted X-rays will produce cones at a fixed 2θ angle that are related to the d-spacing between crystal planes. This image is modified from Cockcroft (1997).

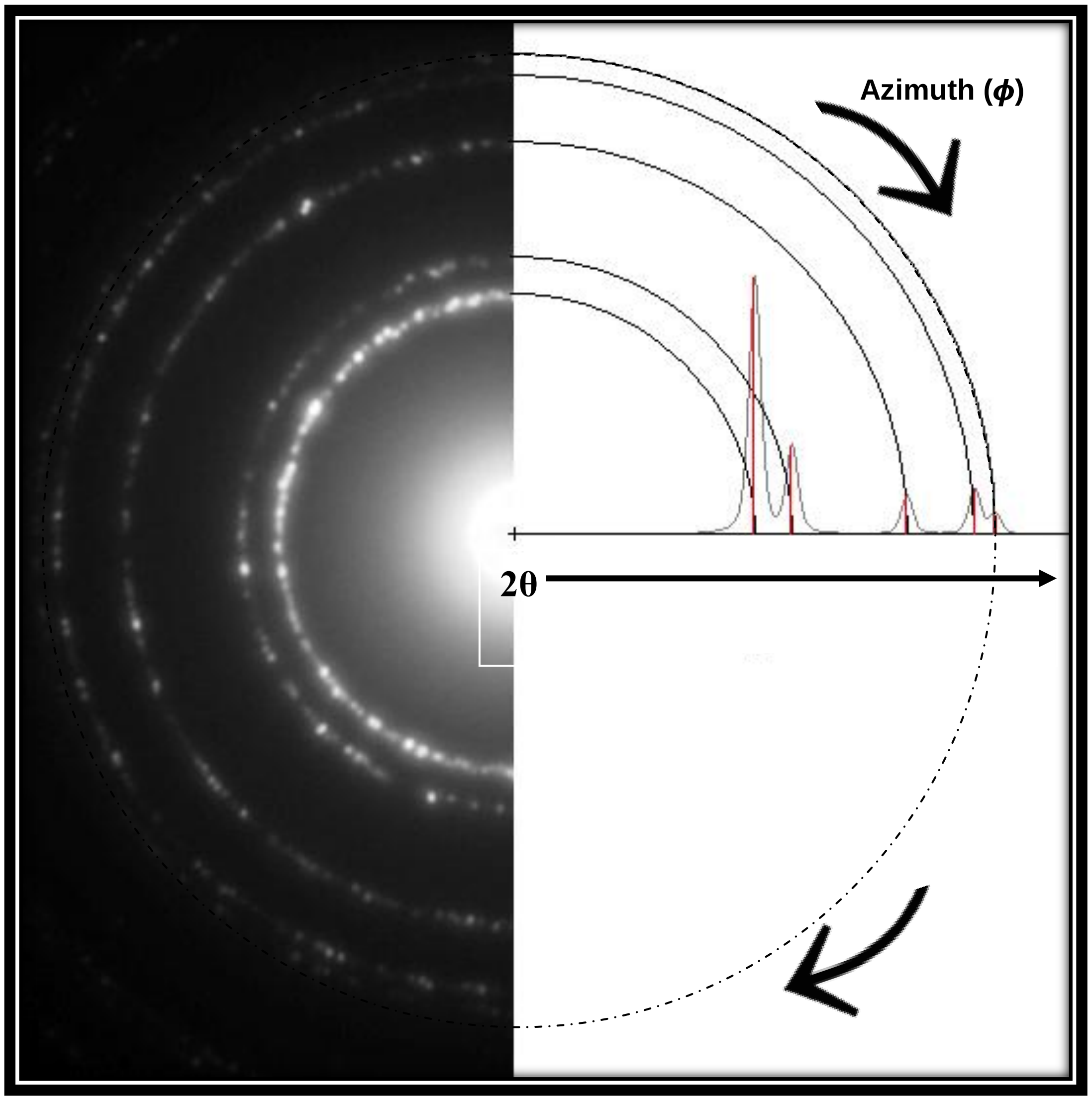


Figure 3: This figure from Li (2010) illustrates how a 2D diffraction pattern is represented by a 1D diffraction pattern, where the collected pixel intensity data of each ring is integrated over azimuth and these calculated intensity profiles are positioned in regards their respected 2θ angle. Notice as 2θ increases the peak intensity profiles decrease. Everything that is not a data peak (reflection) is considered to be background or noise. Hence, if there were intensity readings between the 2θ reflection angles, which aren't from the sample, data peaks to the far right may be hard to analysis due to low peak-to-background ratios. The center of the image is where the most intense light is produced from the X-ray source, which is normally prevented or mitigated by a beam stop. If it wasn't, intensities that are off the scale would dominate the 1D diffraction pattern. The number of crystallographic planes contributing to a diffraction also affects the width of diffraction peaks and compressed samples with internal strain can change the positions and shapes of X-ray diffraction peaks as pressure increases.

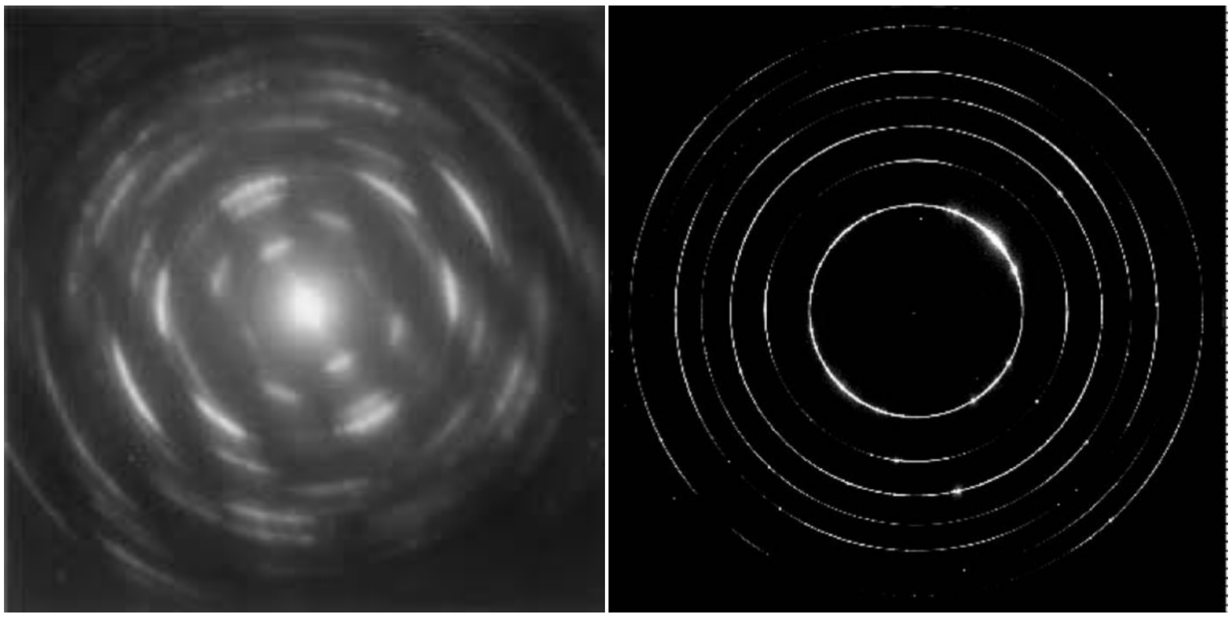


Figure 4: Samples with texture are shown with variations in intensity or brightness around the rings. The left image (Mogilevsky 2003) shows an extreme case with low resolution and the right image (Wang 2002) shows a moderate case with higher resolution.

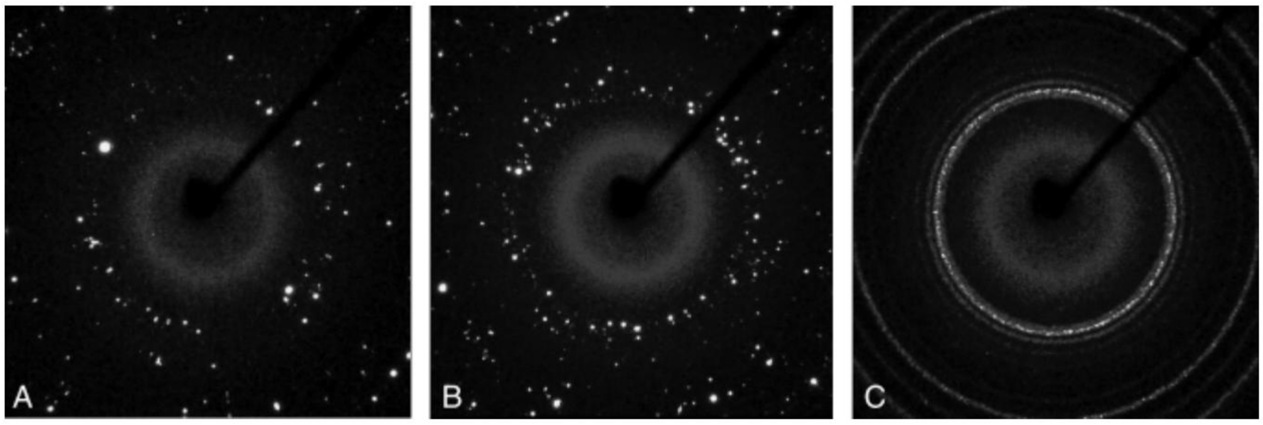


Figure 5: The spotty effect of crystal size on X-ray diffraction images of powdered specimens is shown (Rodriguez-Navarro 2006). Image A through C show polycrystalline samples with average crystal sizes of 87, 37 and 3 micrometers, respectively.

1.4 High Pressure Synchrotron Experiment

For the experimental setup, the major components of the most common high pressure synchrotron diffraction experiments include a diamond anvil cell, sample, monochromatic synchrotron X-ray beam, and a 2D detector. The main components of the diamond anvil cell are shown in figure 6, which includes two diamond anvils for extreme pressurization and a gasket metal foil (e.g. steel, tungsten or rhenium) which acts as the sample and pressure medium holder as well as support for the diamonds. The hole in the center of the gasket serves as a sample pressurization chamber, where the uniaxial pressure supplied by the diamond anvils is transformed into quasihydrostatic pressure using a pressure transmitting medium, such as a gas (argon, neon, hydrogen, helium, nitrogen, etc.), liquid (methanol, ethanol, silicone oil, etc.) or soft solid (NaCl, alkali halides, etc.). The gasket also reduces diamond cracking due to exceedingly large shear forces that result during experimentation. Pressure is usually measured indirectly by use of a standardized pressure calibrant like ruby, gold, etc (Piermarini 2008). The calibrant is inserted into the DAC along with the sample and the pressure is measured utilizing the pressure-dependent peak shift of the calibrant. In the case of ruby, the pressure-dependent peak shift of the optical ruby-fluorescence lines R_1 and R_2 are used as shown in figure 7.

Information about the equipment and setup are necessary because the experimental setup can produce artifacts in the diffraction data that ought to be filtered out. The experimental setup of DAC experiments are made for pressurizing small sample volumes on the order of tens of micrometers and allows for laser heating and X-ray diffraction experiments to all be conducted simultaneously. This is due to the hardness, transparency and other properties of diamond, which allow a vast range of experiments like diffraction, fluorescence, absorption, inelastic scattering and tomography research to be conducted (Bassett 2009). Hence, aside from the expected scattered X-rays and the transmitted beam that is block by a beam stop, additional products like heat, florescent X-rays, incoherent scattered X-rays and electrons can be formed when the synchrotron beam goes through a powdered specimen (see figure 8). Some of these can contribute to background within diffraction patterns. In addition, when X-rays pass through the diamonds (Figure 6) some absorption occurs and this can cause some peak intensities to be low, while diamond diffractions can produce high intensity reflections that can dominate the diffraction pattern. Combined, all these things can create diffraction patterns that don't portray normal crystallographic data distributions of the sample.

In addition, it is important to note that if the beam interacts with the gasket, other reflections will occur. Additionally, before the beam goes through the diamond and right before it reaches the detector, it passes through the air. Hence, air scattering also contributes to background and thus a diffraction image should be taken with no sample inside the DAC. This will contain the general background data that is later subtracted, but this may not always work out.

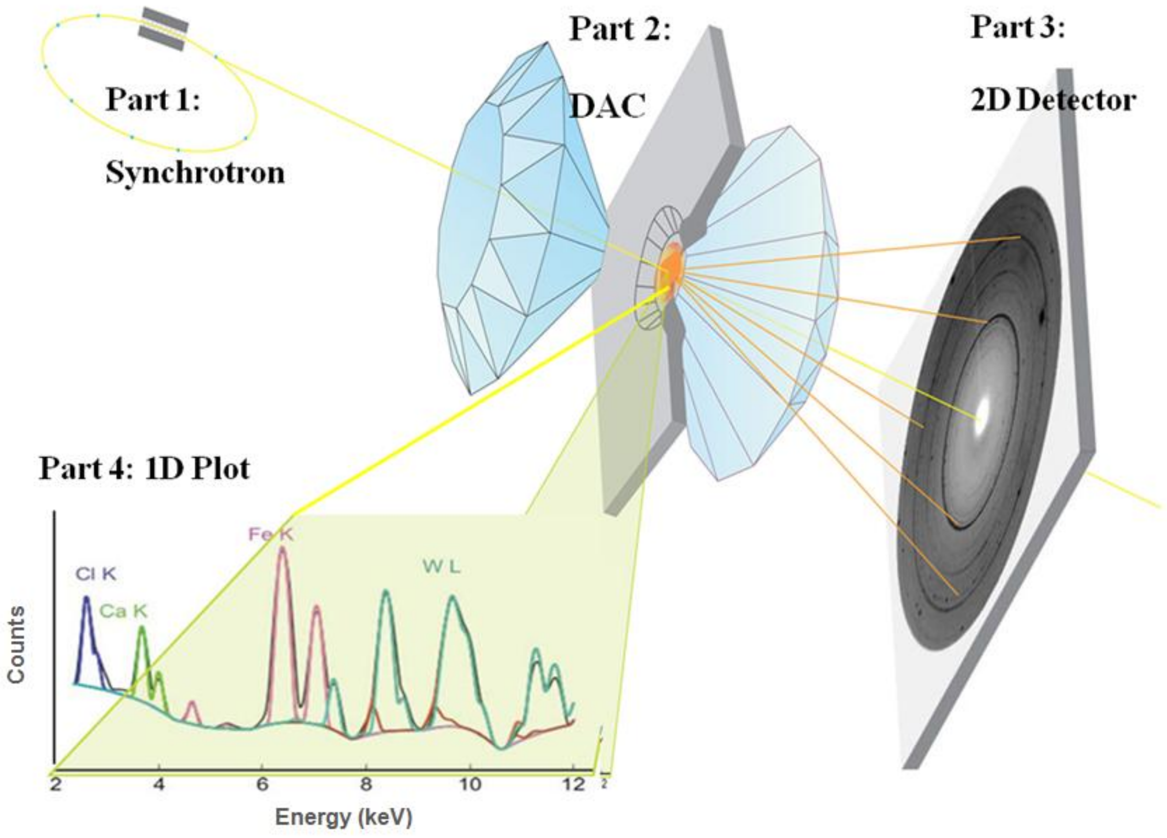


Figure 6 : This modified figure is from ESRF (Andrault 2012). Part 1 represents the inside of a synchrotron, where a highly charged electron particle within a vacuum is magnetically forced to maintain a circular path while moving close to speed of light and is then wiggled by an undulator to produce an synchrotron X-ray beam of intense photon light. Part 2 shows the beam going through a sample within a diamond anvil cell experiment. Part 3 shows the diffracted beam producing a 2D diffraction image or Debye-Scherrer rings upon an area detector. Part 4 shows a 1D fluorescence spectra at various sample locations and illustrates how each element or compound is uniquely characterized by a certain energy, while the height of each peak depends on the concentration of the element or compound.

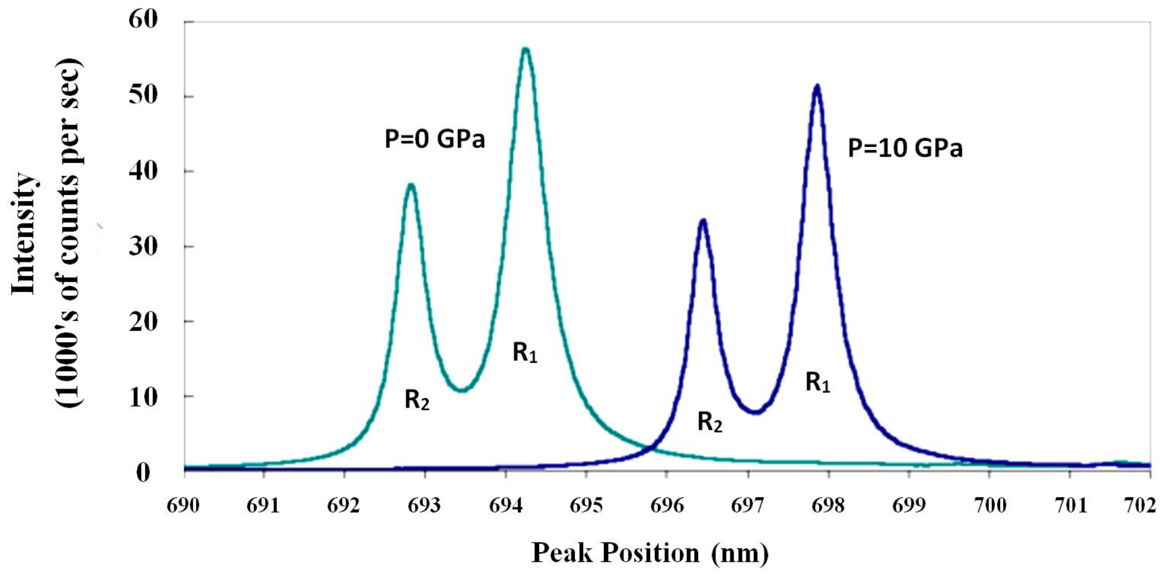


Figure 7: This modified figure from Baker (2012) shows the pressure dependency of ruby fluorescence spectrum starting from ambient pressure (0 GPa) up to 10 GPa.

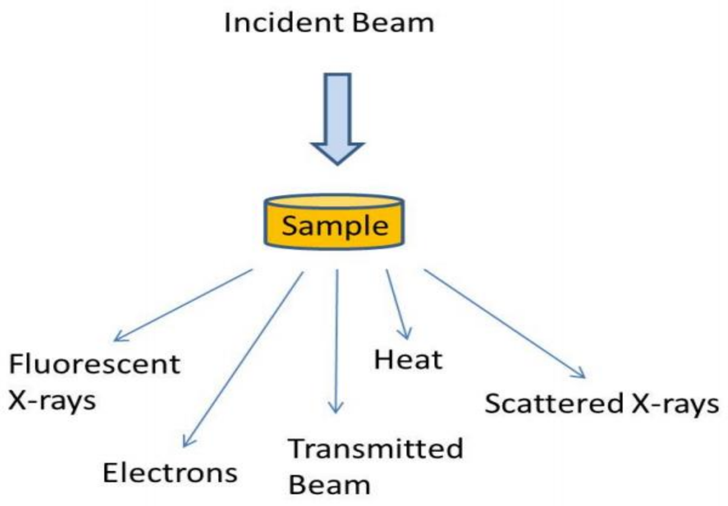


Figure 8: Aside from the expected scattered X-rays and the transmitted beam that is block by a beam stop, this image from (Henry 1951) reveals other products that can be produced when the synchrotron beam goes through a powdered specimen, some of which contribute to background within diffraction patterns.

Another problem with these experiments is the fact that a small beam is used to illuminate a small sample size. Hence, the illuminated sample volume contains only a small subset of crystallites and very few crystallites diffract. This induces low counting statistics. Since crystallites are also randomly positioned, some of their crystal planes may not be aligned to fulfill Bragg's Law and hence even less data may be collected. Even though larger samples would help, sample size is limited due to the nature of the experiment and increasing sample thickness would increase sample adsorption and reduce pressure attainability. Hence, we are left to deal with the factors that contribute to low peak-to-background ratios.

In an ordinary X-ray diffraction experiment scientist observe the number of photons, also known as the count(s), that diffract in a particular direction within a certain amount of time. The data measurements for these experiment normally adhere to "counting statistics", also called Poisson statistics. The Poisson distribution for a large number of counts is virtually identical to a Gaussian or Normal distribution. The standard deviation of a Poisson distributed variable is found to be the square root of the observed number of photons or counts (N) as stated before in Chapter 1.3. These two properties of counting statistics are essential to keep in mind. The latter property implies that the peak-to-background or signal-to-noise is equivalent to N (the peak or signal) divided by the square root of N, which represents the background or noise. Consequently, if N=100, the signal-to-noise equals 10. If 10000 counts were collected, then the peak-to-background ratio would be 100. This reveals the motive for wanting to boost the signal or increase the peak height by increasing the number of counts. For the case of high pressure experiments, filtering out unwanted background and noise that is due to non-sample contributions is more realistically ideal, since experimental constrains limit the amount of counts that can be obtained. In the end, additional counts and filtering noise leads to an improved signal-to-noise ratio, which is very much desired.

1.5 Structure Determination Process

1.5.1 The Phase Problem

Crystal structure solution from powder data is not simple due to various problems. These issues include the uncertainty in the determination of the background, the possible presence of preferred orientation effects and, the collapse of the three-dimensional reciprocal space onto a one-dimensional diffraction pattern, which implies a severe overlapping of the diffraction peaks. As a result, the diffraction residuals of the reflections are only approximately estimated and the uncertainty about the residuals makes some approaches unsuitable for powder crystallography (David 2002).

In general, structure determination involves finding a periodic arrangement of atoms that would produce reflected intensities that fit the intensities of an experimental powder diffraction pattern. For experimental patterns, multiple components such as the structure factor, multiplicity, Lorentz factor, polarization factor, thermal atomic displacement factor, sample absorption, preferred orientation and extinction coefficients all affect peak intensities. Equation 3 is the calculated intensity (I_{calc}) at a particular 2θ bin step size. S_F is a scale factor and L represents the geometry (which involves the Lorentz and multiplicity) and polarization factor, which depends on the experimental setup. F_{hkl} is the structure factor, ϕ is the reflection profile function, P is the preferred orientation, A is the absorption factor and I_{bkg} is the background intensity function.

$$I_{calc} = S_F \sum L\ |F_{hkl}|^2\ \phi(2\theta_i - \theta)(P)(A) + I_{bkg} \qquad \text{(Eq.3)}$$

Out of all these components, the structure factor (F_{hkl}) is the most crucial for structural determination because it describes how the atom arrangement (xyz) affects the incident beam. It is directly related to the intensity (I_{hkl}) of the corresponding reflection h,k,l.

$$F_{hkl} = \sum_{j=1}^{n} f_j\ e^{2\pi i(h(x_j)+k(y_j)+l(z_j))} \qquad \text{(Eq. 4)}$$

In Equation 4, F_{hkl} is the amplitude of the X-ray diffracted by n which is all the atoms in a unit cell, and f_i is the atomic scatter factor, while i is an imaginary number, h, k, and l are Miller indices and x, y, and z are fractional coordinates for the relative location of a particular atom. Miller indices are simply a notation system for crystal lattices and a family of lattice planes is determined by the integers h, k, and l. The calculation of F_{hkl} uses Euler's complex exponential function (Eq. 5), which can be derived by comparing the Taylor expansions of the exponential function and the sum of the two periodic (oscillating) functions sinφ and cosφ.

$$e^{i\varphi} = cos\varphi +\ i\ sin\varphi = c + is \qquad \text{(Eq. 5)}$$

The images in figure 9 are used for visual clarity to illustrate how the complex exponential function is used to represent a 3D space, which in turn can aid in describing real world three-dimensional objects (ex: crystalline samples). These images are used to show the graph of the complex exponential function by plotting the Taylor series of e^{ix} in the 3D and 2D complex space with an x-axis, real-axis (Re) and imaginary-axis (Im).

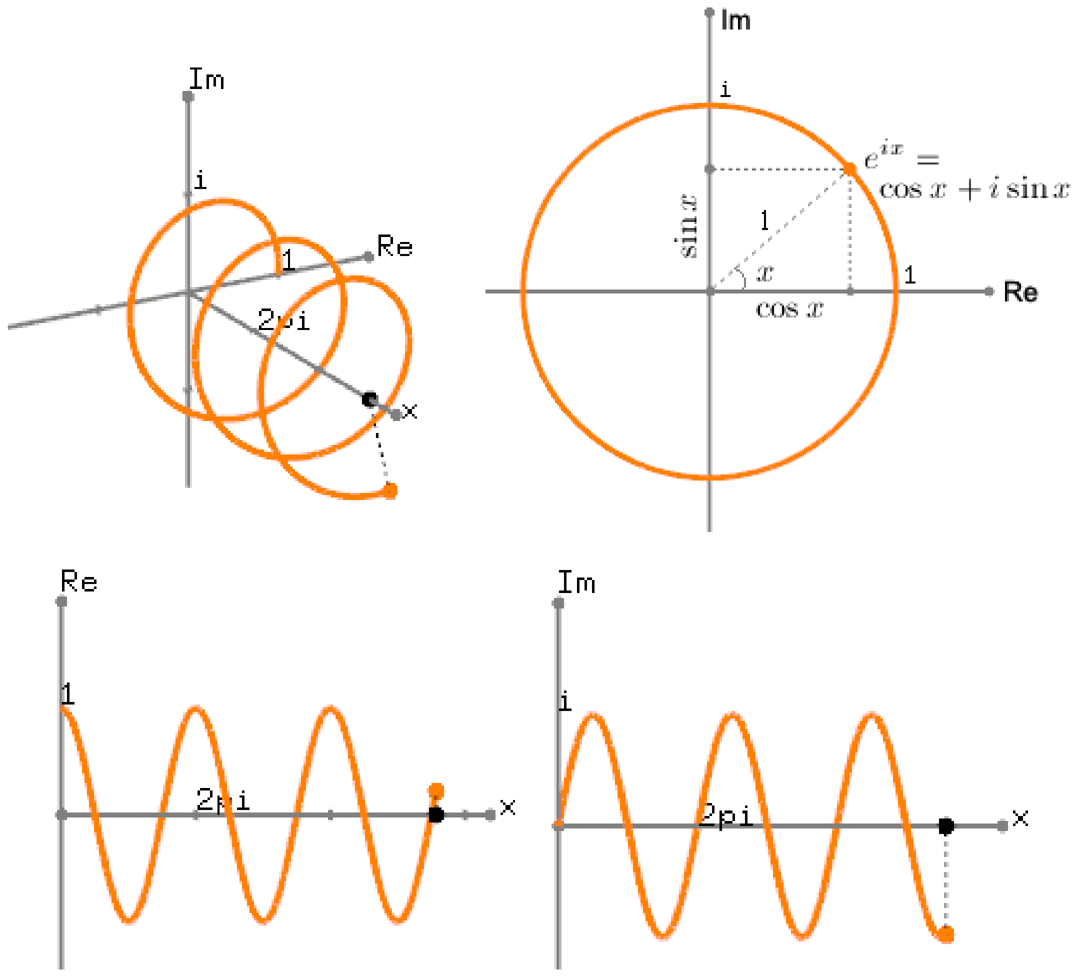


Figure 9: The images are from Song Ho Ahn (2008). On a 3D-raster the complex exponential function is a spiral spring or coil shape, rotating around a unit circle as shown in the upper left corner. The image in the upper right reveals how the complex exponential function relates to the diffraction rings that are produced during experimentation. And, when the complex exponential function is projected to the real number axis (top view of spiral) and imaginary number axis (side view of spiral), it becomes a trigonometric cosine and sine function, respectively.

