Identifying lipid mesophases from a set of Small-Angle X-Ray Scattering (SAXS) peaks

Christopher Brasnett[1] and Annela Seddon[1,2]

[1]H.H. Wills Physics Laboratory, Tyndall Avenue, University of Bristol, Bristol BS8 1FD

[2]Bristol Centre for Functional Nanomaterials, HH Wills Physics Laboratory, Tyndall Avenue, University of Bristol, Bristol BS8 1FD

January 18, 2018

Abstract

A statistical method of identifying lipid mesophases from Small-Angle X-ray Scattering (SAXS), is described, having been implemented in python code at https://github.com/csbrasnett/lipidsaxs

1 Introduction

High-throughput methods of SAXS to identiying the mesophase behaviour of aggregated lipid systems necessarily produce a lot of data. In general - or at least most often - the SAXS patterns such systems produce belong to one of five mesophases: Lamellar, Inverse Hexagonal, and the Primitive, Diamond, and Gyroid cubic phases. They are distinguished through the characteristic spacings of the Bragg peaks present in the SAXS patterns [1]. A consequence of the combination of subtle differences in the characteristic peak spacings, combined with rigorously identifying peaks in 1D azimuthally-integrated SAXS data is that the initial analysis of lipid mesophase data is both time-consuming, and repetitive. The methods described in this paper have been developed and implemented so that less time has to be spent analysing phase behaviour of SAXS data, and more time can be spent on designing experiments.

The Github repository has 4 python scripts: *finder.py*, *detect_peaks.py*, *phase_ID.py*, and *complete.py*. *finder* and *detect_peaks* are responsible for the finding and fitting of Bragg peaks in 1D I vs. q SAXS data, whilst *phase_ID* tries to assign a phase to the collection of Bragg peaks found. *complete* is an example script calling the others in order to demonstrate the type of results that can be expected from the analysis. *complete* has some example parameters in, and only requires a folder of text files of I vs q data in order to run. The exact description of the parameters is contained in this document.

2 Peak identification

The first stage of identifying the mesophase behaviour of lipid systems is necessarily finding the position in q of the Bragg peaks of the scattering pattern. The peak positions are found using the script *finder.py*. *finder.py* requires five variables to be passed to it:

file_name : The location of a text file of the q and Intensity data, formatted into two columns, the first of which is the q data, and the second the I(q).

lo_lim, hi_lim : Cut offs in values of q