The phase contribution of an individual atom (j) can be represented as:

$$\varphi = \varphi_j = 2\pi(h(x_j) + k(y_j) + l(z_j)) \qquad \text{(Eq. 6)}$$

Hence, Eq. 4 can be rewritten as:

$$F_{hkl} = \sum_{j=1}^{n} f_n(c + i\,s) = C + iS \qquad \text{(Eq. 7)}$$

By using the fundamental geometry of the diffraction measurements that is characterized for 2D detectors, as shown in the figure below, we can use the relation tan(2θ) = r/d and equation 7 to derive equation 8.

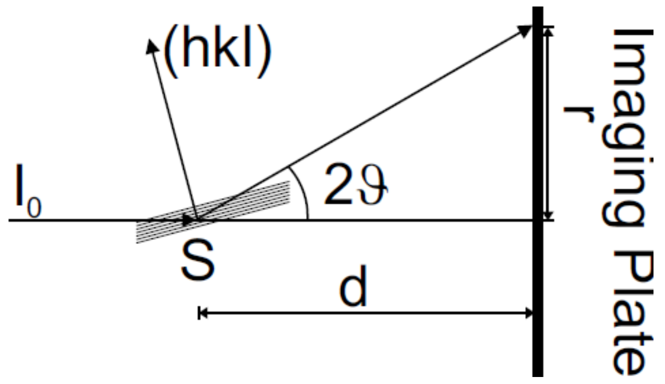


Figure 10: This image from Vogel (2001) illustrates the basic geometry of diffraction experiments using a 2D detector. The diffraction angle (2θ) depends on the distance d between the sample (S) and detector as well as the distance r between the primary beam spot on the detector and the point where the intensity (I) was measured.

Thus, the phase angle of each hkl reflection can be given by:

$$\varphi = tan^{-1}\left(\frac{S}{C}\right) \qquad (\text{Eq. } 8)$$

It can now be noticed that for a centrosymmetric structure, all coefficients s and S will zero out as sin(x) = -sin(-x), and the phase angles will then be limited to 0° or 180°, depending on the sign of C. This reveals the significance between crystal structure and diffraction phase angles.

Furthermore, it is important to note that the complex scattering factor F_{hkl} cannot be calculated directly from the intensity. Only $|F_{hkl}|$, which is the scalar magnitude of F_{hkl}, is available from the experiment and it is not directly computable because the measured intensity is not equal to, but proportional to $|F_{hkl}|^2$. The root cause of this situation is due to what is known in crystallography as the phase problem, which must be solved for the determination of a structure from diffraction data. The phase problem is the name given to the issue of losing information concerning the phase that can occur when making a physical measurement.

For powder diffraction measurements, individual intensities consist of the superimposition of reflections that have the same d-spacing. Hence the multiplicity, which is the number of symmetry equivalent reflections contributing to an intensity, must be taken into account to model the diffraction pattern. A three-dimensional crystal can be depicted by identifying the shape, size and atomic contents of the simplest repeating unit of a crystal and the way these repeating units stack to form the crystal. This simplest repeating unit is known as an atomic unit cell, which is defined in terms of lattice points. These are points in space about which the atoms are free to vibrate in a crystal and temperature affects how much the atoms vibrate, which is known as thermal vibration. Temperature vibrations that are accounted for by the Debye-Waller factor, cause changes in the unit cell. This can increase background scattering and decrease diffraction intensities. The crystal structure itself can produce extinction, which is owed to destructive interference within crystals, which reduces or eliminates intensities. In other words, the extinction measures X-ray light absorption that occurs within the sample.

1.5.2 Indexing

After taking all the aforementioned things in Section 1.5.1 into consideration, it is essential to note that data processing begins with indexing the reflections. This means identifying the dimensions of the crystal's unit cell and determining which reflection peaks correspond to which family of crystal lattice planes (hkl). A byproduct of indexing is determining the symmetry of the crystal, which may allow for the determination of the crystal's space group or a set of possible space groups. The space group represents the full symmetry of the crystal, which is directly related to how crystalline specimens diffract X-rays. Overall, the crystal lattice parameters and space group are needed for defining crystal structures.

Several methods can be used to lead towards the structural determination of data. The figure below shows the basic road map that includes all the general steps that are necessary for structural determination.

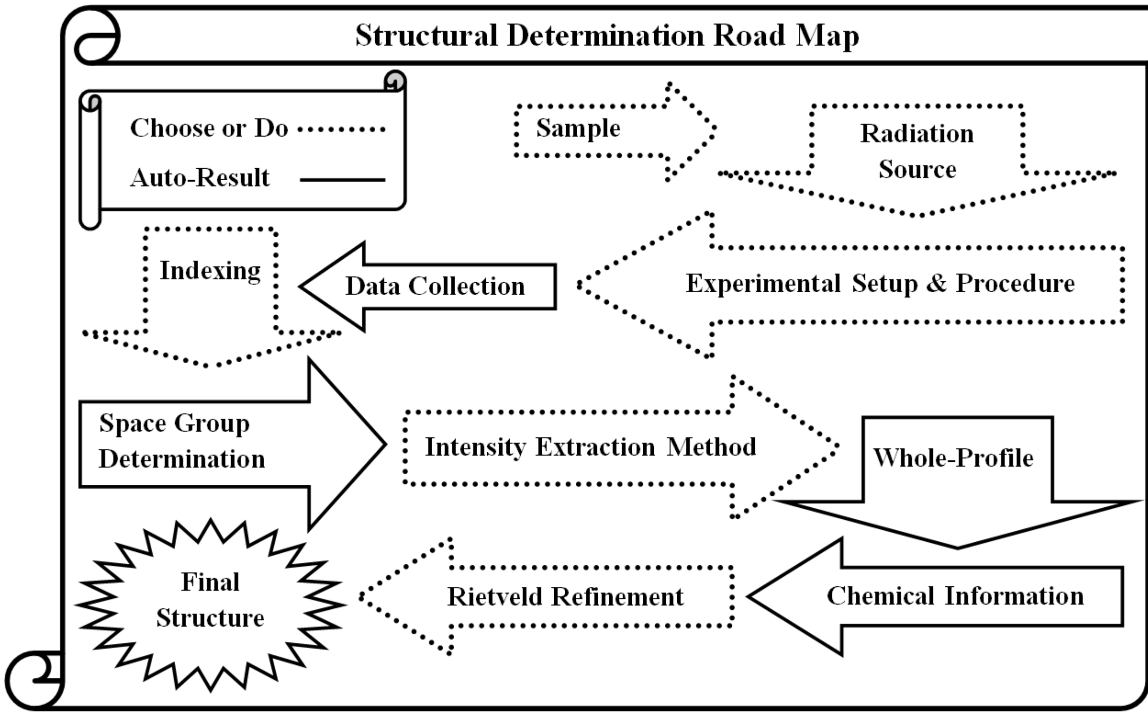


Figure 11: Several methods for data collection, data analysis and structural determination exist. The road map contains all the general steps that are required for structural determination methods. The legend to the upper left of the figure indicates that the dotted steps are steps that the experimentalist chooses to do, since there are several options for these steps. The solid line steps are an automatic result of the proceeding dotted step.

1.5.3 Intensity Extraction

In order to solve a crystal structure from powder data it is necessary to extract as many hkl and intensity values as possible from the data set. Hence, after indexing and space group determination, an intensity extraction method like the Pawley or Le Bail method (Pawley 1981, Le Bail 1988) is used for refinement in order to obtain accurate lattice parameters without employing other structural details. A key benefit with these pre-refinement methods is that they can provide relatively precise integrated intensities, which are imperative for solving the crystal structure.

Until the 1980s, intensity extractions and data refinement were not feasible due to the overlapping nature of the peaks within the powder diffraction profile. However, with the development of high speed computers with large memories and high resolution technology, pattern decomposition became a feasible and essential part of the analysis of powder data. The first technique was created by Pawley in the early 1980s. It is used to extract the intensities and their correlations by refining the intensity of each hkl reflection, unit cell lattice parameters along with background, the instrumental zero error, peak width parameters and peak shape parameters. In a Pawley fit, each hkl value is a parameter in a least squares fitting matrix, which makes it not as easily implementable into Rietveld refinement software as the LeBail method. In regards to its least-squares minimization procedure, this requires normally a $(10+N) \times (10+N)$ square matrix, where N is the number of symmetry-independent reflections generated for the 2θ range from the data. So for a few hundred reflections (#10 x #10 matrix), a computer memory of about 200 kbytes may be required, but for a few thousand reflections (making a #010 x #010 matrix), the memory required for the matrix alone will be several Mbytes. Consequently, the method's early implementation was unstable and hard to control. Only now has methods improved to make the Pawley method somewhat rival in usability to the Le Bail method.

To overcome some of the issues of the Pawley method, Armel Le Bail created the Le Bail method in 1988. His approach was simple and much easier to code into Rietveld refinement programs, but it is like the Pawley method in regards to the standard parameters it uses. It uses a two-step cyclic process and parameters are fitted by a least-squares method. However, unlike the Pawley method, the intensities of the individual peaks are no longer treated as least-squares parameters and are never refined. Thus, each cycle of least-squares is very quick because the matrix remains minute. Le Bail fitting is pretty much like doing a Rietveld fit, but without the atomic information being present. It only requires unit cell and space group information, which it uses to constrain peak values and shifts. All hkl intensities (I_{hkl}) can freely vary, but they are constrained to be nonnegative and a few other rules like equi-partitioning apply. It can also handle multiple phases and impurities and spurious peaks are often observable. The Le Bail method allows for a time effective high quality fitting of massive amounts of powder X-ray diffraction data and it is relevant for following phase transitions, which makes it useful for high temperature and pressure experiments.

Initially, the Le Bail method defines an arbitrary preliminary value for the observed intensities (I_{obs}). In the equation below, y_i(obs) is an observed intensity profile point, y_i(p) is a profile point on a particular peak, and y_i(calc) is the calculated peak profile point.

$$I_{obs}(p) = \frac{\sum y_i(obs)\, y_i(p)}{y_i(calc)} \qquad \text{(Eq. 9)}$$

A particular intensity value may contain multiple peaks and other peaks can be calculated using this equation. The final calculated intensity for a peak is calculated as:

$$y_i(calc) = y_i(p_1) + y_i(p_2) + ... + y_i(p_n) \quad \text{(Eq. 10)}$$

The summation is done over all contributing peak profile points (i). This summation is referred to as a profile intensity partitioning, and it works for any number of overlapping peaks since the intensity is selected based on the multiplicity of the intensities that contribute to a particular peak. Though the observed intensity will be biased owed to the initial arbitrary values that are chosen for the calculated intensity, the intensity values that are observed will normally be closer to their correct value. The process continues by setting a new calculated factor for the observed value. The iterative process is then repeated with the new structure factor estimate and the unit cell, background, peak widths, peak shapes, and resolution function are refined. This improves the parameters and the structure factor is finally reset with a fresh set of observed values.

In terms of the pros and cons, it should be noted that the Pawley method should be used since it provides not only intensities, but estimated standard deviation values as well. But, the Le Bail method offers much greater computational speed. Both methods intrinsically have an issue with peaks that severely overlap, but this is less problematic for the low-symmetry crystal systems because peaks with near identical *d*-spacings occur. However, the problem is very severe for the higher symmetry crystal systems because it is impossible to receive reliable peak intensities due to indistinguishable crystal symmetry (Cockcroft 1997). Figures 12 and 13 illustrate the problematic overlapping nature of peaks and their correlation. Nevertheless, in addition to obtaining reliable intensities, both methods have other advantages. One benefit comes about because in both methods the intensities are not constrained by any structural model, so the final weighted-profile *R*-factor, which determines the reliability of the fit, will be better than the *R*-factor than can be attained from a Rietveld refinement. Another benefit is that the methods are very useful for comparing different space group assignments. So if two space group assignments where possible for a particular crystal structure, then a Pawley or LeBail fit using both space groups could identify the choice that is best.

Figures 12 and 13 are from J. K. Cockcroft, et al. (1997).

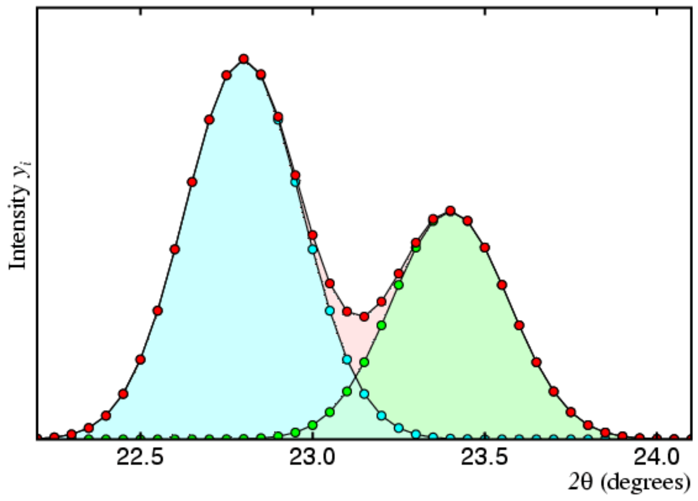


Figure 12: The observed profile (or total count) is shown in red and the contributions of the blue and green peaks are shown. The two peaks are moderately well separated and there is a modest amount of correlation between their intensity values as obtained by least-squares fitting of the observed profile.

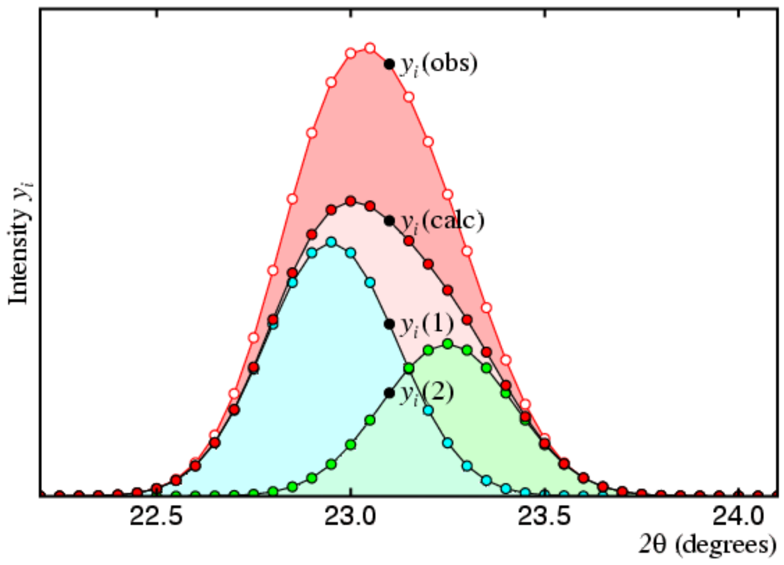


Figure 13: An increase in correlation between the blue and green peaks becomes apparent as they move closer together and consequently any peak intensity values derived from the profile will start to demonstrate correlation. The calculated profile is shown as a black line with red dots and the observed profile is shown as a red curve with white dots. If the blue and green peaks move closer together then all least-squares procedure risks producing incorrect intensity values for the two peaks. For example, if the blue and green peaks had an actual intensity ratio of 3/2, the errors in either the observed profile or peak profile that's used to described the two peaks may possibly lead to ratios such as 5/1, or even worse, like 1/5 for their fitted intensity ratio.

1.5.4 Rietveld Refinement

After the use of either Pawley or Le Bail refinement, the Rietveld method (Rietveld 1969) follows next. This method was devised in the late 1960's by Hugo Rietveld to perform crystal structure refinement on the entire powder diffraction pattern profile and it requires atomic positions, unit cell and space group information. The main goal for of Rietveld refinement is to minimize the residual functions using a non-linear least squares algorithm and thus refine the unit cell paramaters, atomic positions, Debye-Waller factors and overall crystal structure of a compound. Synchrotron X-ray powder diffraction experiments are utilized for the purpose of atomic structure determination and refinement of crystalline materials. Hence, Rietveld refinement is a structural crystallographic refinement technique that is used for the fulfillment of this purpose.

In the Rietveld method, the measured intensity profile of diffraction data and a calculated intensity profile are compared and the variations between both profiles is minimized using a least squares approach. Atomic positions, crystal lattice parameters, background parameters, and profile parameters like peak shape and peak broadening are fitted. The quality of fitting is assessed based on a range of numerical fitting criteria. The main criteria (Young 1995) are the profile residual factors, R_{wp} and R_{exp}, and the goodness of fit (GOF).

$$R_{wp} = \sqrt{\frac{\sum w(I_{obs} - I_{calc})^2}{\sum w(I_{obs})^2}} \qquad \text{(Eq. 11)}$$

$$R_{exp} = \sqrt{\frac{\sum(B - P)^2}{\sum w(I_{obs})^2}} \qquad \text{(Eq. 12)}$$

$$GOF = \frac{R_{wp}}{R_{exp}} \qquad \text{(Eq. 13)}$$

In the above equations, I_{obs} and I_{calc} are respectively the observed and calculated gross intensities at a certain step. B is the number of bins in the interval used, P is the number of parameters that are fitted and w is the weighting scheme. The weighting scheme is vital for evaluating error estimates and some schemes incorporate the standard deviation as an integral part of their formula. Notice that for the GOF, the weighting scheme plays its major role in R_{wp}, where R_{wp} follows directly from the square root of the quantity minimized, scaled by the weighted intensities (Young 1995). See Chapter 1.6 for more details. R_{wp} is the weighted residual factor and R_{exp} is the expected residual factor. In other words, these R-factors are reliability factors which help to measure the agreement between the experimental X-ray diffraction data and crystallographic model. Together, they allow us to measure how well the refined crystal structure predicts the observed data. An ideal GOF value is one that is in between the range 1 to 1.5. Values larger than 1.5 may indicate sample recrystallization or preferred orientation problems, or simply that a poor model is present. Values lower than 1 imply that an unreasonable amount of parameters are present based on the data quality within the model.

1.6 Intensity Error Estimation

Methods for calculating intensity errors for synchrotron powder diffraction data has evolved over time because of the advancement in technology. These methods involve calculating the standard deviation as a confidence measure that is used to quantify the amount of variation in a diffraction dataset of values. Each method is primarily based on the experimental setup and the type of detector used. A continuous growth in detector development eventually brought forth a new revolution in the field of powder diffraction and led to large area detectors, which made their debut in powder diffraction at synchrotron facilities during the start of the 1990s. The first experiments only made use of thin equatorial strips (Norby, 1997) of the image, but with the introduction of freely available software like FIT2D (Hammersley et al., 1996), the integration of the whole image to a standard 1D powder diffraction pattern became routine (Hinrichsen 2007).

Equation 14 shows Hammersley's error estimation method, the weighted average variance, which is used in FIT2D. FIT2D is still most commonly used today, since few programs exist.

$$\sigma_i^2 = \frac{\sum_{j=1}^{N} w_{ij}\left(I_j - O_i\right)^2}{\sum_{j=1}^{N} w_{ij}} \; x \; \frac{N}{N-1} \qquad \text{(Eq. 14)}$$

$$O_i = \frac{\sum_{j=1}^{N} w_{ij} I_j}{\sum_{j=1}^{N} w_{ij}} \qquad \text{(Eq. 15)}$$

O_i is the output pixel, from i=1 to an arbitrary value, and its value represents the weighted average of the contributing input pixels I_j, j=1 to n, where the weights w_{ij} are fractions of the input pixels that contribute to an output pixel. The N /(N - 1) term accounts for the fact that the mean is derived from the data, where N is best set to the sum of weights.

Since this may be calculated from the difference between the weighted average of the squares of the input pixels and the square of the weighted average, equation 16 follows:

$$\sigma_i^2 = \left(\frac{\sum_{j=1}^{N} w_{ij}(I_j)^2}{\sum_{j=1}^{N} w_{ij}} - {O_i}^2\right)\left(\frac{N}{N-1}\right) \qquad \text{(Eq. 16)}$$

However, according to Hammersley, the above equation is only valid if the independence of measurements is assumed, but in reality this is not the case. This is due to the relatively large range of the point spread function (PSF) for 2D detectors compared to the pixel size, which cause a correlation in adjacent pixel values. However, for the latest detectors generations, the PSF is close to one pixel. The PSF describes the spreading of the X-ray beam away from a central point of focus and the presence of the pixel correlations cause the above equation to underestimate the standard deviation since it views the measurements as being independent. There is also generally more correlation between adjacent broadly curved peaks compared to adjacent narrowly curved peaks. Thus, in order to obtain accurate errors to resolve the statistical error analysis issue, determining the statistical distribution of the data becomes of utmost importance.

An ideal powder diffraction pattern, which generates the best intensity distribution, can only be produced by a sample that has a large number of randomly oriented crystal grains that are uniform in size. On the contrary, this ideal situation is not typically the case, especially for high pressure experiments which are limited to a small sample size owed to the experimental setup. Hence, experimental results stray away from the ideal case and for high pressure experiments, diffraction patterns also deviate from normal diffraction patterns due to artifacts that are only common within high pressure data. See Chapter 1.4 for more details.

Following are different types of distribution methods that several authors have witnessed for SXPD data. The probability of finding a desired intensity value in each distribution can be described as $P_{distribution}(I)$. The first is the commonly known Poisson distribution (Chall 2000) which was discussed in Chapters 1.3 and 1.4. It is well known that taking many repeated intensity measurements of a single 2θ diffraction angle yields a Poison Distribution. This is essentially the general case when performing synchrotron diffraction experiments.

$$P_{Poisson}(I) = \frac{\bar{I}^I}{I!}\, e^{-\bar{I}} \qquad \text{(Eq. 17)}$$

The intensity is I, $\sigma^2 = \bar{I}$ is the arithmetic mean intensity and $\sigma = \sqrt{\bar{I}}$ is the observed standard deviation. The standard deviation of the mean intensity of the Poison distribution is $\sigma_I = \sqrt{\bar{I} / N}$, where N is the number of raw intensity data points. When considering textured samples (Vogel 2001), it is proposed that the intensity error should be calculated based on the variations in the distribution of a given population. The equation of variance is as follows:

$$(\sigma_T)^2 = \left(\frac{\sum_{i=1}^{N}(I_i - \bar{I})^2}{N}\right) \qquad \text{(Eq. 18)}$$

Whenever intensities are greater than 20 counts or if $\bar{I} > 20$, a Gaussian or Normal distribution applies and is given below.

$$P_{Normal}(I) = \frac{1}{\bar{I}\sqrt{2\pi}}\, e^{-\frac{(I - \bar{I})^2}{2\bar{I}}} \qquad \text{(Eq. 19)}$$

The standard deviation of equation 19 can be utilized as a measure of the observed intensity's uncertainty, if solely one measurement is conducted, as is the case for experiments using a conventional 1D detector. In this situation of a single measurement, the average intensity is uncertain. For this circumstance, the measured intensity (I) is utilized to find the error estimate. It can be noted that a 68% confidence interval for the Gaussian distribution is one times the standard deviation. Thus, with a confidence of 68%, the mean intensity value is within the calculated error of the measured intensity. It should also be noted that though the Gaussian error approximation ($\sigma = \sqrt{I}$) can estimate raw intensity data successfully, it is not adequate for calculating the uncertainties of averaged intensity data. Averaging all intensity data points inside individual equidistant 2θ bins can help reveal peaks that would otherwise be hidden in complex SXPD data. Figure 14 is a modified figure from Chall (2000) that shows how useful this can be for viewing a complex diffractogram. If a diffraction pattern were divided into equidistant 2θ bins, where a random intensity data point is selected from each bin, then peaks could end up not being identifiable since data points are widely scattered. Thus, the Gaussian error estimation isn't suitable for calculating intensity uncertainties of averaged data because it doesn't take into account the number of raw pixels residing within each bin (Chall 2000).

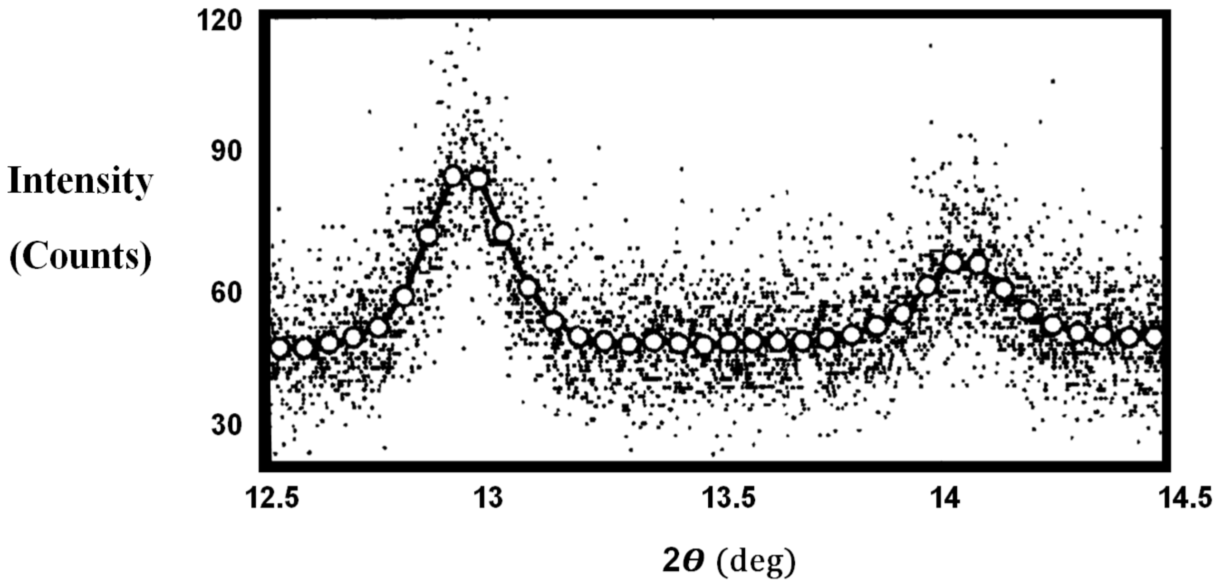


Figure 14: This modified figure from Chall (2000) compares averaged pixel bin values with the raw spread of intensity data. The raw data scatter was found to be well estimated by the error of √I. However, since the error of averaged data is much less, the second peak would not be clearly visible if the error was √I.

Another distribution that was found to describe some SXPD data was the Log-Normal distribution ($P_{\mathrm{LN}}(I)$). When very few crystallites are present as in HP-SXPD, contributions from intensities are likewise fewer and can provide unreliable estimates. The distribution in this case is similar to a Log-Normal distribution (Fewster 2014).

$$P_{LN}(I) = \frac{1}{\sigma I\sqrt{2\pi}}\, e^{-\frac{(\ln(I) - \bar{I})^2}{2\sigma^2}} \qquad (\text{Eq. } 20)$$

In addition, since high pressure XPD data tend to consist of a far lesser amount of very high intensity pixels compared to a large number of low intensity pixels, a power law may be applicable. Therefore, a power law probability distribution may be appropriately applied, such as the Pareto distribution (Hinrichsen 2007).

$$P_{Pareto}(I) = \begin{Bmatrix} 0\, for\; I < b \\ \frac{ab^a}{I^{a+1}}\; for\; I\; \geq b \end{Bmatrix} \qquad (\text{Eq. } 21)$$

For the Pareto distribution function b represents the minimum intensity value and a is a form parameter. The Pareto distribution was originally created to describe the wealth in societies and showed how 20% of a population owned 80% of the wealth. In regards to HP-SXPD data it could be found that 20% of the diffraction rings or 2θ bins account for about 80% of the overall intensity contributions to the pattern.

According to Hinrichsen (2007), the intensity distribution of high pressure XPD data also tend to deviate from ordinary X-ray powder diffraction. It was then found to follow a type of 'Gaussian blur' of the Pareto distribution, which is mainly attributed to the detector's point spread function. To account for this, the convolution ($P_{NP}(I)$) of both functions was used to obtain a best fit. The convolution of these functions is known as the Normal-Pareto distribution and it is illustrated as:

$$P_{NP}(I) = P_{Normal}(x) \otimes P_{Pareto}(x) \qquad \text{(Eq. 22)}$$

The discrepancy between the integrals of the ideal Gaussian and Normal-Pareto distribution may be utilized to approximate the normal portion of the distribution. Because the low intensity slopes of both functions are basically equivalent, the discrepancy can be used to compute the section of high intensity data that needs to be removed, using the estimation that the high intensity slope of normally distributed data is infinite. This allows for the extraction of normally distributed data and fundamentally reliable intensities for high pressure XPD data, which could be useful for structural refinement (Hinrichsen 2007).

Moreover, figure 15 shows three distributions that have been found to best fit and evaluate HP-SXPD data. It was found that out of these three distributions, which Hinrichsen used, the Normal-Pareto distribution was the best fit for the high-pressure data. Obtaining a reasonable goodness of fit for data analysis depends on whether or not a meaningful weighting scheme is used. The weighting scheme critically relies on the acquisition of correct data intensities and their proper error uncertainties. The four main weighting schemes are the unit weights ($w_u = 1$), the default weights ($w_d = 1/I$) which is used by most powder diffraction computer software, the equivalent weights ($w_e = 1/\sigma^2$), where:

$$\sigma^2 = \left(\frac{\sum_{i=1}^{N}(I_i - \bar{I})^2}{N-1}\right) \qquad \text{(Eq. 23)}$$

and finally the correct weights ($w_c = N^2/\sigma^2 = 1/C^2$), where N is the number of raw intensity data points from which the mean intensity is derived and C represents the confidence interval of the mean. The standard deviation (σ) in this case can be approximately calculated from the mean intensity, such that σ equals the square root of the mean, or it can be directly computed using Eq. 23.

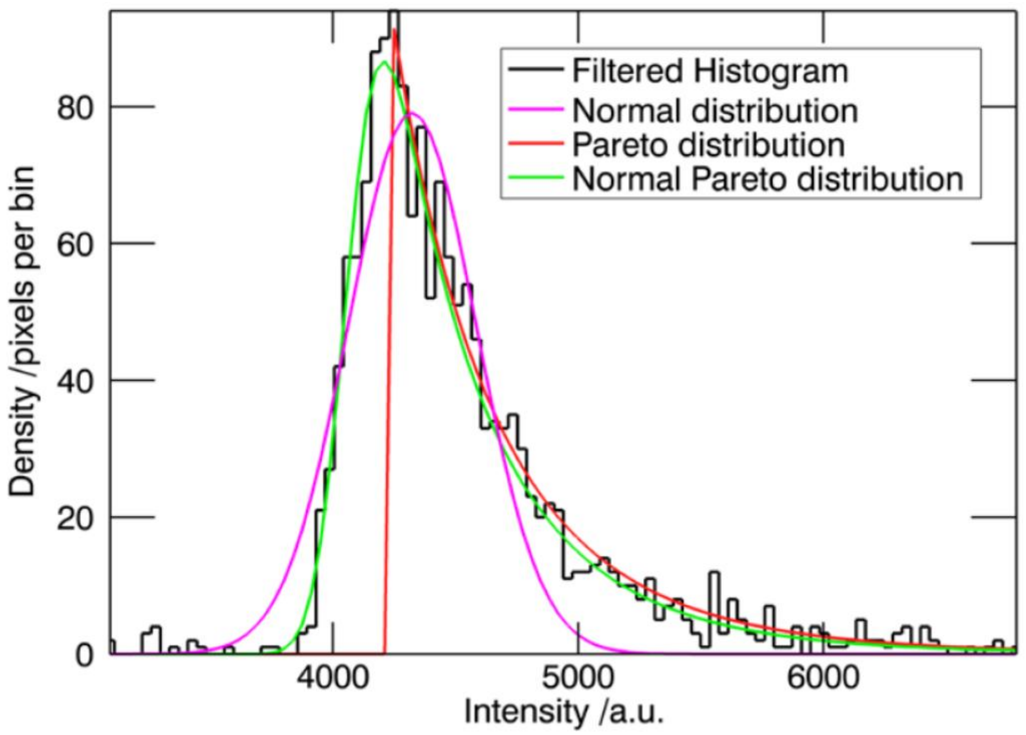


Figure 15: This graph from Hinrichsen (2007) displays an azimuthal intensity distribution of a filtered bin (or Bragg peak) from a high pressure experiment. The Normal, Pareto and Normal-Pareto distribution are represented in pink, red and green respectively. The observed best fit is given by the Normal-Pareto distribution.

If the incorrect weighting scheme is chosen, then wrong and meaningless GOF values are obtained. For example, the default weighting scheme that is used in most software programs mistakenly assume counting statistics (e.g. $\sigma = \sqrt{I}$) are sufficient, and this can cause large "overestimated" errors on the intensities that result in unrealistically low GOF values and dramatically underestimated errors in the refined structural parameters. Figure 16 uses the data in Figure 14 to show the effect of the four weighting schemes on the goodness of fit. Based on the results, only the correct weighting scheme provides meaningful GOF values and other schemes produce extremely low GOF values that are much less than unity. This puts forth the importance of using the correct weighting or else the quality of the fit can't be properly judged.

Another factor to consider is bin width choice, which directly influences the number of equidistant bins that are used for averaging intensity data. Hence, this affects the precision of the standard deviation, which affects the quality of fit. Figure 17 used the data in Figure 14 to demonstrate how the bin width influences the goodness of fit. The results reveal that utilizing smaller bin sizes, which increases the number of bins, lead to more meaningful GOF values that are well within the required 1 to 1.5 range. If underbinning occurs, the number of bins would be too few to resolve peak reflections and the GOF value will rise since the fit would no longer be able to effectively describe the overly broad dataset. The program, py321DIAS, which is describe in Chapter 2.2, allows the user to choose the most suitable bin width and has statistical distribution methods that the user can test for the purposes of obtaining a best fit and for filtering unwanted data.

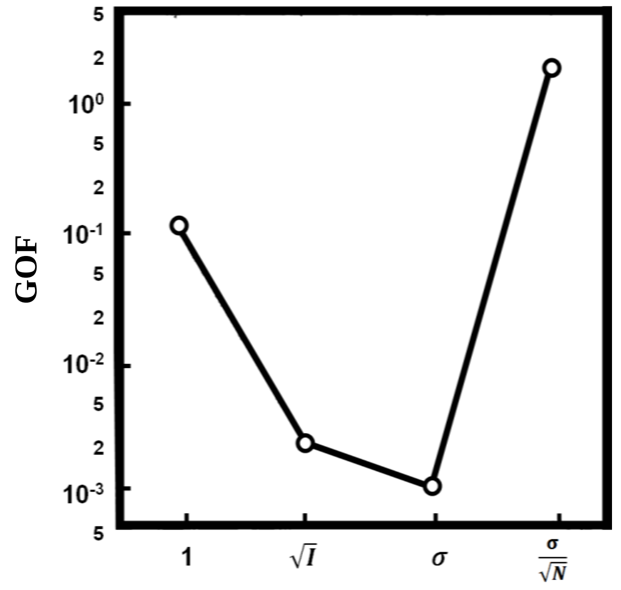


Figure 16: This modified figure from Chall (2000) shows how different weighting schemes influence the goodness of fit. The relationship between the schemes and GOF is shown by using the reciprocal square root of each weighing scheme for the x-axis. All weighting schemes, with the exception of the correct weighting scheme, produce unreasonable low GOF values.

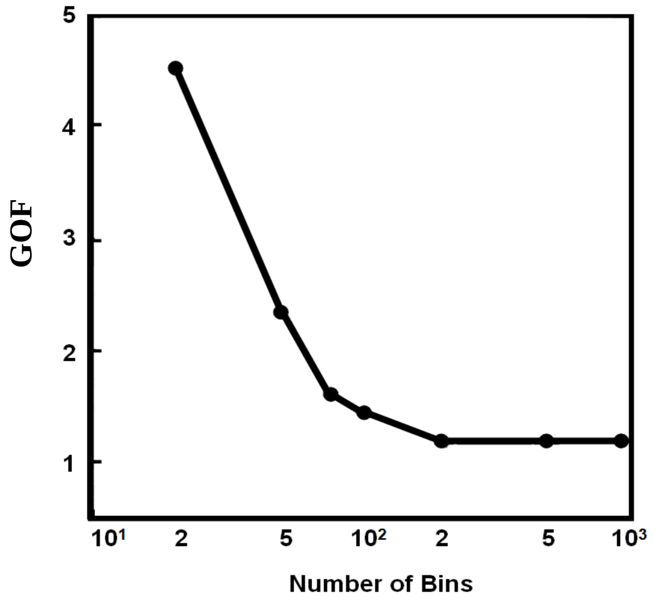


Figure 17: This is a modified figure from Chall (2000). It illustrates how the bin width, which determines the number of 2θ bins, affects the goodness of fit. As the number of bins increase, the value of the GOF monotonically decreases from a large unreasonable value until it converges to a lower more realistically meaningful value that's around 1.2.

Chapter 2 - Programs and Methodology

2.1 FIT2D and pyFAI

The most commonly used program for pre-processing X-ray powder diffraction data is FIT2D (Hammersley 1996 and 2016), which was written in Fortran and created in the 1990s. However, with the advancement of synchrotrons as discussed in Chapter 1.2 and with the advent of more complex experiments that require fast data rates, FIT2D and other software like SPD (Boesecke 2007), which was written in C language, reached their limitations due to their monolithic nature which could not keep up with the pace of data flow of the latest detectors (Kieffer 2014). This led to the creation of pyFAI (Kieffer 2013), which started its developmental stages in 2011 and is becoming more popular. It takes advantage of modern day parallel software and a computer optimization technique called memoization, which is a way to lower a function's time cost in exchange for a greater use of computer memory space. It also uses a computer evaluation method called lazy_evaluation, which prevents the use of a calculation or function until it is needed. This helps to avoid repeating evaluations and reduces the amount of time needed to run certain functions by an exponential factor compare to other strategies.

Though pyFAI is better equipped for fast and large data pre-processing, both FIT2D and pyFAI are good for reducing two-dimensional XPD images into 1D diffraction plots. However, they utilize a default pixel-splitting algorithm that causes statistical correlations among data in adjacent bins for 1D diffraction patterns. These cause statistical uncertainties to rarely be determined (Yang 2014). As stated and emphasized in prior chapters, obtaining reliable intensities and their correct uncertainties are crucial for acquiring a meaningful weighting scheme for the purpose of obtaining a sensible goodness of fit, which is vital for high-pressure SXPD data.

The advantage of pyFAI is the fact that it uses Python programming, which is a free and fairly readable open source computer programming language that acts like a snake in terms of its ability to wrap around and in a special way consume components of other programming languages like C++ and Java, making them a combine part of it. Python also has a vast library for scientific computing, offers many graphics packages and toolsets and allows for the free "consumption" or add-on of other modules, libraries, frameworks, and tool-kits that can be used for a large variety of applications in mathematics, physics, chemistry, crystallography, graphics design, etc. Therefore, we decided to create an integrated open source data intensity analysis software that builds on pyFAI since it uses Python, which is currently very popular for many scientific purposes. By doing so, we can take advantage of Python's potential and utilize pyFAI's well established and functioning capabilities for XPD applications and add a data intensity analysis procedure that will solve data intensity issues.

2.2 Data Intensity Analysis Software

2.2.1 py321DIAS

Py321DIAS was inspire by Beyond FIT2D (Sims 2014) and uses Python's standard GUI (Graphical User Interface) package known as Tkinker. Unlike Beyond FIT2D, which uses PIL (Python Imaging Library) for reading in images and numpy for saving data as .txt files, py321DIAS takes advantage of libraries that are used in pyFAI, like FabIO (Knudsen 2013), for reading and saving data and matplotlib (Hunter 2007), which is a mathematical plotting library. Py321DIAS is mainly an integrated open source data intensity analysis software that can covert 2D X-ray powder diffraction images taken from 2D detectors into 1D diffraction data using azimuthal integration with a routine that performs a statistical bin analysis of the collected diffraction intensities. The statistical analysis utilizes distribution functions to fit the data. It will be used to also provide reliable intensities with calculated intensity errors and any contributions to the diffraction pattern not coming from the sample will be automatically removed utilizing this analysis. This will increase the quality of the data and allow for better fitting, which will ultimately produce better GOF values. Due to this, the program can provide more meaningful intensity error estimates for high-pressure synchrotron XPD data. Therefore, the data can be indexed, pre-refined and utilized for performing an in-depth structural determination purposes using Rietveld refinement.

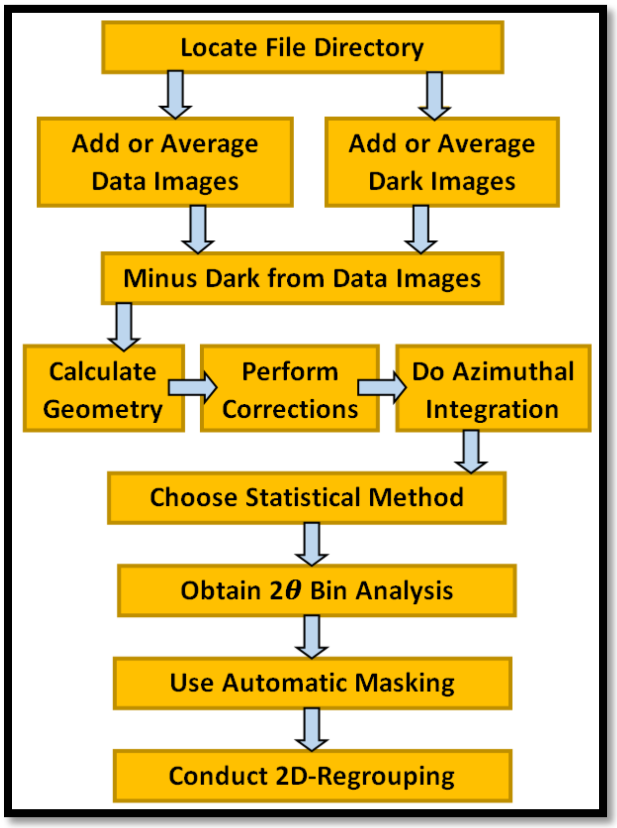
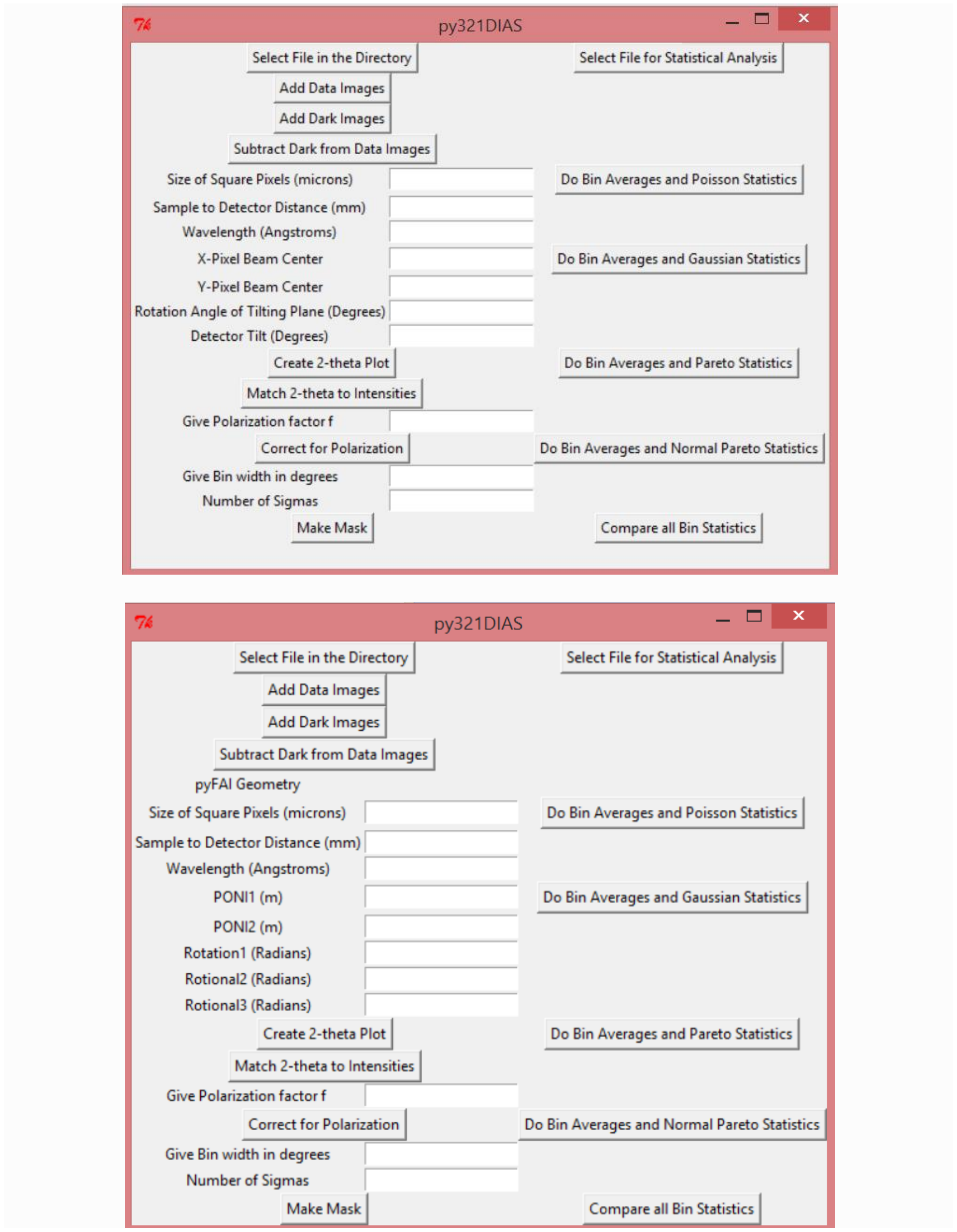


Figure 18: This flow chart illustrates the procedural process for py321DIAS's image processing.


py321DIAS
Select File in the Directory
Select File for Statistical Analysis
Add Data Images
Add Dark Images
Subtract Dark from Data Images
Size of Square Pixels (microns)
Do Bin Averages and Poisson Statistics
Sample to Detector Distance (mm)
Wavelength (Angstroms)
X-Pixel Beam Center
Do Bin Averages and Gaussian Statistics
Y-Pixel Beam Center
Rotation Angle of Tilting Plane (Degrees)
Detector Tilt (Degrees)
Create 2-theta Plot
Do Bin Averages and Pareto Statistics
Match 2-theta to Intensities
Give Polarization factor f
Correct for Polarization
Do Bin Averages and Normal Pareto Statistics
Give Bin width in degrees
Number of Sigmas
Make Mask
Compare all Bin Statistics
py321DIAS
Select File in the Directory
Select File for Statistical Analysis
Add Data Images
Add Dark Images
Subtract Dark from Data Images
pyFAI Geometry
Size of Square Pixels (microns)
Do Bin Averages and Poisson Statistics
Sample to Detector Distance (mm)
Wavelength (Angstroms)
PONI1 (m)
Do Bin Averages and Gaussian Statistics
PONI2 (m)
Rotation1 (Radians)
Rotional2 (Radians)
Rotional3 (Radians)
Create 2-theta Plot
Do Bin Averages and Pareto Statistics
Match 2-theta to Intensities
Give Polarization factor f
Correct for Polarization
Do Bin Averages and Normal Pareto Statistics
Give Bin width in degrees
Number of Sigmas
Make Mask
Compare all Bin Statistics
Figure 19: Displayed are templates for the final graphical user interface for py321DIAS. It will work for both FIT2D and pyFAI geometries.

2.2.2 Reading In, Adding, Averaging and Subtracting Data

Py321DIAS takes diffraction images as 2D numpy arrays like pyFAI and it uses the imaging library known as FabIO (Knudsen 2013) to read in one or more images and for saving outputs. FabIO is advantageous because it is capable of reading more than 20 image formats produced by detectors from 12 different manufacturers (Kieffer 2013). The first module for py321DIAS deals with adding or averaging a set of 2D diffraction images. The X-ray diffraction data, from FIT2D, consists of 16 bit .tiff files that have a 2048 by 2048 pixel frame. A set of dark images and data images can first be added together or averaged, but this must be done one set at a time. Dark images contain any background electronic noise that the detector records without the sample or beam present. Since this 'dark noise' shows up when collecting data, it must be removed. Hence, once the summed or averaged images are created and saved, the dark image set can be subtracted from the data image set using the second module for py321DIAS.

2.2.3 Geometry Correction

The third module for py321DIAS deals with creating the geometry for the diffraction data. For an ideal situation, the intersection of the Debye-Scherrer cone (Figure 2) with the 2D area detector would be a circle. In actuality, the cone axis of the Debye-Scherrer cone and the normal of the area detector's surface are not parallel and the intersection degenerates. As a result, in order to obtain correct diffraction angles from cartesian coordinates (x, y) on the 2D image detector, a geometry correction is paramount. In general, a rotation is applied in order to correct for detector distortion. For example, in FIT2D, if the diffraction cone is tilted by an angle Ø (Figure 21), due to the detector setup, a rotation angle of -Ø is applied to acquire a flattened orientation. The azimuthal integration methods that py321DIAS uses is the same as pyFAI, which is derived from Boesecke’s geometry and Manolo and Sole's histogram algorithm (Ashiotis 2015, Boesecke 2007), and FIT2D (Hammersley 1996 and 2016).

In regards to pyFAI, Kieffer (2013 and 2014) states that since this program is written in Python, which was written in the C programming language, data are stored lines by lines. This implies that in order to move from a certain pixel to one on its right, one must offset the position by the pixel width. To move to the pixel above the current pixel, an individual needs to offset by the length of the line. So, if one considers a pixel at the position (x, y), its value can be retrieved by data[y,x]. The order (y, x) is the proper convention. The *x* axis is referred to as the fast dimension because pixels are adjacent to one another, but since pixels are a line length away for each in regards to the *y* axis, it is known as the slow axis (Kieffer 2013 and 2014).

In practice, Kieffer (2013 and 2014) mentions that the two main conventions for representing images are the imaging and diffraction application. In regards to imaging, when the observing eye of the camera looks at a scene, the origin is generally located at the top of the image. However, for diffraction, the observer is stationed at the sample position and views the detector from there. Since "signed" angles are measured (as described in Chapter 1.5), the origin is ideally situated at the lower left of the image. Because pyFAI is a diffraction application, the origin is thus at the lower-left corner of the image and this makes the polar angle increase from 0^o along the x axis to 90^o along the y axis. Therefore, like pyFAI, the experiment geometry of py321DIAS is defined by the position of the detector with the origin being located at the sample position or more specifically, where the X-ray beam crosses the diffractometer's main axis. The figure below show the geometry setup that is used.

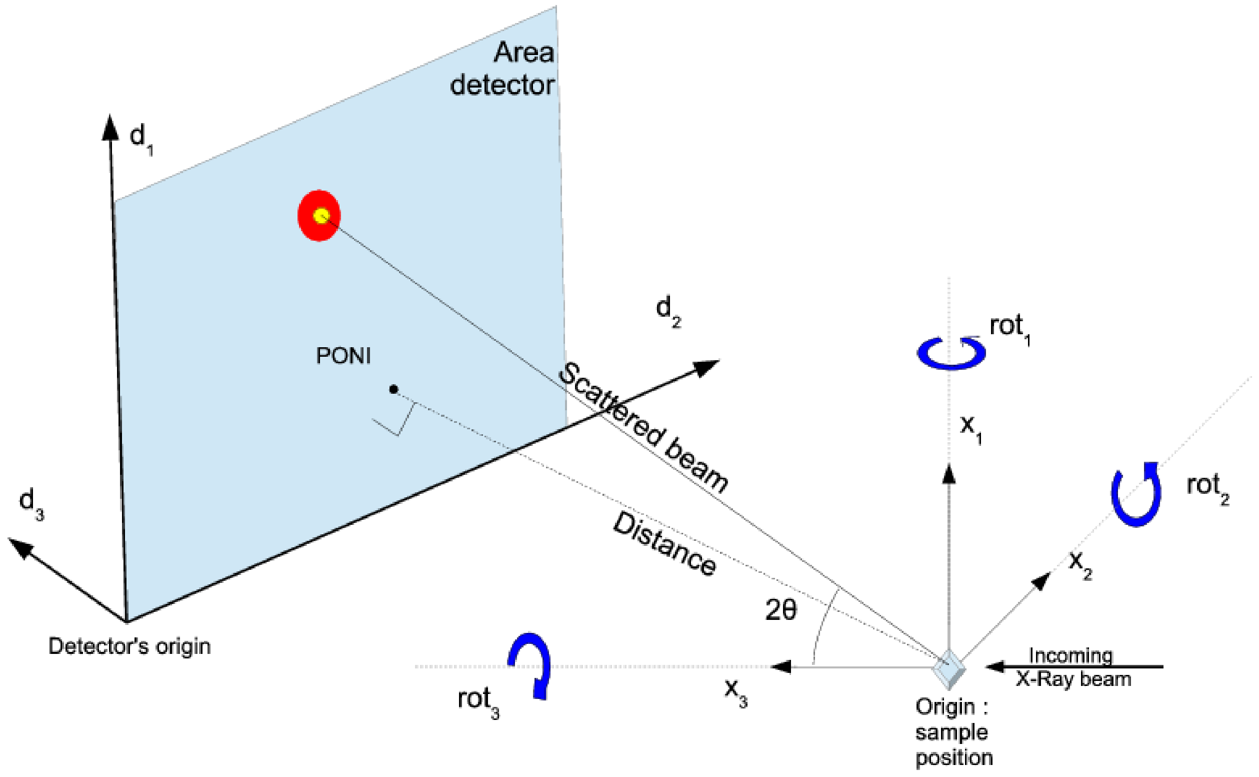


Figure 20 This diagram from Ashiotis (2015) shows the geometry that py321DIAS adopts from pyFAI. By default, pyFAI corrects for 1/cos(2θ).

Since the beam center is not well defined with highly tilted detectors, it is not used directly. In figure 20, axis x_1 and axis x_2 represent the vertical direction y and the horizontal direction x along the detector, respectively. Hence, x_2 points to center of the storage ring. Axis x_3 represents the z direction, which is along the path of the beam and it is defined to be orthogonal, while together x_1, x_2 and x_3 is a direct orientation. This places the sample at a position that is normally less than z=0. The orthogonal projection of the origin on the detector surface is called the PONI, since the center is defined as the point from where the detector normal points to the sample. For non-planar detectors, the point of normal incidence (PONI) is defined in the plan z=0 within the detector coordinate system. If the detector axes don't match, a satisfactory raster orientation must be chosen (Ashiotis 2015). The PONI generally differs from the position of the primary incident beam on the detector and both points are only identical when the detector is perpendicular to the incident beam.

Rot1, rot2 and rot3 are rotations along the x_1, x_2 and x_3 axis, which are always expressed in radians. Because of the axial symmetry of the Debye-Scherrer cones, rot3 cannot be optimized, but it can be adjusted manually to deal with certain situations. The detector is considered to be in transmission mode when all rotations are zero. The point of normal incidence and the distance between sample and detector are kept constant during the rotations. The corresponding rotation matrices are defined as:

$$rot_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(r_1) & -sin(r_1) \\ 0 & sin(r_1) & cos(r_1) \end{pmatrix} \quad \text{(Eq. 24)}$$

$$rot_2 = \begin{pmatrix} cos(r_2) & 0 & sin(r_2) \\ 0 & 1 & 0 \\ -sin(r_2) & 0 & cos(r_2) \end{pmatrix} \quad \text{(Eq. 25)}$$

$$rot_3 = \begin{pmatrix} cos(r_3) & -sin(r_3) & 0 \\ sin(r_3) & cos(r_3) & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{(Eq. 26)}$$

For the standard configuration, the lab coordinate R of a detector pixel is:

$$R = \begin{pmatrix} D_{poni1} \\ D_{poni2} \\ -D \end{pmatrix} \quad \text{(Eq. 27)}$$

D_{poni1} and D_{poni2} are the distances on the detector which are relative to point of normal incidence and thus represents x and y distances. The last component -D is the sample to detector distance, which represents the position of the PONI relative to the sample. Thus, the lab coordinate R_{rot} of a rotated detector can be given by R_{rot} = (rot_1)(rot_2)(rot_3)R. Due to this, the scattering angle 2θ and the azimuth of the scattered beam can be found by expressing R_{rot} in polar coordinates.

In Python notation the coordinates of the PONI is as follows:

```
PONI = R.{0,0,D}

PONI = [D*(cos(rot1)*cos(rot3)*sin(rot2) + sin(rot1)*sin(rot3)),
        D*(-(cos(rot3)*sin(rot1)) + cos(rot1)*sin(rot2)*sin(rot3)),D*cos(rot1)*cos(rot2)]
```

When the detector is at z=D, any pixel on detector at coordinate (D1, D2) is measured in meters.

```
P={D1,D2,D}
```

Hence, any rotated pixel on the detector in regards to the x1, x2 and x3 axes can be described as:

```
R.P = [r1, r2, r3] where r1, r2, and r3 are defined below:

r1 = R.P.x1 = D1*cos(rot2)*cos(rot3) + D2*(cos(rot3)*sin(rot1)*sin(rot2) -
    cos(rot1)*sin(rot3)) + D*(cos(rot1)*cos(rot3)*sin(rot2) + sin(rot1)*sin(rot3))
r2 = R.P.x2 = D1*cos(rot2)*sin(rot3)  + D2*(cos(rot1)*cos(rot3) +
    sin(rot1)*sin(rot2)*sin(rot3)) + D*(-(cos(rot3)*sin (rot1)) + cos(rot1)*sin(rot2)*sin(rot3))
r3 = R.P.x3 = D2*cos(rot2)*sin(rot1) - D1*sin(rot2) + D*cos(rot1)*cos(rot2)
```

The sample distance (origin) to detector point (D1,D2) would be:

```
|R.P| = sqrt(pow(Abs(D*cos(rot1)*cos(rot2) + D2*cos(rot2)*sin(rot1) - D1*sin(rot2)),2) +
    pow(Abs(D1*cos(rot2)*cos(rot3) + D2*(cos(rot3)*sin(rot1)*sin(rot2) -
    cos(rot1)*sin(rot3)) +  D*(cos(rot1)*cos(rot3)*sin(rot2) + sin(rot1)*sin(rot3))),2)  +
    pow(Abs(D1*cos(rot2)*sin(rot3) + D*(-(cos(rot3)*sin(rot1)) +
    cos(rot1)*sin(rot2)*sin(rot3)) + D2*(cos(rot1)*cos(rot3) +
    sin(rot1)*sin(rot2)*sin(rot3))),2))
```

cos(2θ) is defined as the R.P component along x3 over the distance from origin to data point |R.P|. Thus,

```
θ = ArcCos [-(R.P).x3/|R.P|]

θ = Arccos((-(D*cos(rot1)*cos(rot2)) - D2*cos(rot2)*sin(rot1) + D1*sin(rot2))/
    sqrt(pow(Abs(D*cos(rot1)*cos(rot2) + D2*cos(rot2)*sin(rot1) - D1*sin(rot2)),2) +
    pow(Abs(D1*cos(rot2)*cos(rot3) + D2*(cos(rot3)*sin(rot1)*sin(rot2) -
    cos(rot1)*sin(rot3)) + D*(cos(rot1)*cos(rot3)*sin(rot2) + sin(rot1)*sin(rot3))),2) +
    pow(Abs(D1*cos(rot2)*sin(rot3) + D*(-(cos(rot3)*sin(rot1)) +
    cos(rot1)*sin(rot2)*sin(rot3)) + D2*(cos(rot1)*cos(rot3) +
    sin(rot1)*sin(rot2)*sin(rot3))),2)))
```

Finally, tan(2θ) is defined as sqrt(r1**2 + r2**2) / r3 so, θ = ArcTan2 [sqrt(r1**2 + r2**2)/ r3].

The slightly modified formulas on page 33 are from Kieffer and he states that getting 2θ from its tangeant seems to be more accurate around the beam stop when very distant from the sample and this method is faster by about 25 percent (Kieffer 2014). Therefore, both methods are presently used interchangeably.

For FIT2D, the diffraction angle for individual pixels (x, y) can be computed by:

$$2\vartheta = \arctan\sqrt{\frac{((x-x_0)cos\beta+(y-y_0)sin\beta)^2\ cos^2\varphi+(-(x-x_0)sin\beta+(y-y_0)cos\beta)^2}{(d+\left((x-x_0)cos\beta+(y-y_0)sin\beta\right)sin\varphi)^2}} \quad \text{(Eq. 28)}$$

The cone rotated around the y axis may also be rotated around the z-axis by an angle β.

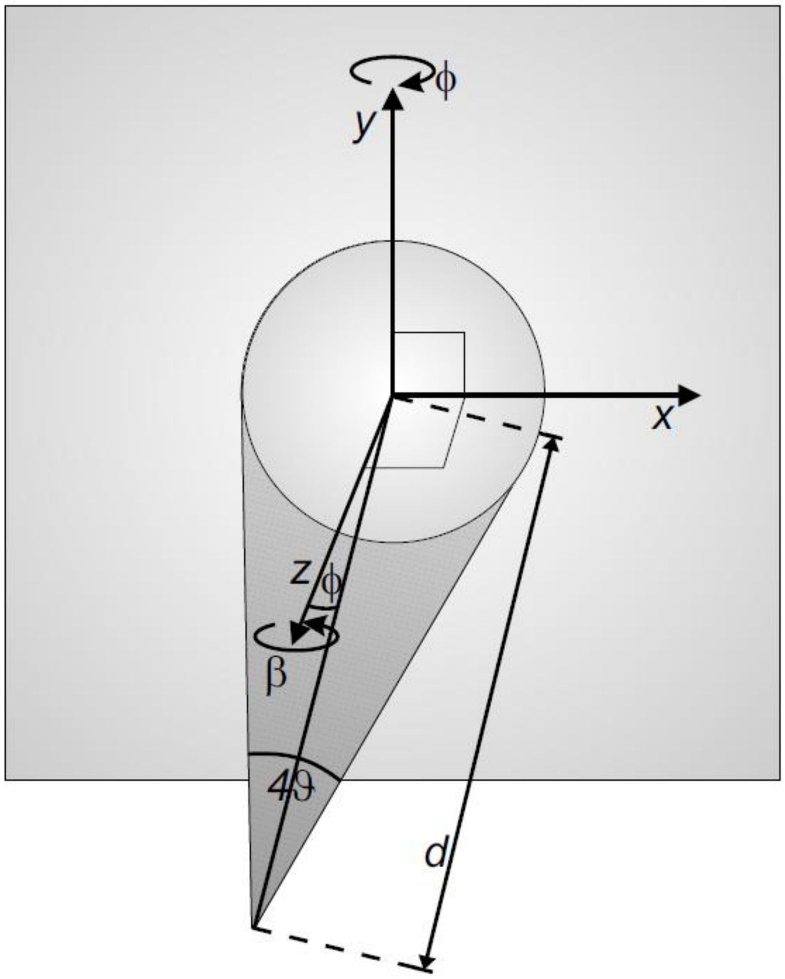


Figure 21: This figure from Vogel (2001) illustrates the orientation for the geometry calculation for FIT2D. FIT2D corrects for 1/cos^3(2θ).

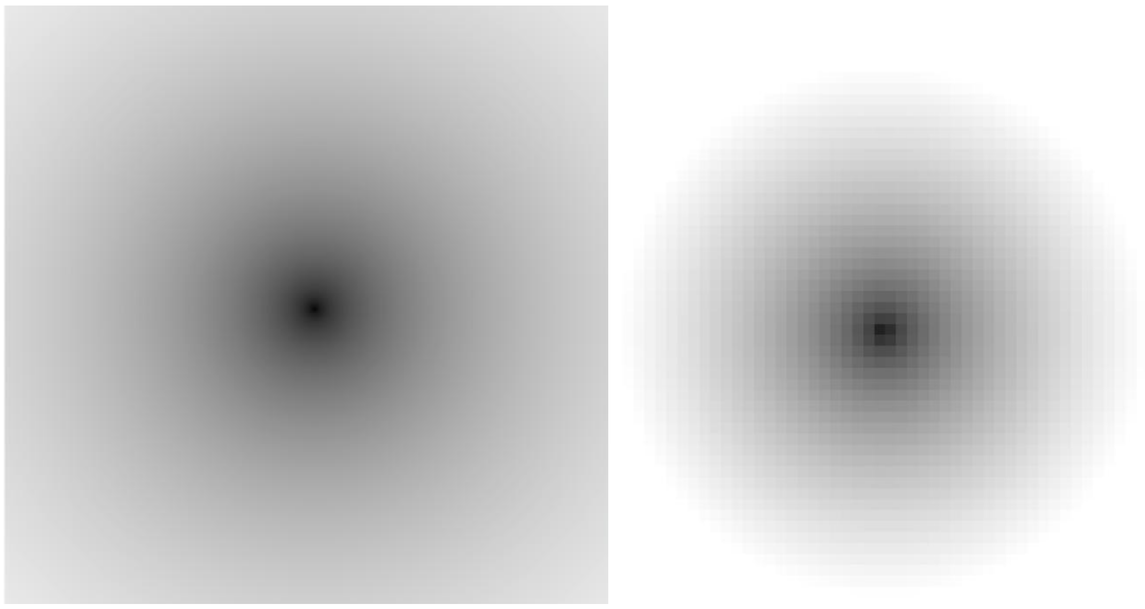


Figure 22: Displayed is how py321DIAS shows the calculated 2θ geometry as a .tif file image. The image on the left, which is the true correct geometry image, was calculated in radians and the image on the right, which is a blown up version of its actually dot size, was computed in degrees.

It was found that if Hammersley's geometry formula is used like in Beyond FIT2D, there is on average a 0.03% variance between the computed 2θ values compared to pyFAI's FIT2D geometry. The geometry looks like the images above depending on the units that are chosen to display them. In python, theta values are by default in radians, which is how Python is programmed to read in and output values for trigonometric calculations. Therefore, its default graphical display values coincide with radian outputs by default.

2.2.4 Polarization Correction

Following next is the polarization correction. The polarization factor is determined by the intensity relations of the horizontally and vertically polarized synchrotron radiation and the monochromator angle. The polarization correction has little or no azimuthal variation if the polarization factor is close to 0. Hence, for this case the phase angle becomes irrelevant. For values that are close to 1, which is generally the case for synchrotron radiation, there is a very strong azimuthal dependence and acquiring the correct polarization phase angle becomes paramount (Hinrichsen 2007).

In addition, the sample can serve as a polarizer owed to dichroism and the effect on intensity varies based on the disparity in the polarization angle of the source and sample. The effect on intensities can be corrected based on the distance from the X-ray beam center. The equation below is the formula for polarization (P) and f is the polarization rate orthogonal to the diffraction plane.

$$P = \frac{1 - A\cos^2(2\theta)}{1 + A} \qquad and \qquad A = \frac{1 - f}{1 + f} \qquad \text{(Eq. 29)}$$

Unpolarized intensities would have a f of zero, and synchrotron plane polarized light would have an f of approximately one. The polarization information is generally provided in the dataset.

2.2.5 Azimuthal Integration

After the necessary corrections have been addressed, integrating the data follows. Azimuthal integration is the process of creating 1D plots by reducing diffraction images taken from area detectors. Dioptas (Prescher 2015), which is a new innovative Python based software for rapid data processing and exploration of large amounts of 2D area detector X-ray diffraction data, was used for data testing and data comparison. Py321DIAS's output data can easily be ran into Dioptas or through pyFAI for quick integration.

2.2.6 Statistical 2θ Bin Analysis and Automatic Masking

For high pressure synchrotron X-ray powered diffraction data, there is a vast amount of low intensity pixels and a minute number of high intensity pixels. Therefore, the intensity data has been found to be well described by probability distribution functions such as Poisson (Chall 2000), Gaussian (Chall 2000, Hinrichsen 2007 and 2009), Log Normal (Fewster 2014, Hinrichsen 2007 and 2009) and the Pareto and Normal Pareto (Hinrichsen 2007 and 2009). In py321DIAS, the data will be statistically analyzed based on the user specified distribution filtration method. The data will then be assumed to be best distributed and outlier intensities, which are outside a chosen range of the standard deviation will be automatically filtered out. Afterwards, it shall be normalized to correct for variance in the amount of pixels contributing to a certain measurement.

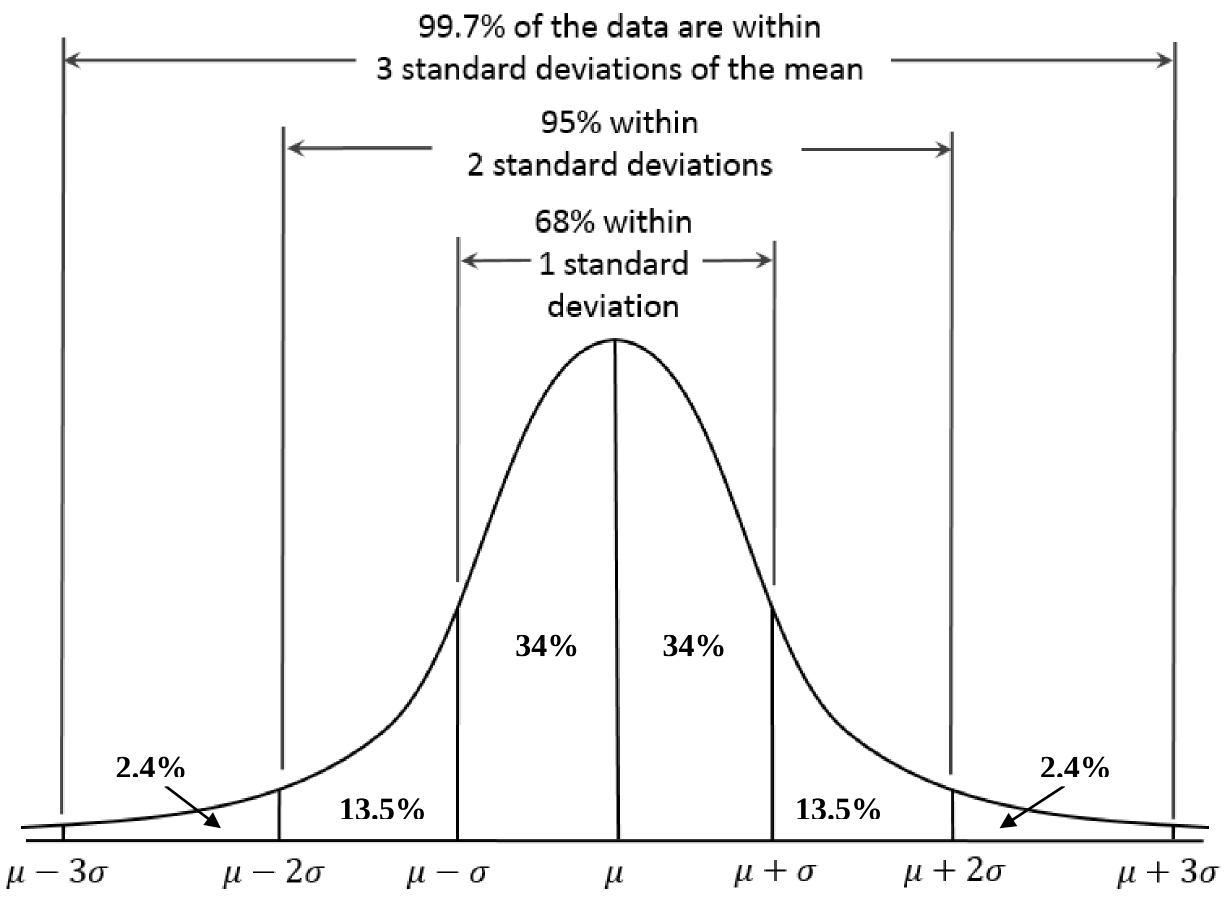


Figure 23: This diagram illustrates the method for filtering out unwanted data, where a statistical distribution analysis will be used to mask pixels that are outside a certain number of standard deviations from the mean.

Chapter 3: Results and Discussion

3.1 Results of Adding, Averaging and Subtracting Data

Py321DIAS was utilized to sum, average and subtract a set of diffraction data images, which was read in and outputted as .tif files using FabIO. Averaging the intensity data was done for comparison with the .edf files that were produced via the pyFAI-average module, which is a tool that was made for averaging a set of dark current images using a mean or median filter. It was found that our program was able to produce the same intensity results and has the capability of reading in and saving outputs in other formats such as .txt using NumPy (Oliphant 2007) or .edf via FabIO. As stated before, Dioptas was used for quick data integration, data testing and data comparison and because it uses pyFAI to integrate the data, while proving to give results that are comparable to FIT2D. Hence, it is the main standard for verifying the validity of our programs integration and regrouping methods. Figures 24 through 26, illustrates the combined usability of both programs.

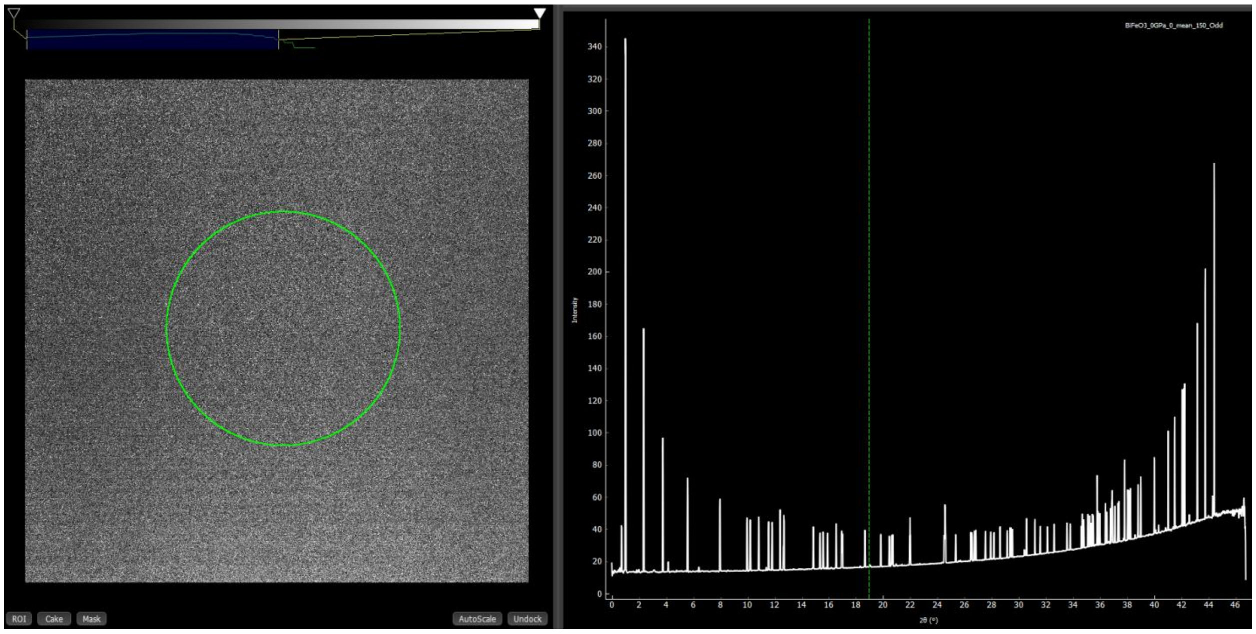


Figure 24: Averaged dark images were obained using py321DIAS and the 2D and 1D plots were produced using Dioptas. Intensities along the y axis range from about 10 to 350 with 2θ from 0 to 47.

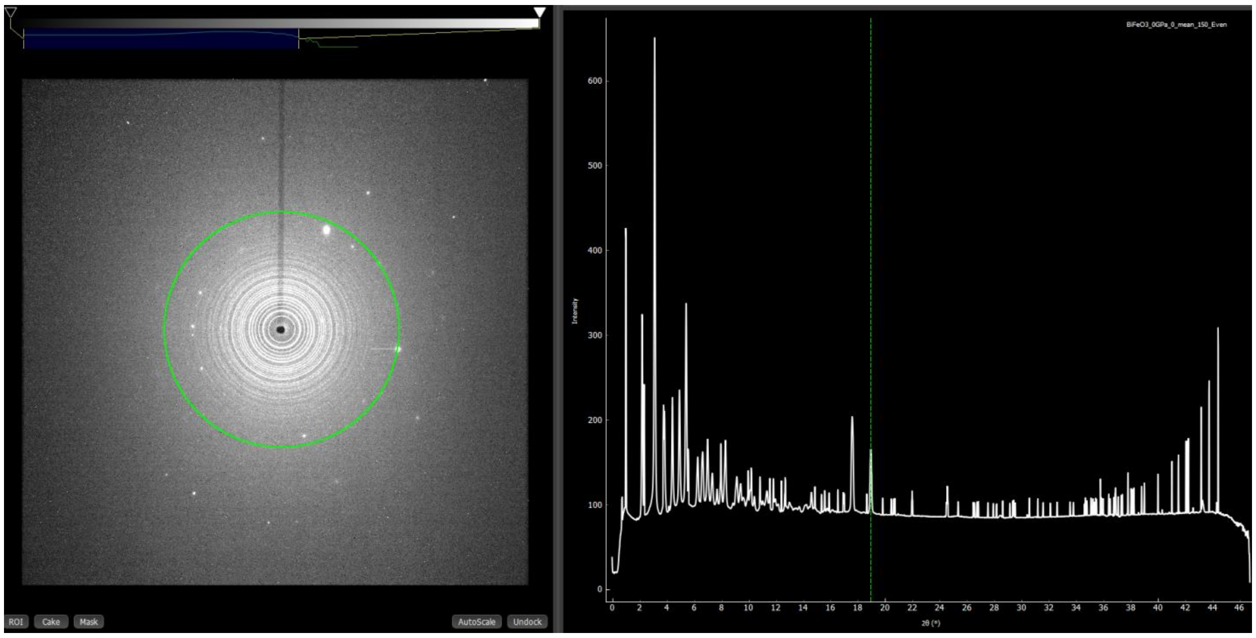


Figure 25: Averaged data images were obtained using py321DIAS and loaded into Dioptas to generate the 2D and 1D plots. Intensities range from about 10 to 650 and 2θ is from 0 to 47. The Bragg peak at 18.953° is highlighted and corresponds to the green Debye Scherrer Ring.

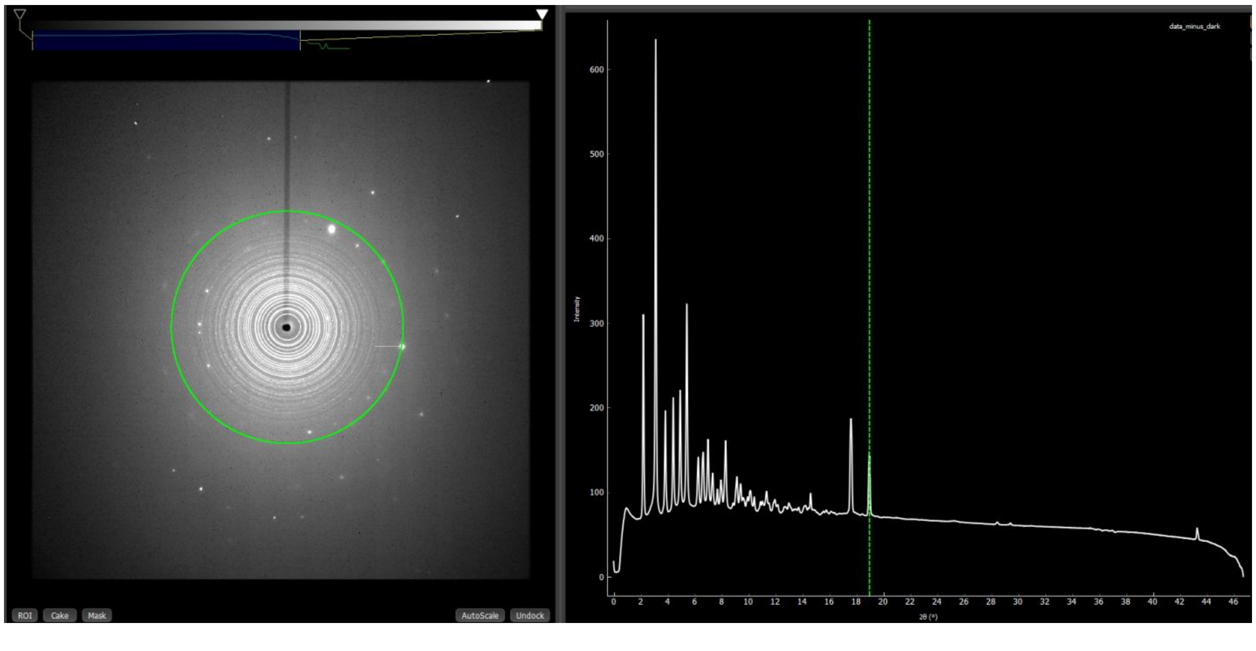


Figure 26: The averaged data minus dark image that was obtained by py321DIAS was loaded into Dioptas to create the 2D and 1D plots. The Bragg peak at 18.953° is highlighted and corresponds to the green Debye Scherrer Ring. Intensities range from about 10 to 675 and 2θ ranges from 0 to 47.

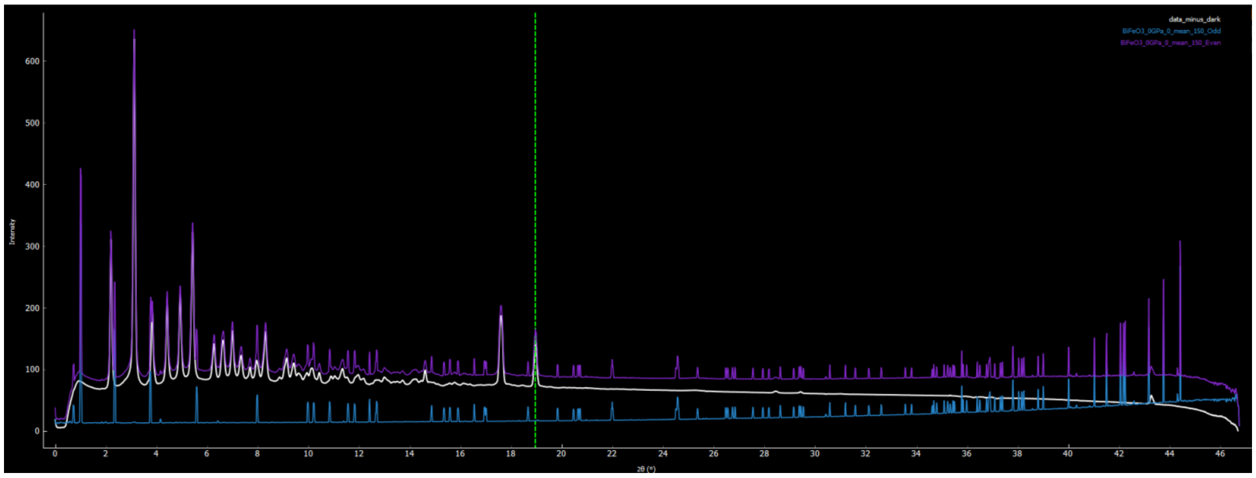
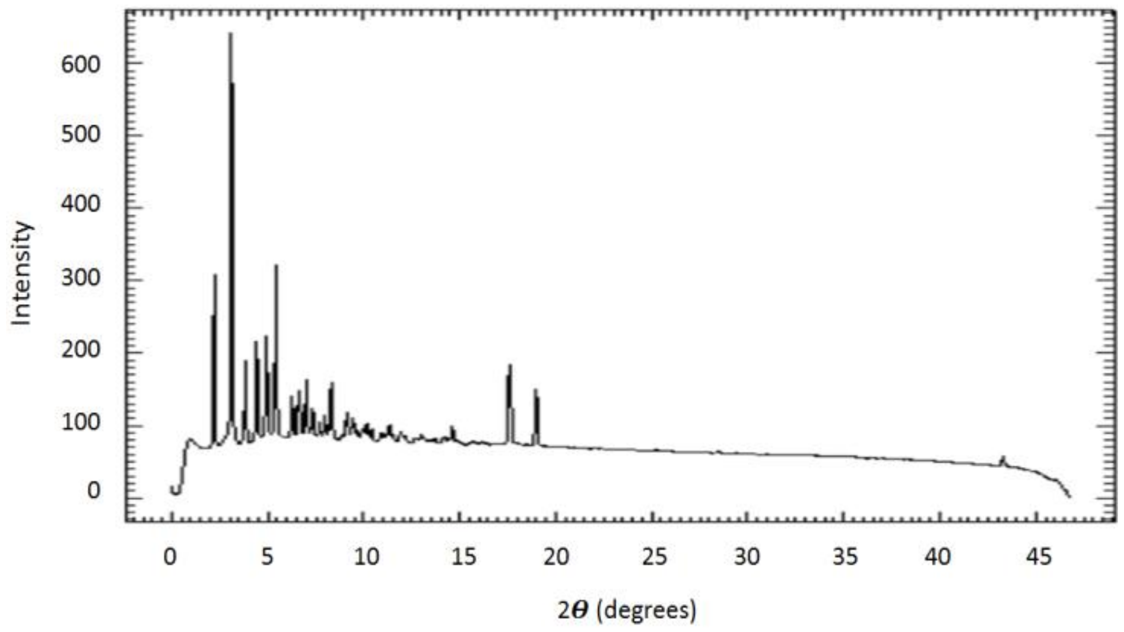


Figure 27: Using Dioptas to overlay the 1D results of figures 24, 25 and 26, a good comparison of the data is shown. Background (summed dark images) is overlaid in blue, data with background is overlaid in purple and data minus background is plotted in white.

3.2 Geometry and Integration Results

After adding and subtracting data, Dioptas was used to create a .poni file, which contained the correct geometry parameter format for pyFAI usage within py321DIAS. Using our program's module for creating 1D and 2D plots, which also incorporates the corrections that were discussed in previous sections, the figure to the bottom, in figure 28, was produced by implementing pyFAI's splitBBox method. It was found that the 2θ values correspond very well with those in Dioptas and are close to the values obtained from FIT2D, which can be seen in the top image, in figure 28. Therefore, the validity of our integration method is established and proven to be highly effective. Other pyFAI methods for acquiring a 1D plot were sort after and tested, and they all produced the same or very similar 1D plots. Figure 29 shows the 2D plots that were generated from the 1D plot (in figure 28) using our program.

In addition, it should be noted that Fit2D automatically flips images depending on their file-format. During the initial testing stages of our program, when simply using pyFAI's splitBBox method alone, there was a complete intensity mismatch which made the diffraction pattern useless. Kieffer found that this flipping effect occurs for Tiff and Mar345 images (Kieffer 2014). The 1D pattern in figure 30 reveals how severe this effect is, if not corrected. To obtain correct results for the 1D and 2D plots as well as for the statistical results in the next section, the numpy.flipud was used on the averaged data minus dark Tiff file.



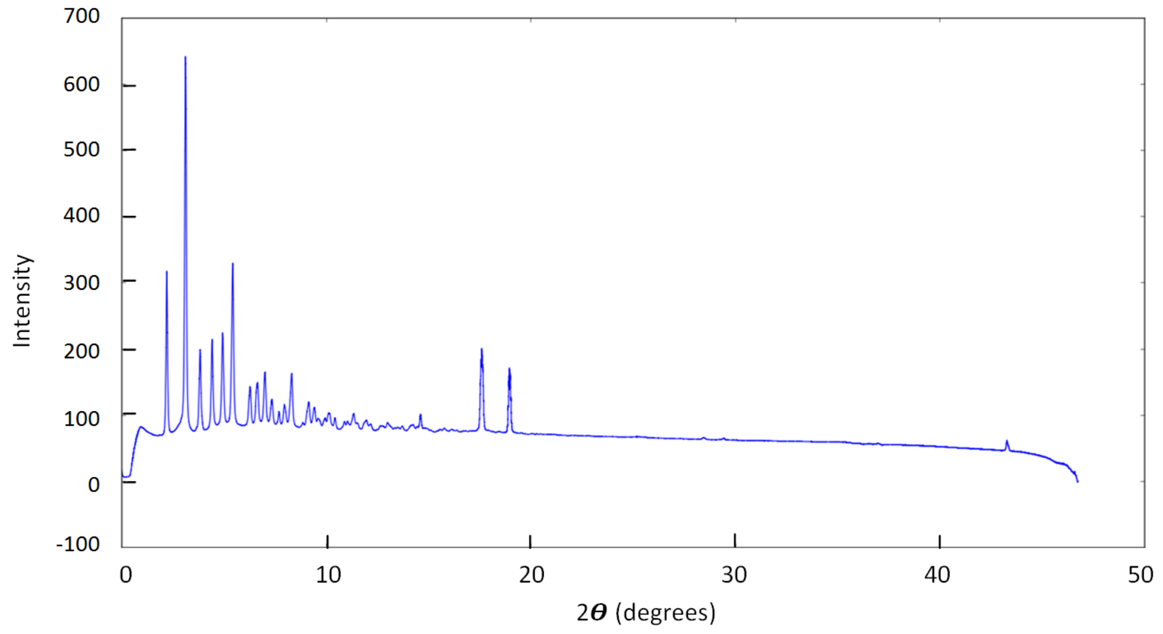


Figure 28: The 1D plot on top was produced using FIT2D to integrate the averaged data minus dark images to check the results from Dioptas and they match up well when compared. The plot on the bottom was created by py321DIAS, which utilized pyFAI's splitBBox method and 'FIT2D.poni' file method. The 2θ values are very accurate and hence, the intensities are integrated well within the diffraction pattern. Matplotlib was used to display the image and save it as a .png file.

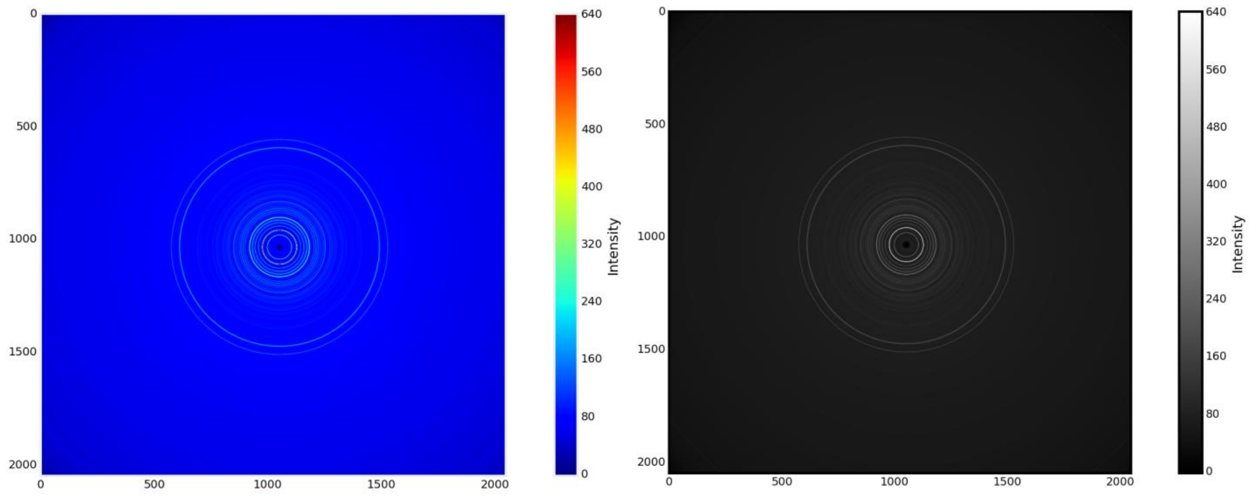


Figure 29: A 2048 by 2048 pixel colored 2D diffraction plot and grayscale 2D image were constructed using the same data as in figure 28. The variation of intensities are shown on the right of each image. The high intensities are mostly located near the center.

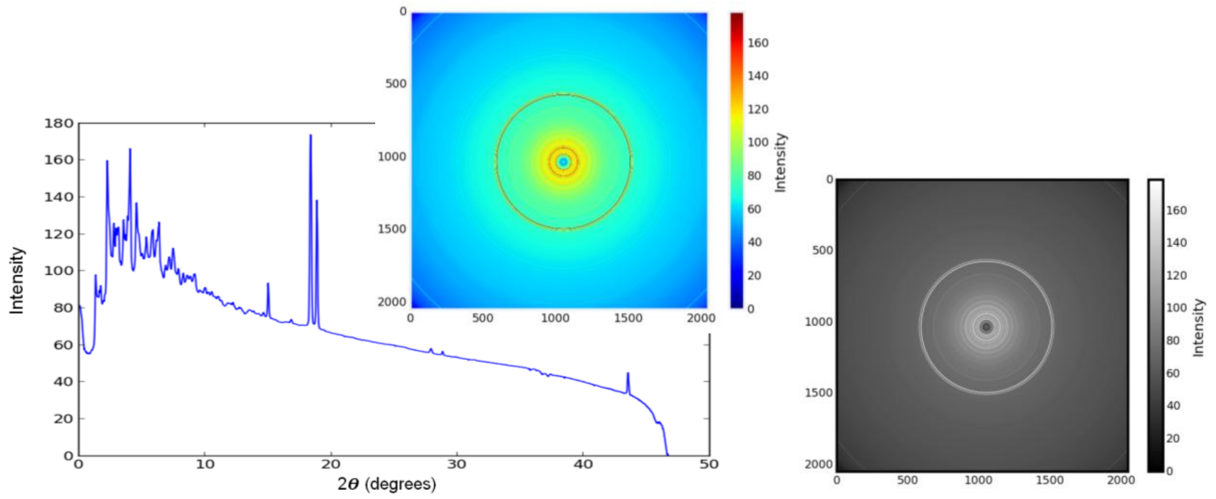


Figure 30: An incorrect lD pattern, colored 2D diffraction image and grayscale 2D diffraction image of the averaged data occurred due to not implementing numpy.flipud on the data minus dark Tiff file. Though the 2D images may look more colorful than the correct 2D plots, they are completely wrong and not useable since the integrated intensity data drastically suffers.

3.3 Statistical Data Analysis

After acquiring the correct geometry and a reasonable 1D diffraction pattern, the statistical analysis comes next. This part of the program is used for viewing the data for the purpose of statistical data testing. First, a scatter plot of intensity data within a certain 2θ range is created as shown in figure 31. The approximate value of the mean intensity in comparison to all the intensity values for bins 18.57 and 18.59 are soon and the center of each circle is closest to the mean value. The spread of each dataset can be computed as the variance. In addition, since the matplotlib library is used for plotting, the user can zoom in on any desirable sections of interest within the range. This was implemented for easy viewing of the data, so that finding a bin size that would be appropriate for statistically analyzing a specific set of data could be simple. A distribution method can then be used to analyze a desired bin.

Furthermore, the program allows for testing the effect that different bin sizes may have on the statistical fit as soon in figure 31. However, a bin size of 0.02 was used for the remainder of this project to compare results with Beyond FIT2D. The frequency verses intensity diagrams (figures 31 and 32) show how many pixels belong to individual intensity values that are contained within a certain bin. Compared to the Poisson distribution, the Gaussian distribution was found to better fit these bins and was used for fitting each bin. A statistical distribution analysis can then be used to mask pixels that are outside a certain number of standard deviations from the mean. These masked pixels represent pixels of very low or extremely high intensities, which symbolize background data or high intensity diamond reflections that normally interferes with obtaining accurate intensity uncertainties that are necessary for obtaining a reasonable goodness of fit. It was found that over masking occurs when using the Gaussian filtration method and hence for the following chapter, the Poisson filtration method was used.

It should be noted that the initial fitting of certain sections of the data cannot be fitted well with the use of a distribution function due too extreme outliers. This can easy be seen by viewing the scatter plot in figure 33. Regions like these can cause a fair amount of wanted data to be masked. Therefore, it is suggest that a higher number of standard deviations from the mean be chosen to reduce the amount of wanted data that may be filtered out due to the analysis. However, it may be possible to create a mean correction method to deal with this issue. It will be a method that can automatically assess how many data points are far away from the main bulk of data and slightly adjust the mean by a certain percentage so that it is closer to the mean that corresponds to the main bulk of data. This will also allow the mean for bins that aren't greatly affected by extreme outliers to relatively remain the same and therefore, the filtration method would still be effective in these bins and become more effective in bins that have outliers.

Besides the mean correction method, another method can be used to retain a greater amount of wanted data. Though it has not been implemented yet, an automatic filtration method for choosing a certain scalar range for multiplying the standard deviation can be implemented. For example, the program will start from $s_{max}\sigma$ and end at $s_{min}\sigma$, while decreasing the scalar (s) by one (s-1) after every calculation. This will allow the user to start a filtration method that begins at a max number of standard deviations from the mean for the purpose of keeping wanted data. A small number of outliers will initially be masked, but it will improve the mean for the next filtration step. This will continue until $s_{min}\sigma$ is reached, while improving the mean every time for the next proceeding step. Hence, this should minimize data lose and remove a descent amount of unwanted data. However, a method like this may require a large amount of computational time depending on the scalar range that is chosen.

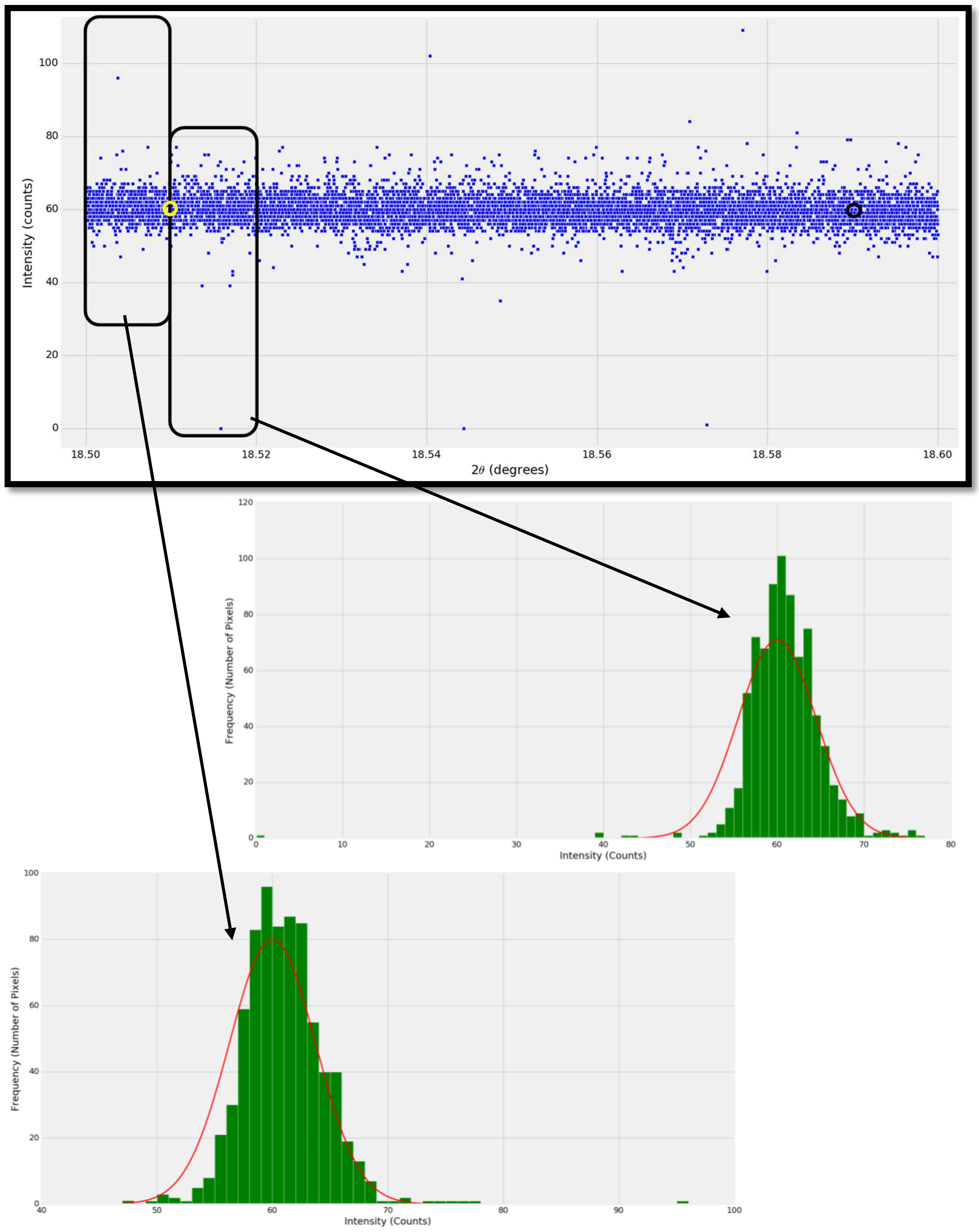
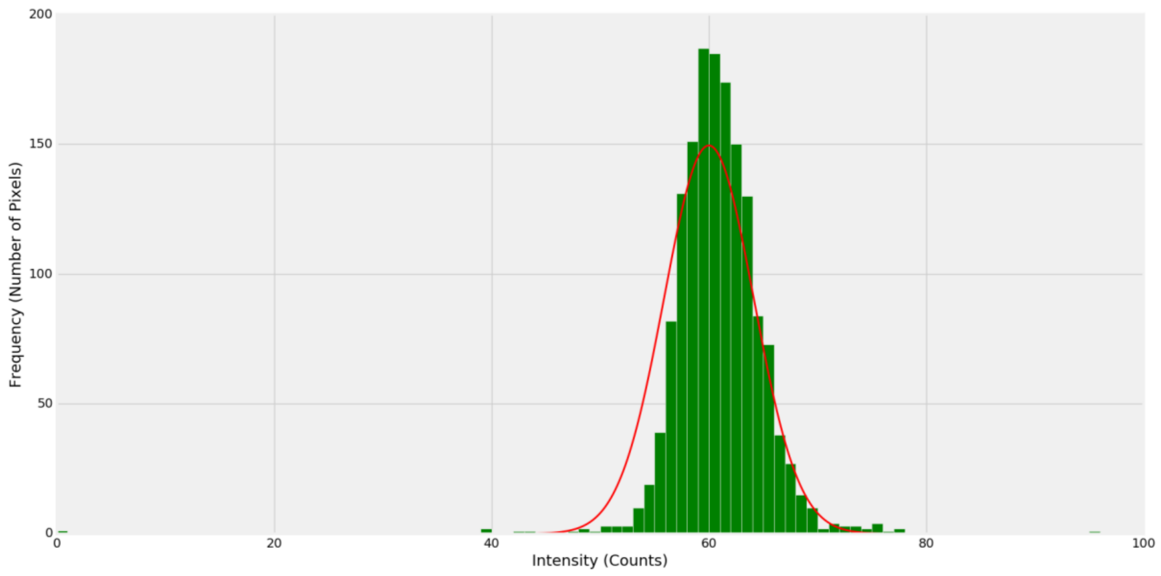


Figure 31: The 2θ bins 18.51 and 18.59 with means 60 (inside yellow circle) and 59 (inside black circle), respectively. The boxes show bins 18.505 and 18.515 and they are half the bin size of the other bins. Histograms for bins 18.505 and 18.515 are fitted with at Gaussian distribution.



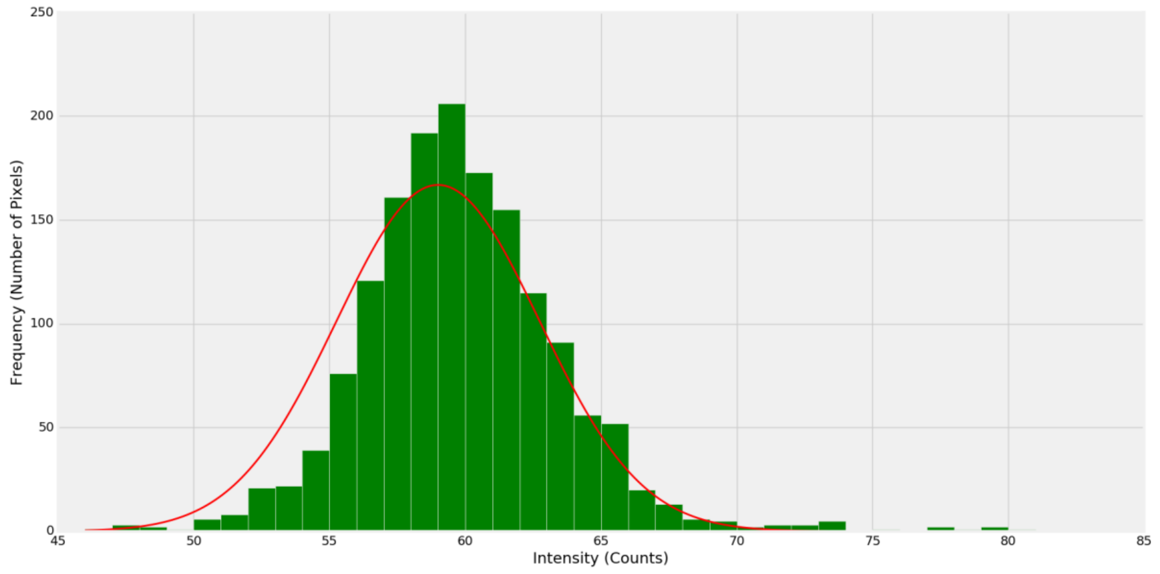


Figure 32: The histograms for bins 18.51 and 18.59 are both fitted by a Gaussian curve.

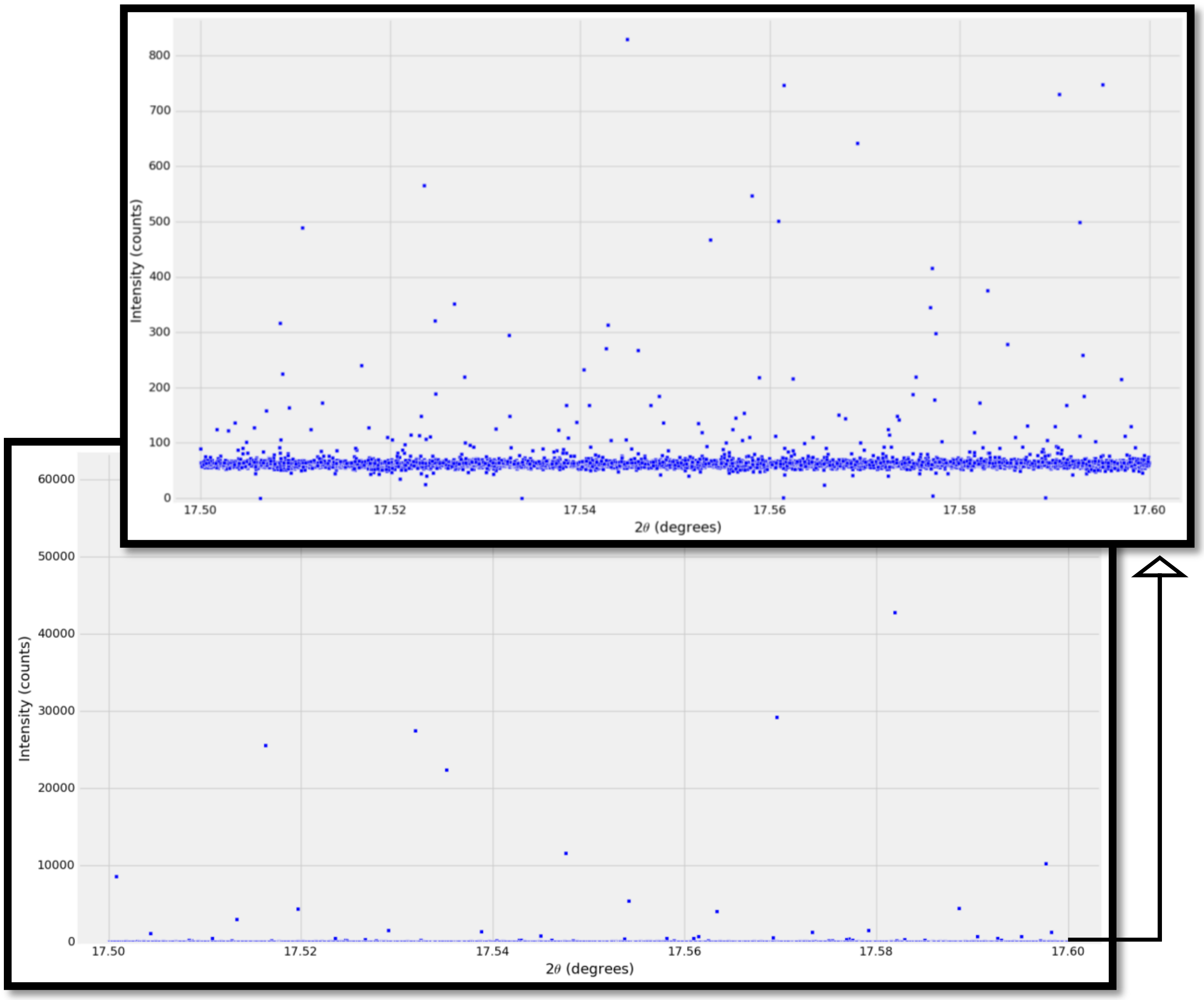


Figure 33: Bins 17.51, 17.53, 17.55, 17.57 and 17.59 all have extreme outliers that will cause the true mean for the main portion of the dataset to be significantly different. A closer look (top figure) reveals that the true mean for the main bulk of data within these bins is between 60 and 75. The current means for these bins are 92, 100, 120, 134 and 146, respectively.

3.4 Automatic Data Masking

Finally, due to the statistical data analysis, an automating masking method is producible for the purpose of removing unwanted data, while adding more meaning to the value of the mean and intensity estimates. This should ultimately allow for better use of structural determination methods. This module makes use of the statistical filtration method that was used in Beyond FIT2D (Sim 2014). The program has made important fixes to the code for optimal results. The images below reveal how the number of standard deviations from the mean affects the amount of masking that is preformed. It does an effective job of removing high intense spots that are caused by diamond reflections, even at a higher number of standard deviations from the mean. Therefore, the proceeding images demonstrate how beneficial our automatic masking method can be. If these high intensities are masked at higher numbers of standard deviations from the mean then, the scalar range filtration method that was discussed in the previous section would be even more effective. It should also be noted that py321DIAS currently flips the 2D image after masking and therefore, Diaptos was used to unflip the images in order to obtain the correct 1D patterns in figure 36.

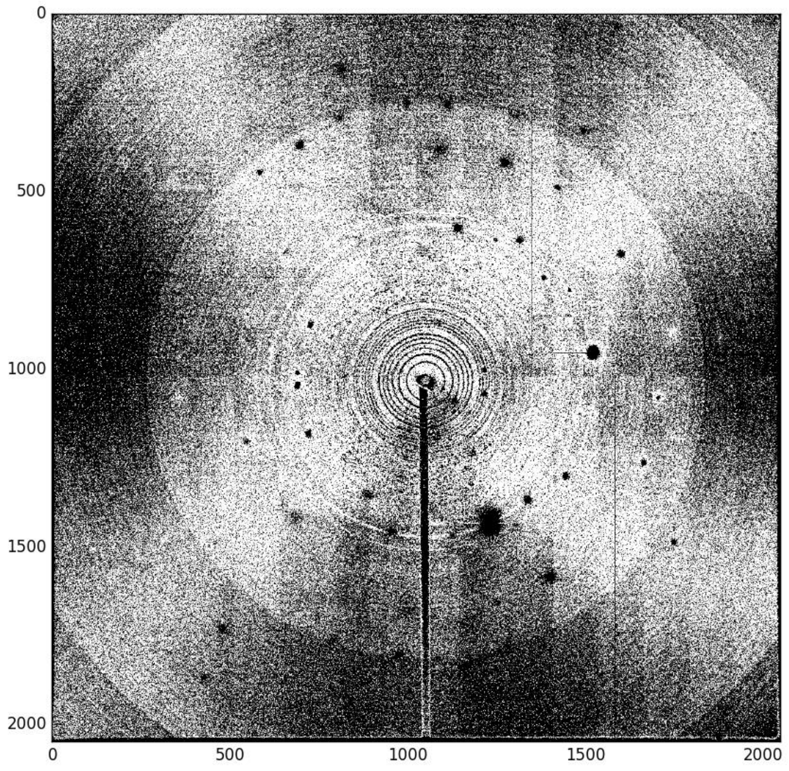
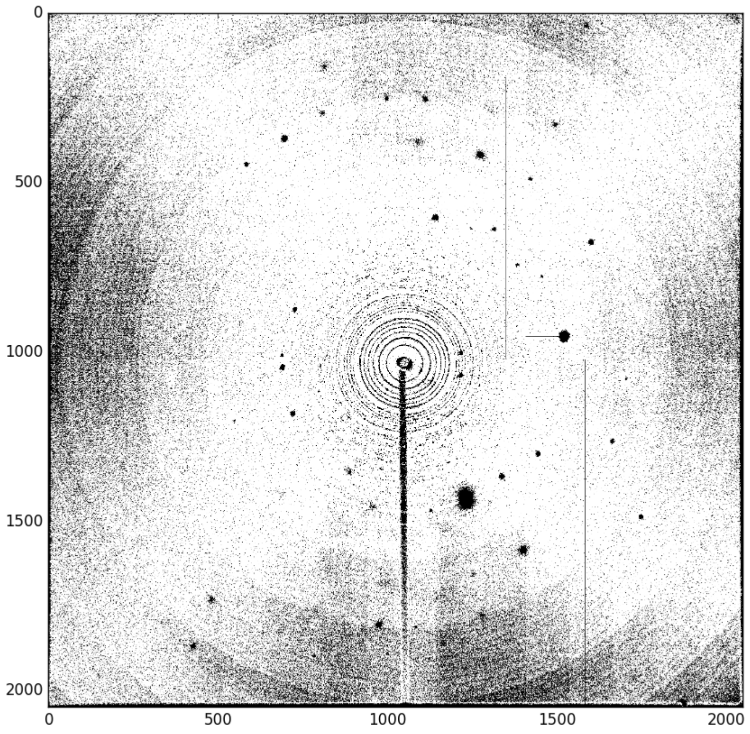


Figure 34: This image shows the amount of pixels that were masked, which was 45.43%, after performing a 1 standard deviation Poisson mask with a 2θ step size of 0.02°. Black pixels represent masked data which were given a value of zero, while white pixels represent pixel intensity information that were given a value of 255.



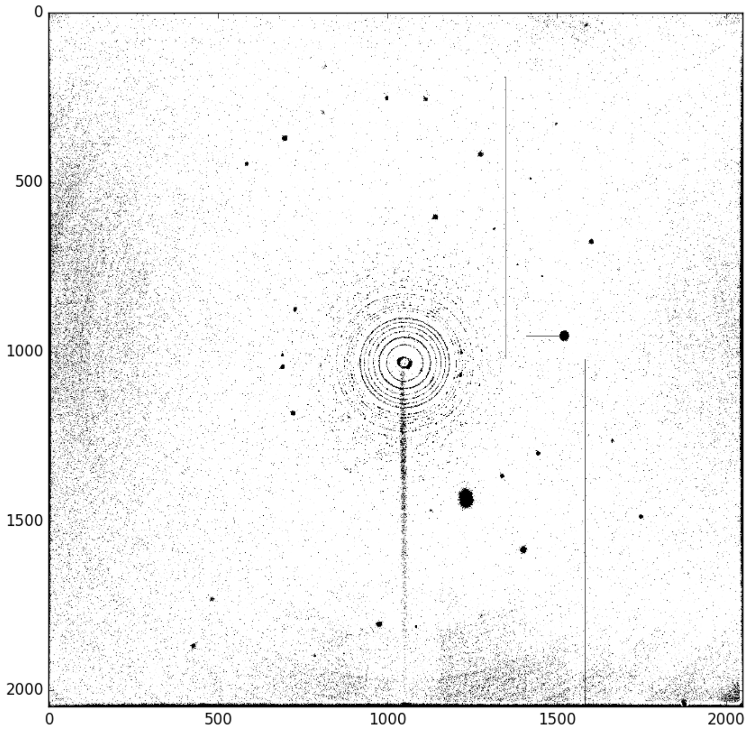
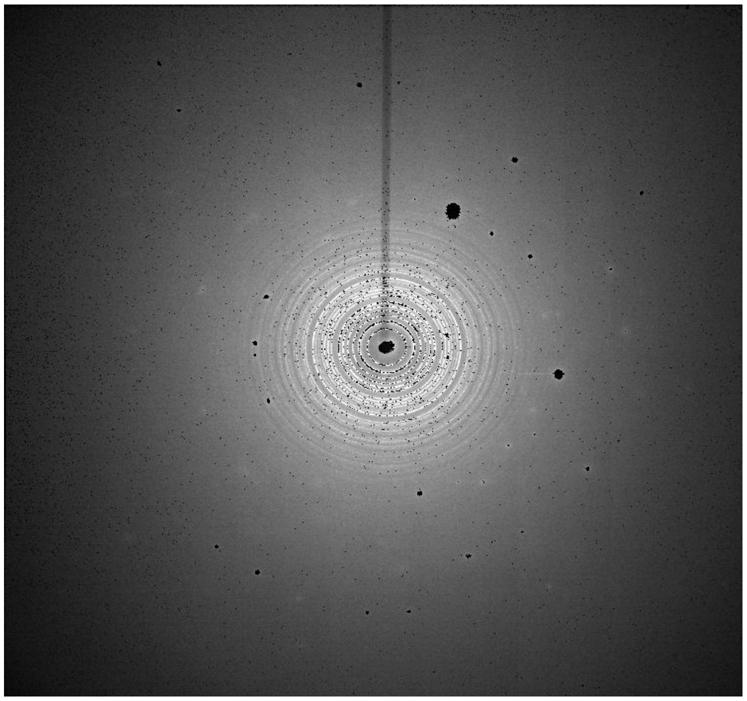


Figure 35: The images above show the amount of pixels that were kept (white pixels = 255) and masked (black pixels = 0) after performing a 2 and 3 standard deviation Poisson mask (top to bottom image, respectively). Bin widths of 0.02° were used and 13.78% and 3.87% of the pixel intensity data was masked, respectively.



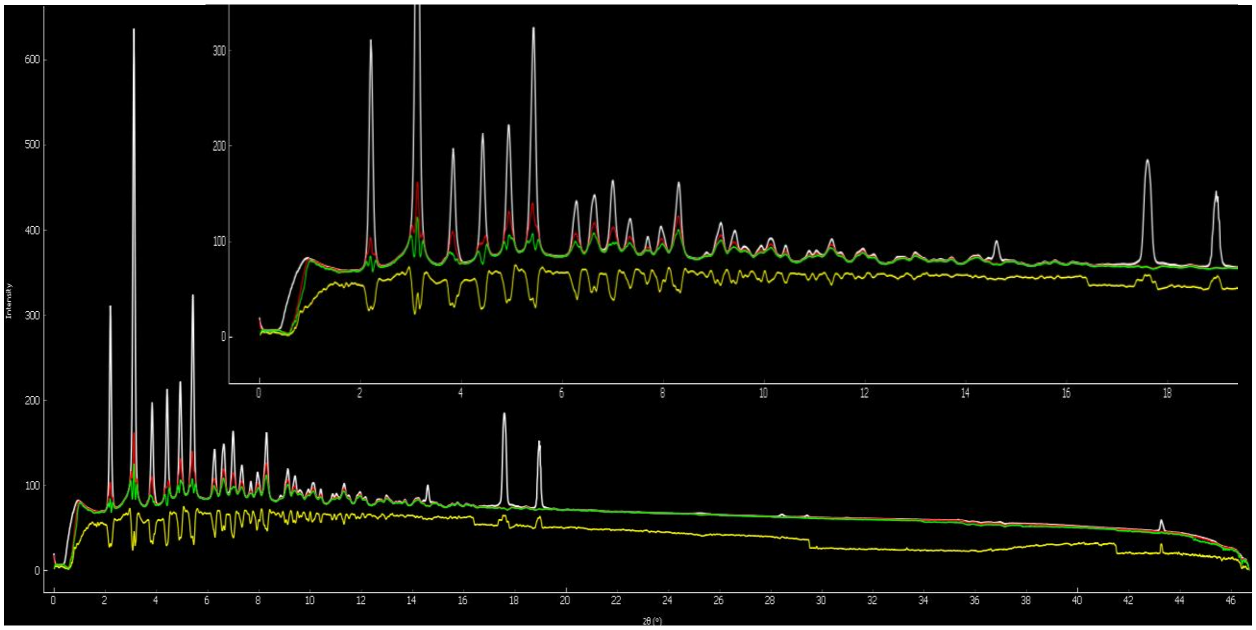


Figure 36: Azimuthal integrated patterns (a full view and close up view at low 2θ angles) for a 1 standard deviation (yellow), 3 standard deviation (green) and 4 standard deviation (red) Poisson mask are shown as 1D plots. Masking at a higher number of standard deviations away from the mean seems to effectively keep the majority of data pixels, while reducing integrated intensity values in a reasonable way. The 2D image on top shows a 4 standard deviation Poisson mask with kept pixels maintaining their original intensities and masked values are set to zero (black). Having kept pixels maintain their original intensity values for each masking method is paramount for obtaining correct 1D plots.

Chapter 4: Conclusion

It is clear that our program does an efficient job of obtaining correct 2θ values and performing azimuthal integration and 2D regrouping, which is on par with FIT2D and Dioptas, due to the use of pyFAI. The 2D and lD images for the masking method demonstrate that at a higher number of standard deviations away from the mean, more wanted data is kept, while unwanted data is still effectively masked and therefore making the automatic masking more meaningful.

Furthermore, this project mainly focused on utilizing the Normal and Poisson distributions. Overall, depending on the bin, a Poisson or Normal distribution could be fitted to the data. In general, the Poisson would work better for bins with lower mean values and the Normal distribution would be more appropriate for higher mean values. Other distributions like the Pareto and Normal-Pareto need to be worked on and will later be fully implemented into the program. The use of these statistical data analysis methods allows for a standardized method of automatic masking to be performed, instead of manual and tedious masking. Pixels that are filtered out by a standard analysis method has shown a dramatic effect on the mean and ultimately should allow for acquiring a better fitting and better estimated uncertainties for structural determination methods like Rietveld. When the mean correction or scalar range filtration methods are implemented, this will help to keep more wanted data, while removing a significant amount of unwanted data, which will in turn add more meaning to the mean as well as estimated uncertainties that are vital for structural refinement.

Moreover, py321DIAS has only been tested on Windows 64bit operating systems using Python 2.7 and Eclipse as the main Python editor. The main dependencies needed for full usage of all py321DIAS modules include NumPy (Oliphant 2007), SciPy (Jones 2001), matplotlib (Hunter 2007), FabIO (Knudsen 2013) and pyFAI (Kieffer 2013 and 2014). This program can also save files as .txt for easy data viewing or .tif or .edf files which are usable in the Dioptas (Prescher 2015) or FIT2D (Hammersley 1996 and 2016) for quick and easy data testing. Py321DIAS software modules will be composed in an open source repository at http://github.com and used for the data acquisition and analysis software package at the XPD beamline at National Synchrotron Light Source II at Brookhaven National Laboratory.

References

Ahn, H. S. "Euler's Equation." *Canada* 2008 Web. 20 Feb. 2016.

Andrault, D., et al. "Solid-Liquid Iron Partitioning in Earth's Deep Mantle." *Nature* 487.7407 (2012): 354

Ashiotis, G., et al. "The Fast Azimuthal Integration Python Library: pyFAI." *Journal of Applied Crystallography* 48 (2015): 510-19.

Baker, L. J. *Synthesis and High-Pressure Structural Studies of* AuX_2 *(X=Al, Ga, In) Compounds.* Thesis. University of Nevada, 2012.

Bassett, W. A. "Diamond Anvil Cell, 50th Birthday." *High Pressure Research* 29.2 (2009): Cp5-186.

Boesecke, P. "Reduction of Two-Dimensional Small- and Wide angle X-ray Scattering Data." *Journal of Applied Crystallography* 40 (2007): S423-S27.

Chall, M., et al. "Estimating Intensity Errors of Powder Diffraction Data from Area Detectors." *High Pressure Research* 17.3-6 (2000): 315-23.

Cockcroft, J. K., et al. "Powder Diffraction on the WEB." *Birkbeck College* 1997. Web. 21 Feb. 2016.

David, W.I. "Powder Diffraction: Theory and Practice." Acta Crystallographia A, 64 (2002).

Dubrovinsky, L., et al. "Implementation of Micro-Ball Nanodiamond Anvils for High-Pressure Studies above 6 Mbar." *Nature Communications* 3 (2012).

Fewster, P. F. "A New Theory for X-ray Diffraction." *Acta Crystallographica A* 70 (2014): 257-82.

Hammersley, A. P., et al. "Two-Dimensional Detector Software: From Real Detector to Idealised Image or Two-Theta Scan." *High Pressure Research* 14.4-6 (1996): 235-48.

Hammersley, A. P. "Fit2d: A Multi-Purpose Data Reduction, Analysis and Visualization Program." *Applied Crystallography* 49 (2016): 646-52.

Heinz, D. L., J. S. Sweeney, and P. Miller. "A Laser-Heating System That Stabilizes and Controls the Temperature - Diamond Anvil Cell Applications." *Review of Scientific Instruments* 62.6 (1991): 1568-75.

Henry, N. F. M., H. Lipson and W.A. Wooster. *The Interpretation of X-Ray Diffraction Photographs*. London: Macmillan, 1951.

Hinrichsen, B., R. E. Dinnebier, and M. Jansen. "On the Intensity Distribution within Debye-Scherrer Rings. What Is Different in High Pressure Experiments? Part Ii: Practical Application." *Zeitschrift für Kristallographie Supplements* 2009.30 (2009): 147-53.

Hinrichsen, B., R. E. Dinnebier, and M. Jansen. " Powder3D IP 0.1 Tutorial ." *Max Planck Institute for Solid State Research*. (2007): 3-19.

Hunter, J. D. "Matplotlib: A 2d Graphics Environment." *Computing in Science & Engineering* 9.3 (2007): 90-95.

Jones, E., T. E. Oliphant and P. Peterson. *SciPy: Open source scientific tools for Python,* 2001.

Kieffer, J., and D. Karkoulis "PyFAI, a Versatile Library for Azimuthal Regrouping." *11th International Conference on Synchrotron Radiation Instrumentation (Sri 2012)* 425 (2013).

Kieffer, J., and G. Ashiotis. "PyFAI, a Python Library for High Performance Azimuthal Integration on GPU." *7th European Conference on Python in Science* (2014): 1-8.

Knudsen, E. B., et al. "Fabio: Easy Access to Two-Dimensional X-Ray Detector Images in Python." *Journal of Applied Crystallography* 46 (2013): 537-39.

Le Bail, A., H. Duroy and J. L. Fourquet. “Ab-initio structure determination of $LiSbWO_6$ by X-ray powder diffraction,” *Materials Research Bulletin* 23:3 (1988): 447–452.

Li, X.Z., "PCED2.0 - A Computer Program for Advanced Simulation of Polycrystalline Electron Diffraction Pattern." University of Nebraska, Ultramicroscopy 110 (2010): 297-304.

Lynch, D. W. "Tantalus, a 240 MeV Dedicated Source of Synchrotron Radiation, 1968-1986." *Journal of Synchrotron Radiation* 4 (1997): 334-43.

Mogilevsky, P., et al., "Evolution of Texture in Rhabdophane-Derived Monazite Coatings." *Journal of the American Ceramic Society* 86.10 (2003): 1767-72.

Norby, P. "Synchrotron Powder Diffraction using Imaging Plates: Crystal Structure Determination and Rietveld Refinement." Journal of Applied Crystallography 30 (1997): 21-30.

Oliphant, T. E. "Python for Scientific Computing." *Computing in Science & Engineering* 9.3 (2007): 10-20.

Pawley, G. S. "Unit-Cell Refinement from Powder Diffraction Scans." *Journal of Applied Crystallography* 14.Dec (1981): 357-61.

Piermarini, G. J. "Diamond Anvil Cell Techniques." *Static Compression of Energetic Materials* (2008): 1-74.

Prescher, C., and V. B. Prakapenka. "Dioptas: A Program for Reduction of Two-Dimensional X-Ray Diffraction Data and Data Exploration." *High Pressure Research* 35.3 (2015): 223-30.

Price, M. J. "The Reduction of Energy Loss Due to Synchrotron Radiation." *Particle Accelerators* 12 (1982): 231-35.

Rietveld, H. M. "A Profile Refinement Method for Nuclear and Magnetic Structures." *Journal of Applied Crystallography* 2 (1969): 65.

Robinson, A. L. "History of Synchrotron Radiation." *Synchrotron Radiation News* 28.8 (2015): 4-9.

Rodriguez-Navarro, A. B., et al. "Automatic Crystal Size Determination in the Micrometer Range from Spotty X-ray Diffraction Rings of Powder Samples." *Journal of the American Ceramic Society* 89.7 (2006): 2232-38.

Rowe, E. M. and F. E. Mills. "Tantalus I: A Dedicated Storage Ring Synchrotron Radiation Source." *Particle Accelerators* 4 (1973): 211-27.

Sims, M. *Beyond FIT2D: Calculating Intensity Errors for Data Analysis of X-ray Synchrotron Powder Diffraction Data*. Thesis. State University of New York at Stony Brook, 2014.

Singh, O., and G. Decker. "Beam Stability at the Advanced Photon Source." *2005 IEEE Particle Accelerator Conference (Pac), Vols 1-4* (2005): 1578-80.

Vogel, S.C. *High-Pressure and Texture Measurements with an Imaging Plate*. Thesis. Kiel, 2001.

Wang, Y. D., et al. "Separating the Recrystallization and Deformation Texture Components by High-Energy X-rays." *Journal of Applied Crystallography* 35 (2002): 684-88.

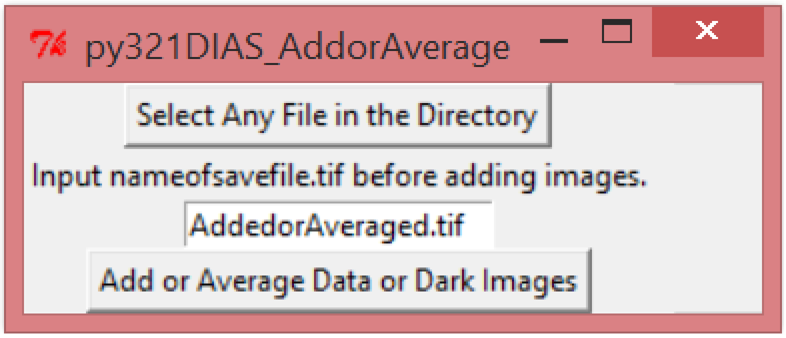
Wille, K. "Synchrotron Radiation Sources." *Reports on Progress in Physics* 54.8 (1991): 1005-68.

Yang, X., P. Juhas, and S. J. L. Billinge. "On the Estimation of Statistical Uncertainties on Powder Diffraction and Small-Angle Scattering Data from Two-Dimensional X-ray Detectors." *Journal of Applied Crystallography* 47 (2014): 1273-83.

Young, R. A. *Introduction to the Rietveld Method*. Oxford, 1995.

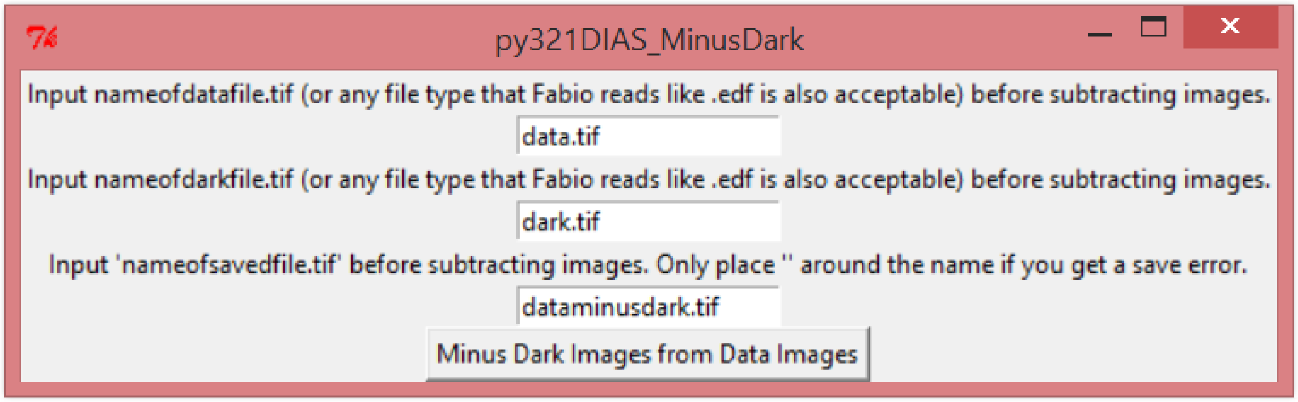
Appendix

Appendix A: Module for Adding or Averaging Data


py321DIAS_AddorAverage
Select Any File in the Directory
Input nameofsavefile.tif before adding images.
AddedorAveraged.tif
Add or Average Data or Dark Images

```
'''
Module: py321DIAS_AddorAverage

Description: Adds or Averages files and saves the summed or averaged image as a .tif, .edf or
.txt file. The saved file(s) are located in: C:\Users\Directory\workspace\PythonProjectFolder

Warning: .txt files makes the program run slower and takes up about 6 times more memory
'''

import Tkinter, tkFileDialog
import os, fnmatch
from tkSimpleDialog import *
import numpy
import fabio
import math
import time

global directoryname, diffimg0, file, data_images_summed, originalpic

def directorylocator():
    global diffimg0, originalpic
    diffimg0 = tkFileDialog.askopenfilename()
    originalpic=diffimg0
    global directoryname
    directoryname=os.path.dirname(unicode(diffimg0))

def addimages():#adds or averages .tif's and saves them as a .tif, .edf or .txt files
    global directoryname, diffimg0, data_images_summed
    start_time = time.time()
    blank = numpy.zeros(4194304, dtype=numpy.int32)
    blank.shape = (2048, 2048)
    data_images_summed= numpy.array(blank,numpy.int32)
    counter=1

    for file in os.listdir(directoryname):
        if fnmatch.fnmatch(file, '*.tif'):
        #if fnmatch.fnmatch(file, '*.edf'):
            arrays=fabio.open(os.path.join(directoryname, file)).data
            data_images_summed += arrays
            counter += 1
        print counter - 1
        print 'Program took', time.time() - start_time, 'seconds to run.'
    denominator = counter - 1
    meanI = data_images_summed/denominator
    Avgtif = fabio.tifimage.tifimage(data=meanI.astype(numpy.int32))
    Avgtif.save('%s' %Entry.get(nameofsummedfile))
    #Avgtif = fabio.edfimage.edfimage(data=data_images_summed.astype(numpy.int32))
    #Avgtif.save('%s' %Entry.get(nameofsummedfile)) #if used put .edf instead of .tif when
saving
    #Summedtif = fabio.tifimage.tifimage(data=data_images_summed.astype(numpy.int32))
    #Summedtif.save('%s' %Entry.get(nameofsummedfile))
    #Summedtif = fabio.edfimage.edfimage(data=data_images_summed.astype(numpy.int32))
    #Summedtif.save('%s' %Entry.get(nameofsummedfile)) #if used put .edf instead of .tif when
saving
    #numpy.savetxt('%s' %Entry.get(nameofsummedfile), data_images_summed) #if used put .txt
when saving
    #numpy.savetxt('%s' %Entry.get(nameofsummedfile), meanI) #if used put .txt when saving
    print 'Done!'

root = Tkinter.Tk()
root.wm_title("py321DIAS_AddorAverage")

Button(root, text='Select Any File in the Directory',
       command=directorylocator).grid(row=2, column=0, columnspan=1)

l1 = Tkinter.Label(root, text="Input nameofsavefile.tif before adding images.")
nameofsummedfile = Tkinter.Entry(root)
l1.grid(row=4, column=0)
nameofsummedfile.grid(row=6, column=0)

Button(root, text='Add or Average Data or Dark Images',
       command=addimages).grid(row=8, column=0, columnspan=2)

root.mainloop()
```

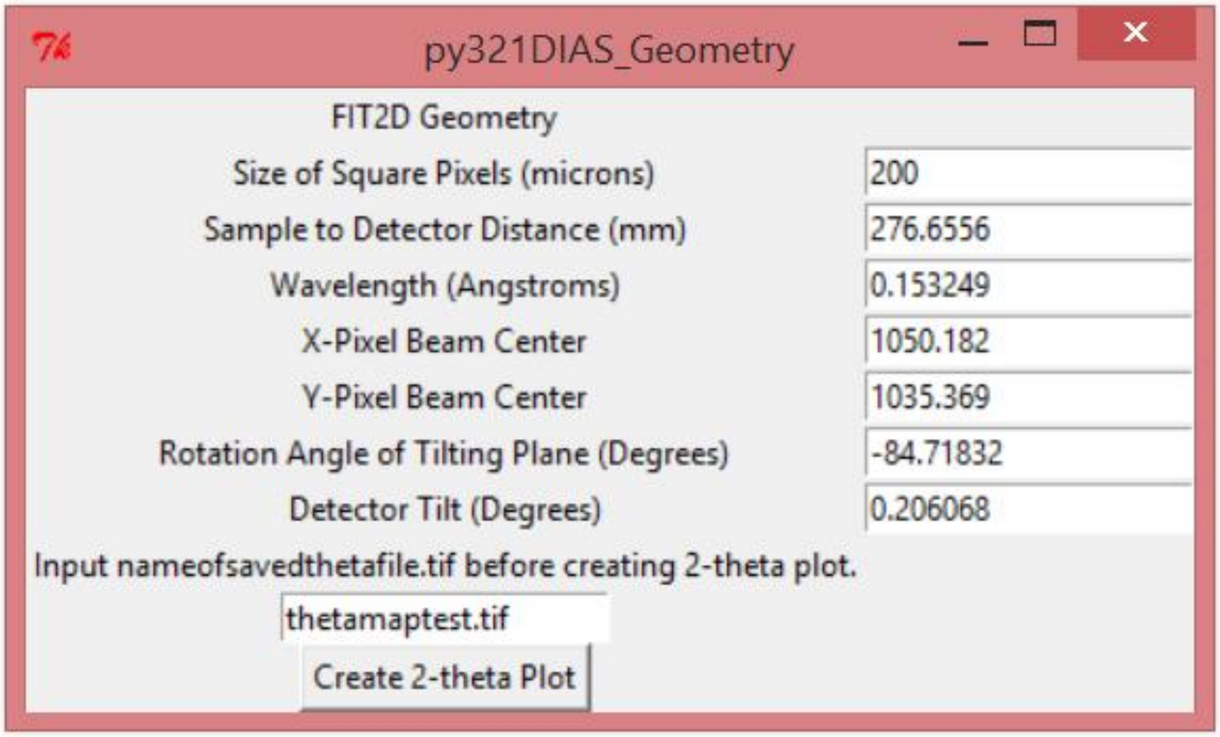
Appendix B: Module for Subtracting Dark Images


py321DIAS_MinusDark
Input nameofdatafile.tif (or any file type that Fabio reads like .edf is also acceptable) before subtracting images.
data.tif
Input nameofdarkfile.tif (or any file type that Fabio reads like .edf is also acceptable) before subtracting images.
dark.tif
Input 'nameofsavedfile.tif' before subtracting images. Only place '' around the name if you get a save error.
dataminusdark.tif
Minus Dark Images from Data Images

```
'''
Module: py321DIAS_MinusDark

Description: Subtracts dark image from data image and saves the new image as a .tif or edf file.
The saved file(s) are located in: C:\Users\MainDirectoryName\workspace\PythonProjectFolder

NOTE: The .tif files must be located in C:\Users\Directory\workspace\PythonProjectFolder in
order for the program to work.
'''
import Tkinter, tkFileDialog
import os
from tkSimpleDialog import *
import numpy
import fabio
import math

def MinusDark():
    data = fabio.open('%s' %Entry.get(tifdata)).data
    dark = fabio.open('%s' %Entry.get(tifdark)).data
    #data = fabio.open('BiFeO3_0GPa_0_mean_150_Even.edf').data #TestData
    #dark = fabio.open('BiFeO3_0GPa_0_mean_150_Odd.edf').data #TestData
    dark_corrected_data = data - dark
    tif = fabio.tifimage.tifimage(data=dark_corrected_data.astype(numpy.int32))
    tif.save('%s' %Entry.get(savefilename))
    #tif = fabio.edfimage.edfimage(data=dark_corrected_data.astype(numpy.int32))
    #tif.save('data_minus_dark.edf')
    print 'Subtraction Completed'
    print 'Saved file(s) are located in
C:\Users\MainDirectoryName\workspace\PythonProjectFolder'

root = Tkinter.Tk()
root.wm_title("py321DIAS_MinusDark")
```

```
Button(root, text='Minus Dark Images from Data Images',
        command=MinusDark).grid(row=14, column=0, columnspan=2)

l1 = Tkinter.Label(root, text="Input nameofdata.tif (or any file type that Fabio reads like .edf is
also acceptable) before subtracting images.")
tifdata = Tkinter.Entry(root)
l1.grid(row=2, column=0)
tifdata.grid(row=4, column=0)

l2 = Tkinter.Label(root, text="Input nameofdark.tif (or any file type that Fabio reads like .edf is
also acceptable) before subtracting images.")
tifdark = Tkinter.Entry(root)
l2.grid(row=6, column=0)
tifdark.grid(row=8, column=0)

l3 = Tkinter.Label(root, text="Input 'nameofsavedfile.tif' before subtracting images. Putting in
.tif and placing '' around the name is paramount.")
savefilename = Tkinter.Entry(root)
l3.grid(row=10, column=0)
savefilename.grid(row=12, column=0)


root.mainloop()
```

Appendix C: Module for Calculating FIT2D Geometry



Size of Square Pixels (microns)	200
Sample to Detector Distance (mm)	276.6556
Wavelength (Angstroms)	0.153249
X-Pixel Beam Center	1050.182
Y-Pixel Beam Center	1035.369
Rotation Angle of Tilting Plane (Degrees)	-84.71832
Detector Tilt (Degrees)	0.206068

Input nameofsavedthetafile.tif before creating 2-theta plot.

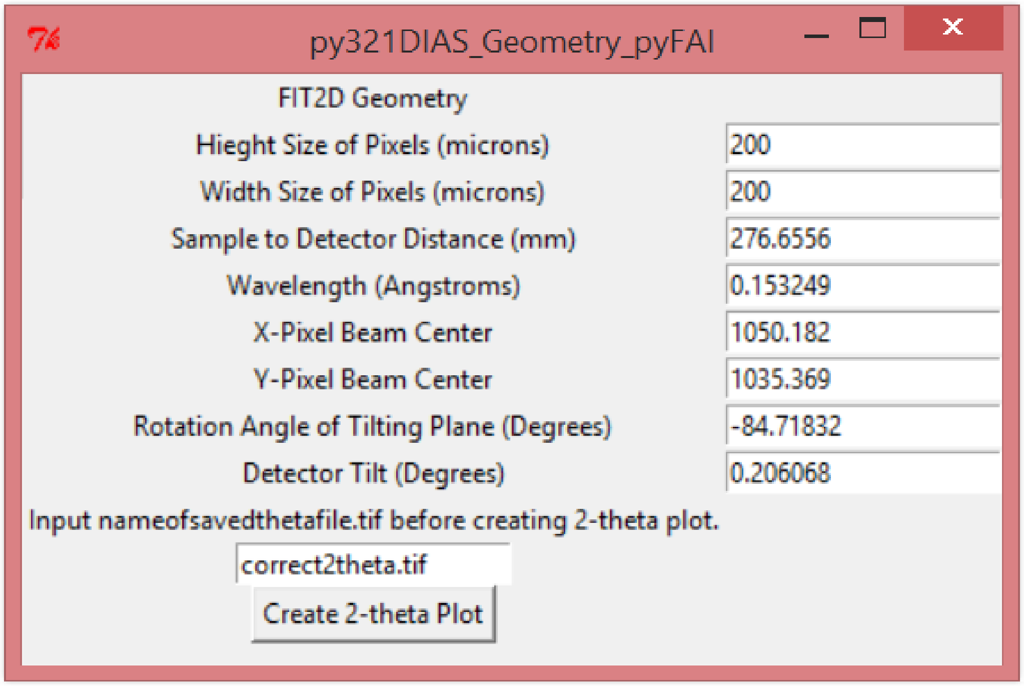
thetamaptest.tif

```
'''
Module: py321DIAS_Geometry

Description: Calculates 2-theta file and saves the information as
a .tif, .edf or .txt file. The saved file(s) are located in:
C:\Users\MainDirectoryName\workspace\PythonProjectFolder
'''

import sys
import Tkinter, tkFileDialog
import os, fnmatch
from tkSimpleDialog import *
import numpy
#from PIL import Image
from numpy import sin
from numpy import cos
from numpy import arctan
from math import  pi
from math import sqrt
import time
import math
numpy.set_printoptions(threshold='nan')
```

```
global thetamap,  twothetaindeg

def geometry():
    global thetamap,  twothetaindeg
    start_time = time.time()
    thetamap = numpy.zeros(4194304, dtype=float)
    thetamap.shape = (2048, 2048)
    SAMP2DET_DIS2 = float(Entry.get(SAMP2DET_DIS))*(10**(-3))
    SAMP2DET_DIS3 = float(Entry.get(sqpixel))*(10**(-6))
    XPCoorx=float(Entry.get(XPCoor))*SAMP2DET_DIS3
    YPCoorx=float(Entry.get(YPCoor))*SAMP2DET_DIS3
    phi=float(Entry.get(DetTilt))*pi/180
    beta=float(Entry.get(RotAng))*pi/180
    for y in range(2048):
        for x in range(2048):
            x1=x*SAMP2DET_DIS3
            y1=y*SAMP2DET_DIS3
            a=(((x1-XPCoorx)*cos(beta))+ ((y1-YPCoorx)*sin(beta)))**2
            b=(cos(phi))**2
            term1=a*b
            term2=((-1*(x1-XPCoorx)*sin(beta))+ ((y1-YPCoorx)*cos(beta)))**2
            c=(((x1-XPCoorx)*cos(beta))+((y1-YPCoorx)*sin(beta)))*sin(phi)+SAMP2DET_DIS2
            denominator=c**2
            d=(term1 + term2)/denominator
            e=sqrt(d)
            thetamap[x][y] = numpy.rad2deg(arctan(e))
            #twothetaindeg[math.degrees(x)][math.degrees(y)] = arctan(e)
            #thetamapping = thetamap[x][y]
    Thetatif = fabio.tifimage.tifimage(thetamap.astype(float))
    Thetatif.save('%s' %Entry.get(nameofthetamapfile)) #Puts name of file in %s
    #Thetatif.save('thetamap.tif') #How saving looks in the code without Entry.get method
    #Thetatif = fabio.edfimage.edfimage(data=thetamap.astype(numpy.float32))
    #Thetatif.save('%s' %Entry.get(nameofthetamapfile)) #if used put .edf instead of .tif when
saving
    #numpy.savetxt('%s' %Entry.get(nameofthetamapfile), thetamap) #if used put .txt when
saving
    print "Program took", time.time() - start_time, "seconds to run"
    print 'Theta mapping complete'

root = Tkinter.Tk()
root.wm_title("py321DIAS_Geometry")

lab = Tkinter.Label(root, text="FIT2D Geometry")
lab.grid(row=10, column=0)

l1 = Tkinter.Label(root, text="Size of Square Pixels (microns)")
```

```
sqpixel = Tkinter.Entry(root)
l1.grid(row=20, column=0)
sqpixel.grid(row=20, column=1)

l3 = Tkinter.Label(root, text="Sample to Detector Distance (mm)")
SAMP2DET_DIS = Tkinter.Entry(root)
l3.grid(row=24, column=0)
SAMP2DET_DIS.grid(row=24, column=1)

l4 = Tkinter.Label(root, text="Wavelength (Angstroms)")
wavelength = Tkinter.Entry(root)
l4.grid(row=26, column=0)
wavelength.grid(row=26, column=1)

l6 = Tkinter.Label(root, text="X-Pixel Beam Center")
XPCoor = Tkinter.Entry(root)
l6.grid(row=30, column=0)
XPCoor.grid(row=30, column=1)

l7 = Tkinter.Label(root, text="Y-Pixel Beam Center")
YPCoor = Tkinter.Entry(root)
l7.grid(row=32, column=0)
YPCoor.grid(row=32, column=1)
l8 = Tkinter.Label(root, text="Rotation Angle of Tilting Plane (Degrees)")
RotAng = Tkinter.Entry(root)

l8.grid(row=34, column=0)
RotAng.grid(row=34, column=1)

l9 = Tkinter.Label(root, text="Detector Tilt (Degrees)")
DetTilt = Tkinter.Entry(root)
l9.grid(row=36, column=0)
DetTilt.grid(row=36, column=1)

l10 = Tkinter.Label(root, text="Input nameofsavedthetafile.tif before creating 2-theta plot.")
nameofthetamapfile = Tkinter.Entry(root)
l10.grid(row=38, column=0)
nameofthetamapfile.grid(row=40, column=0)

Button(root, text='Create 2-theta Plot', command=geometry).grid(row=43, column=0,
columnspan=1)

root.mainloop()
```

Appendix D: Module For Calculating FIT2D Geometry using pyFAI Library



FIT2D Geometry

Hieght Size of Pixels (microns)	200
Width Size of Pixels (microns)	200
Sample to Detector Distance (mm)	276.6556
Wavelength (Angstroms)	0.153249
X-Pixel Beam Center	1050.182
Y-Pixel Beam Center	1035.369
Rotation Angle of Tilting Plane (Degrees)	-84.71832
Detector Tilt (Degrees)	0.206068

Input nameofsavedthetafile.tif before creating 2-theta plot.

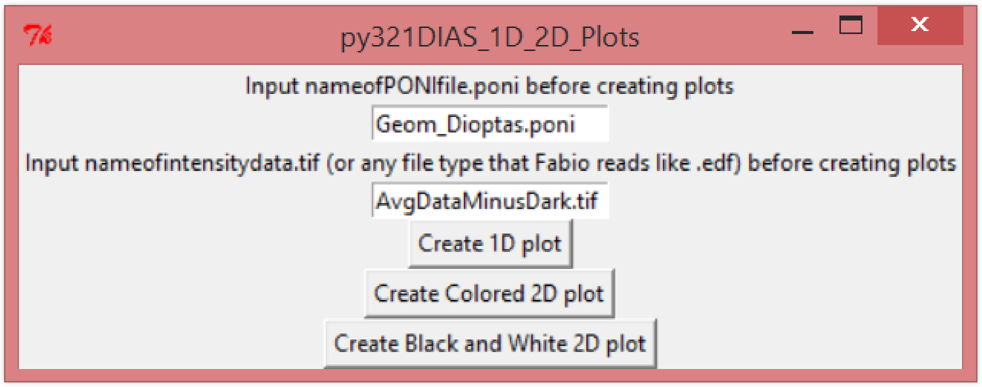
correct2theta.tif

```
'''
Module: py321DIAS_Geometry_pyFAI

Description: Calculates 2-theta file and saves the information as
a .tif, .edf or .txt file. The saved file(s) are located in:
C:\Users\MainDirectoryName\workspace\PythonProjectFolder
'''
import time
import fabio
from pyFAI.geometry import Geometry
import sys
import Tkinter, tkFileDialog
import os, fnmatch
from tkSimpleDialog import *
import numpy
#from PIL import Image
#numpy.set_printoptions(threshold='nan') #Print all data in Python Console and is not
recommended
import math
import pyFAI
import numpy
import fabio
import matplotlib.pyplot as plt

def FIT2Dgeometry_pyFAI():
    start_time = time.time()
    g = pyFAI.geometry.Geometry()
    directDist = float(Entry.get(Samp2Det_Dis))
    centerX = float(Entry.get(XPCoor))
    centerY = float(Entry.get(YPCoor))
    tiltofDet = float(Entry.get(DetTilt))
    RotAngofTiltPlan = float(Entry.get(RotAng))
    pixelX = Entry.get(hpixel)
    pixelY = Entry.get(vpixel)

    tth = g.setFit2D(directDist, centerX, centerY,
                tiltofDet, RotAngofTiltPlan,
                pixelX, pixelY, splineFile=None)

    #tth = g.setFit2D(directDist=276.6556, centerX=1050.182, centerY=1035.369,
    #            tilt=0.206068, tiltPlanRotation=-84.71832,
    #            pixelX=200, pixelY=200, splineFile=None)
    tth = g.getFit2D()
    #mapping = g.twoThetaArray((2048, 2048))
    mapping = numpy.rad2deg(g.twoThetaArray((2048, 2048)))
    sortedindex = mapping.argsort(axis=None)
    xaxisval = mapping.flatten(sortedindex)
    #mapping = g.twoThetaArray((numpy.array([2048]), numpy.array([2048])))
    #thetatif = fabio.tifimage.tifimage(data=mapping.astype(numpy.float32))
    #thetatif.save('%s' %Entry.get(nameofthetamapfile)) #Puts name of file in %s
    #thetatif.save('thetamap.tif') #How saving looks in the code without Entry.get method
    #thetatif = fabio.edfimage.edfimage(data=thetamap.astype(numpy.float32))
    #thetatif.save('%s' %Entry.get(nameofthetamapfile)) #if used put .edf instead of .tif when
saving
    #numpy.savetxt('%s' %Entry.get(nameofthetamapfile), thetamap) #if used put .txt when
saving
    print mapping
    print sortedindex
    print xaxisval
#    plt.plot(mapping)
#    plt.show(mapping)
#    plt.xlabel('x')
#    plt.ylabel('y')
#    plt.title('2-Theta Geometry')
    print mapping.shape
    print "Program took", time.time() - start_time, "seconds to run"
    print 'Theta mapping complete'
    #print tth
    #print mapping

root = Tkinter.Tk()
root.wm_title("py321DIAS_Geometry_pyFAI")
```

```
lab = Tkinter.Label(root, text="FIT2D Geometry")
lab.grid(row=10, column=0)

l1 = Tkinter.Label(root, text="Hieght Size of Pixels (microns)")
vpixel = Tkinter.Entry(root)
l1.grid(row=18, column=0)
vpixel.grid(row=18, column=1)

l2 = Tkinter.Label(root, text="Width Size of Pixels (microns)")
hpixel = Tkinter.Entry(root)
l2.grid(row=20, column=0)
hpixel.grid(row=20, column=1)

l3 = Tkinter.Label(root, text="Sample to Detector Distance (mm)")
Samp2Det_Dis = Tkinter.Entry(root)
l3.grid(row=24, column=0)
Samp2Det_Dis .grid(row=24, column=1)

l4 = Tkinter.Label(root, text="Wavelength (Angstroms)")
wavelength = Tkinter.Entry(root)
l4.grid(row=26, column=0)
wavelength.grid(row=26, column=1)

l5 = Tkinter.Label(root, text="X-Pixel Beam Center")
XPCoor = Tkinter.Entry(root)
l5.grid(row=30, column=0)
XPCoor.grid(row=30, column=1)

l6 = Tkinter.Label(root, text="Y-Pixel Beam Center")
YPCoor = Tkinter.Entry(root)
l6.grid(row=32, column=0)
YPCoor.grid(row=32, column=1)

l7 = Tkinter.Label(root, text="Rotation Angle of Tilting Plane (Degrees)")
RotAng = Tkinter.Entry(root)
l7.grid(row=34, column=0)
RotAng.grid(row=34, column=1)

l8 = Tkinter.Label(root, text="Detector Tilt (Degrees)")
DetTilt = Tkinter.Entry(root)
l8.grid(row=36, column=0)
DetTilt.grid(row=36, column=1)

l9 = Tkinter.Label(root, text="Input nameofsavedthetafile.tif before creating 2-theta plot.")
nameofthetamapfile = Tkinter.Entry(root)
```

```
l9.grid(row=38, column=0)
nameofthetamapfile.grid(row=40, column=0)

Button(root, text='Create 2-theta Plot', command=FIT2Dgeometry_pyFAI).grid(row=43,
column=0, columnspan=1)

root.mainloop()
```

Appendix E: Module for 1D and 2D plots



Input nameofPONIfile.poni before creating plots

Geom_Dioptas.poni

Input nameofintensitydata.tif (or any file type that Fabio reads like .edf) before creating plots

AvgDataMinusDark.tif

```
'''
Module: py321DIAS_1D_2D_plots

Description: This module takes advantage of pyFAI and performs azimuthal integration to
acquire a 1D diffraction pattern and it can produce a 2D diffraction pattern from the 1D data,
while using matplotlib to save the image in a variety of formats and anywhere on the computer.
'''
import time
import fabio
#from pyFAI.geometry import Geometry
import sys
import Tkinter, tkFileDialog
import os, fnmatch
from tkSimpleDialog import *
import numpy
#from PIL import Image
#numpy.set_printoptions(threshold='nan') #Print all data in Python Console and is not
recommended
import math
import pyFAI
import numpy
import fabio
from PIL import Image
import time
import matplotlib.pyplot as plt
from matplotlib import pyplot
import matplotlib.cm as cm
from matplotlib import style
from matplotlib.pyplot import grid
#style.use('fivethirtyeight')#very good but shows gridlines
#style.use('ggplot')#good but shows gridlines
#print(plt.style.available)
```

```
def plot1D():
    start_time = time.time()
    ai = pyFAI.azimuthalIntegrator.AzimuthalIntegrator()
    ai.load('%s' %Entry.get(geometryponifile))
    Intensitydata = fabio.open('%s' %Entry.get(intensityfile)).data
    Intensitydata = numpy.flipud(Intensitydata)
    #ai.load('BiFeO3FIT2DGeometry_Dioptas.poni')
    #Intensitydata = fabio.open('data_minus_dark.tif').data
    x=[]
    y=[]
    x, y = ai.integrate1d(Intensitydata, npt=2048*2048, unit='2th_deg', polarization_factor=0.999,
method = 'BBox')
    plt.plot(x, y)
    plt.xlabel('2-Theta')
    plt.ylabel('Intensity')
    print "Program took", time.time() - start_time, "seconds to run"
    plt.show()

def plot2D():
    style.use('fivethirtyeight')
    start_time = time.time()
    ai = pyFAI.azimuthalIntegrator.AzimuthalIntegrator()
    ai.load('%s' %Entry.get(geometryponifile))
    Intensitydata = fabio.open('%s' %Entry.get(intensityfile)).data
    Intensitydata = numpy.flipud(Intensitydata)
    #ai.load('BiFeO3FIT2DGeometry_Dioptas.poni')
    #Intensitydata = fabio.open('data_minus_dark.tif').data
    x=[]
    y=[]
    x, y = ai.integrate1d(Intensitydata, npt=2048*2048, unit='2th_deg', polarization_factor=0.999,
method = 'BBox')
    TwoDplot = ai.calcfrom1d(x, y, shape=(2048,2048), mask=None, dim1_unit='2th_deg',
correctSolidAngle=None)
    plt.imshow(TwoDplot)
    print "Program took", time.time() - start_time, "seconds to run"
    cbar = plt.colorbar()
    cbar.set_label('Intensity')
    plt.grid(False)
    plt.show()

def BlackWhite2Dplot():
    #style.use('fivethirtyeight')
    style.use('grayscale')
    start_time = time.time()
    ai = pyFAI.azimuthalIntegrator.AzimuthalIntegrator()
    ai.load('%s' %Entry.get(geometryponifile))
    Intensitydata = fabio.open('%s' %Entry.get(intensityfile)).data
    Intensitydata = numpy.flipud(Intensitydata)
    #ai.load('BiFeO3FIT2DGeometry_Dioptas.poni')
    #Intensitydata = fabio.open('data_minus_dark.tif').data
    x=[]
    y=[]
    x, y = ai.integrate1d(Intensitydata, npt=2048*2048, unit='2th_deg', polarization_factor=0.999,
method = 'BBox')
    TwoDplot = ai.calcfrom1d(x, y, shape=(2048,2048), mask=None, dim1_unit='2th_deg',
correctSolidAngle=None)
    plt.imshow(TwoDplot, cmap=cm.gray)
    print "Program took", time.time() - start_time, "seconds to run"
    cbar = plt.colorbar()
    cbar.set_label('Intensity')
    plt.grid(False)
    plt.show()

root = Tkinter.Tk()
root.wm_title("py321DIAS_1D_2D_Plots")


l1 = Tkinter.Label(root, text="Input nameofPONIfile.poni before creating plots")
geometryponifile = Tkinter.Entry(root)
l1.grid(row=2, column=0)
geometryponifile.grid(row=4, column=0)

l2 = Tkinter.Label(root, text="Input nameofintensitydata.tif (or any file type that Fabio reads like
.edf) before creating plots")
intensityfile = Tkinter.Entry(root)
l2.grid(row=6, column=0)
intensityfile.grid(row=8, column=0)

Button(root, text='Create 1D plot',
       command=plot1D).grid(row=10, column=0, columnspan=1)

Button(root, text='Create Colored 2D plot',
       command=plot2D).grid(row=12, column=0, columnspan=1)

Button(root, text='Create Black and White 2D plot',
       command=BlackWhite2Dplot).grid(row=20, column=0, columnspan=1)

root.mainloop()
```

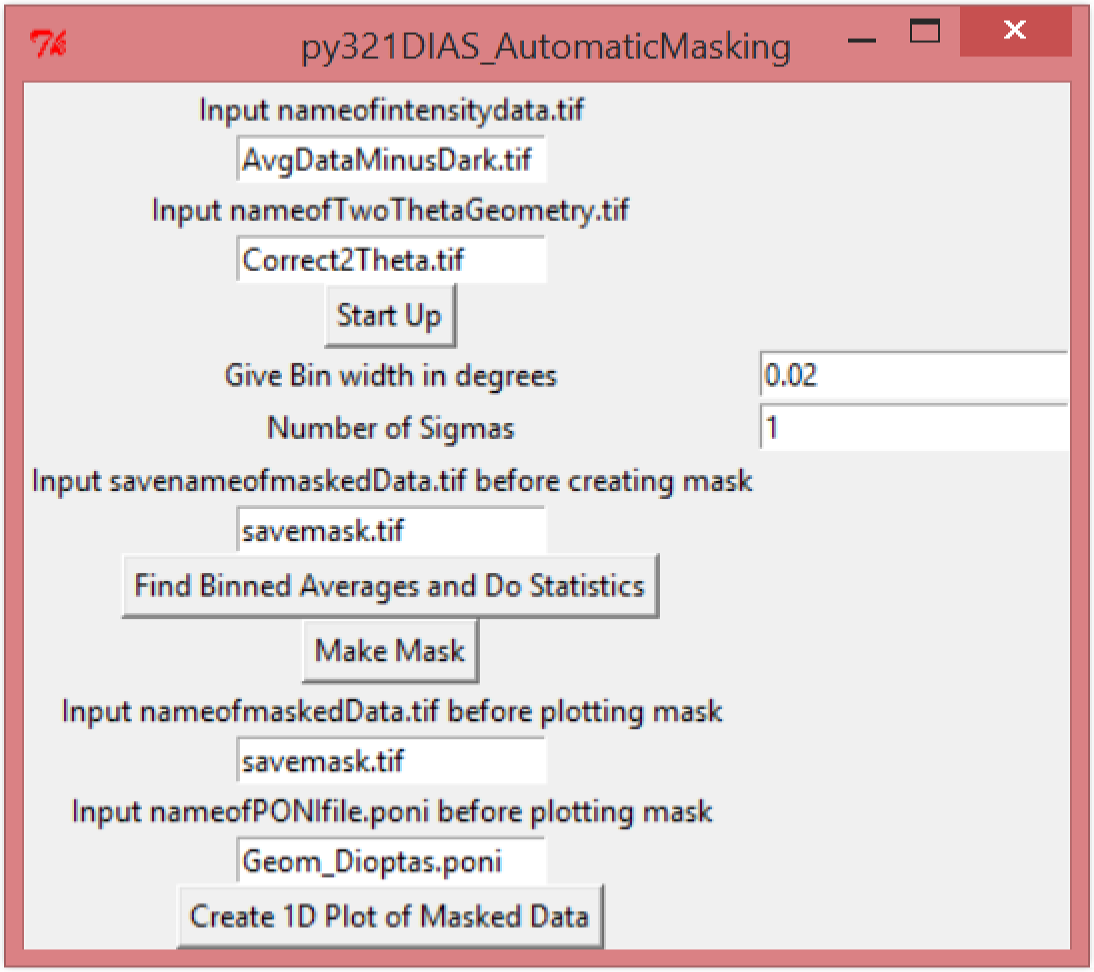
Appendix F: Module for Statistical Analysis

```
'''
Module: py321DIAS_StatisticalAnalysis_G_P

Description: Produces a scatter plot of intensity data within a certain 2-theta range and
allows a certain bin and bin size to be chosen for creating a frequency verses intensity histogram
that is fitted by a distribution method. Matplotlib is used to plot and save the images in a variety
of formats and anywhere on the computer.

Note: This is the only module that does not have an official graphical user interface yet and is
still being modified for optimal usage.
'''

import fabio
import numpy
from numpy import sqrt
from numpy import exp
from math import pi
from scipy.misc import factorial
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
from matplotlib import style
from matplotlib.pyplot import grid
import time
style.use('fivethirtyeight')

start_time = time.time()

Intensitydata = fabio.open('AvgDataMinusDark.tif').data
Intensitydata = numpy.flipud(Intensitydata)

thetamap = fabio.open('Correct2Theta.tif').data
thetamap_degree =numpy.rad2deg(thetamap)

bin_width = 0.02
bin_midpoint = bin_width/2

intensity_value = 18.51
lower_bound= float(intensity_value - bin_midpoint)
upper_bound= float(intensity_value + bin_midpoint)
theta_bin_index_location=numpy.where((thetamap_degree>=lower_bound) &
(thetamap_degree<=upper_bound))
max_index = numpy.amax(theta_bin_index_location)
theta_bin_values = thetamap_degree[theta_bin_index_location]
bin_intensities = Intensitydata[theta_bin_index_location]
```

```
print "bin_intensities", bin_intensities

def ScatterPlot_Region():
    plt.scatter(theta_bin_values, bin_intensities, c='b', marker=',') #use marker= to choose the
shape of the points ex:*, ^, s, h, x, D, 8, etc.
    #plt.xlim(numpy.amin(theta_bin_values), numpy.amax(theta_bin_values))
    plt.xlim(numpy.amin(theta_bin_values)-0.01, numpy.amax(theta_bin_values)+0.01)
    plt.ylim(numpy.amin(bin_intensities)-5, numpy.amax(bin_intensities)+5)
    plt.xlabel(r"2$\theta$ (degrees)")
    plt.ylabel('Intensity (counts)')
    plt.show()
#ScatterPlot_Region()

Is = bin_intensities
print Is
minIs = numpy.rint(numpy.amin(Is))
maxIs = numpy.rint(numpy.amax(Is))
print 'min intensity =', minIs, 'max intensity =', maxIs
bins =  maxIs - minIs
print "bins", bins
print "---------------1------------------"
result = plt.hist(Is, bins, histtype='bar',rwidth=1, color='g')
print 'result', result
print "---------------2------------------"
N = Is.size
meanI = numpy.sum(Is)/N
print 'meanI', meanI
print "---------------3------------------"
sigma_P = sqrt(meanI)
sigma_G = sqrt((numpy.sum((Is-meanI)**2))/(N-1))
print 'sigma_G', sigma_G, 'sigma_P', sigma_P
print "---------------4------------------"
x_plot = numpy.linspace(numpy.rint(numpy.amin(Is)), numpy.rint(numpy.amax(Is)),
max_index)
print 'x', x_plot
print "---------------5------------------"
dx = (result[1][1] - result[1][0])
print 'dx',dx
print "---------------6------------------"
scale = len(Is)*dx
print 'length of yaxis', len(Is)
print 'scale', scale
print "---------------7------------------"

normG = mlab.normpdf(x_plot,meanI,sigma_G)*scale
#plt.plot(x_plot, normG, color='r', lw=2)
```

```
def Gaussian(Is, meanI, sigma):
    Normdist = (1/(sigma*sqrt(2*pi)))*exp(-(Is - meanI)**2/(2*sigma**2))
    return Normdist

plt.plot(x_plot, scale*Gaussian(x_plot, meanI, sigma_G), 'r', lw=2)

def poisson(I, mean):
    return scale * (mean**I/factorial(I)) * exp(-mean)

#plt.plot(x_plot, poisson(x_plot, meanI), 'c-', lw=2)

plt.xlabel('Intensity (Counts)')
plt.ylabel('Frequency (Number of Pixels)')
plt.show()
```

Appendix G: Module for Automatic Masking


py321DIAS_AutomaticMasking

Input nameofintensitydata.tif

AvgDataMinusDark.tif

Input nameofTwoThetaGeometry.tif

Correct2Theta.tif

Give Bin width in degrees: 0.02

Number of Sigmas: 1

Input savenameofmaskedData.tif before creating mask

savemask.tif

Input nameofmaskedData.tif before plotting mask

savemask.tif

Input nameofPONIfile.poni before plotting mask

Geom_Dioptas.poni

```
'''
Module: py321DIAS_AutomaticMasking

Description: Uses a statistical analysis method to automatically masked data that is outside a
certain number of standard deviation away from the mean.
'''
import sys
import Tkinter, tkFileDialog
import os, fnmatch
from tkSimpleDialog import *
import numpy
import math
import pyFAI
import numpy
import fabio
from PIL import Image
import time
```

```
from numpy import sqrt
import matplotlib.pyplot as plt
from matplotlib import pyplot
import matplotlib.cm as cm
from matplotlib import style
from matplotlib.pyplot import grid
#style.use('fivethirtyeight')#very good but shows gridlines
#style.use('ggplot')#good but shows gridlines
#print(plt.style.available)

global mask_data, thetamap, sortedpic_index, keepit_pix, keepit_inten, sortedtheta, sortedinten
def StartUp():
    global mask_data, thetamap, sortedpic_index, keepit_pix, keepit_inten, sortedtheta,
sortedinten
    start_time = time.time()
    Intensitydata = fabio.open('%s' %Entry.get(Intensitydata_tif)).data
    #Intensitydata = fabio.open('AvgDataMinusDark.tif').data
    Intensitydata = numpy.flipud(Intensitydata)
    thetamap = fabio.open('%s' %Entry.get(TwoTheta_tif)).data
    #thetamap = fabio.open('Correct2Theta.tif').data

    sorted_index = thetamap.argsort(axis=None)
    pic_index=numpy.arange(4194304)
    sortedpic_index=pic_index[sorted_index]
    sortedtheta=numpy.rad2deg(thetamap.flatten()[sorted_index]) #changes outputs from radians
to degrees
    sortedinten=Intensitydata.flatten()[sorted_index]
    print "Program took", time.time() - start_time, "seconds to run"

def eliminate(bin_intensity_array, sigma, mapped_pic_index1):
    start_time = time.time()
    global sigmanum, keepit_pix, keepit_inten
    midpoint1=numpy.median(bin_intensity_array)
    lower_bound= float(midpoint1 - 0.5*sigmanum*sigma)
    upper_bound= float(midpoint1 + 0.5*sigmanum*sigma)
    index_location=numpy.where((bin_intensity_array>=lower_bound) &
(bin_intensity_array<=upper_bound)) # produces good intensity LOCATIONS
    keepit_pix.extend(mapped_pic_index1[index_location])
    keepit_inten.extend(bin_intensity_array[index_location])
    print "Program took", time.time() - start_time, "seconds to run"

def freq_occurrence(Is,sigma1, midpoint1, mapped_pic_index1):
    start_time = time.time()
    frequency=[]
    intensities=Is
    last=int(numpy.size(Is)-1) #upper bound index
    sorted_bin=sorted(Is)
    lowerbound=sorted_bin[0]
    upperbound=sorted_bin[last]
    #print 'upper and lower intensity bounds:', lowerbound, upperbound
    bin_intensity_array = []
    total = int(round(upperbound-lowerbound))
    counter2=1 # TO COUNT THRU THE WHOLE INTERVAL;
    while counter2 < total: # counts number of times a particular intensity shows up
        intensity_value = float ((round(lowerbound)-2)+counter2)
        lower_bound= float(intensity_value - .5)
        upper_bound= float(intensity_value + .5)
        index_location=numpy.where(( intensities>=lower_bound) & (intensities<=upper_bound))
#gives good intensity locations
        bin_size = numpy.size(index_location)
        frequency.append(bin_size) # adds to array containing the number of occurrences
        bin_intensity_array.append(intensity_value)
        counter2 = counter2 + 1
    frequency_statistics=eliminate(Is, sigma1, mapped_pic_index1)
    #plt.plot(bin_intensity_array, frequency)
    #plt.show()
    print "Program took", time.time() - start_time, "seconds to run"

def bin_averaging():
    start_time = time.time()
    global sortedtheta, keepit_pix, keepit_inten, sortedinten, lower_bound, sortedpic_index,
upper_bound, sigmanum, trash, keepit
    keepit_pix=[]
    keepit_inten=[]
    trash=[]
    sigmanum1 = (Entry.get(sigmanum))
    sigmanum=float(sigmanum1)
    delta = float(Entry.get(delta_input))
    delta2theta= float(delta) # should be an angle
    halfdelta=float(delta2theta/2)
    counter=1
    counter_total= int(numpy.amax(sortedtheta)/delta)+1
    index_location=[]
    bin_avg=[] # array containing bin average values (same order as theta)
    arrayindex=[] # array containing midpoint values
    while counter < counter_total:
        print counter
        midpoint = float (delta2theta*counter)
        lower_bound= float(midpoint - halfdelta)
        upper_bound= float(midpoint + halfdelta)
        index_location=numpy.where((sortedtheta>=lower_bound) & (sortedtheta<=upper_bound))
#keeps theta values between bounds
```

```
        kept_pic_index=sortedpic_index[index_location] #pixel locations for theta values within
bounds
        kept_bin=sortedinten[index_location]
        kept_theta=sortedtheta[index_location]
        bin_sum=float(numpy.sum(kept_bin)) #sums values in bin
        bin_size = float (numpy.size(index_location))
        #print 'midpoint', midpoint
        if bin_size ==0:
            counter = counter+1
            bin_average= float(bin_sum/1)
            sigma=sqrt(bin_average) #reduces to n-sigma

        else:
            bin_average= float(bin_sum/bin_size)
            sigma=sqrt(bin_average) #reduces to n-sigma
            freq_occurrence(kept_bin, sigma, midpoint, kept_pic_index) # at particular theta
            arrayindex.append(midpoint) # adds value to array midpoint values
            bin_avg.append(bin_average) # adds value to array bin average values
            counter = counter + 1
        #print 'counter1', counter
#    print 'pixel locations:', keepit_pix
#    print 'pixel intensities:', keepit_inten
    print "Program took", time.time() - start_time, "seconds to run"

def maskmaker(): # zero is black white is one
    start_time = time.time()
    global keepit_pix, keepit_inten
    mask=numpy.zeros((4194304), dtype=int)
    mask[keepit_pix] = keepit_inten #keepit_pix#=255 #255 makes the good intensity areas white
    #print 'mask', mask
    mask_data=numpy.resize(mask,(2048,2048))
    masked_img=Image.fromarray(mask_data)
    MaskedData = fabio.tifimage.tifimage(data = mask_data.astype(numpy.int32))
    MaskedData.save('%s' %Entry.get(savemaskedData))
    plt.imshow(masked_img)#,cmap='Greys')
    plt.show()
    print "Program took", time.time() - start_time, "seconds to run"

def Plot1DMask():
    start_time = time.time()
    openmaskedData = fabio.open('%s' %Entry.get(maskedData_tif)).data
    ai = pyFAI.azimuthalIntegrator.AzimuthalIntegrator()
    ai.load('%s' %Entry.get(geometryponifile))
    x=[]
    y=[]
    x, y = ai.integrate1d(openmaskedData, npt=2048*2048, unit='2th_deg',
polarization_factor=0.999, method = 'BBox')
    plt.xlabel('2-Theta')
    plt.ylabel('Intensity')
    print "Program took", time.time() - start_time, "seconds to run"
    plt.show()


root = Tkinter.Tk()
root.wm_title("py321DIAS_AutomaticMasking")



l11 = Tkinter.Label(root, text="Input nameofintensitydata.tif")
Intensitydata_tif = Tkinter.Entry(root)
l11.grid(row=2, column=0)
Intensitydata_tif.grid(row=4, column=0)

l12 = Tkinter.Label(root, text="Input nameofTwoThetaGeometry.tif")
TwoTheta_tif = Tkinter.Entry(root)
l12.grid(row=6, column=0)
TwoTheta_tif.grid(row=8, column=0)

Button(root, text='Start Up', command=StartUp).grid(row=10, column=0, columnspan=1)

l13 = Tkinter.Label(root, text="Give Bin width in degrees")
delta_input = Tkinter.Entry(root)
l13.grid(row=12, column=0)
delta_input.grid(row=12, column=1)

l14 = Tkinter.Label(root, text="Number of Sigmas")
sigmanum = Tkinter.Entry(root)
l14.grid(row=14, column=0)
sigmanum.grid(row=14, column=1)

l15 = Tkinter.Label(root, text="Input savenameofmaskedData.tif before creating mask")
savemaskedData = Tkinter.Entry(root)
l15.grid(row=16, column=0)
savemaskedData.grid(row=18, column=0)

Button(root, text='Find Binned Averages and Do Statistics',
command=bin_averaging).grid(row=20, column=0, columnspan=1)

Button(root, text='Make Mask', command=maskmaker).grid(row=22, column=0, columnspan=1)

l16 = Tkinter.Label(root, text="Input nameofmaskedData.tif before plotting mask")
maskedData_tif = Tkinter.Entry(root)
```

```
l16.grid(row=24, column=0)
maskedData_tif.grid(row=26, column=0)

l17 = Tkinter.Label(root, text="Input nameofPONIfile.poni before plotting mask")
geometryponifile = Tkinter.Entry(root)
l17.grid(row=28, column=0)
geometryponifile.grid(row=30, column=0)

Button(root, text='Create 1D Plot of Masked Data', command=Plot1DMask).grid(row=34,
column=0, columnspan=1)

root.mainloop()
```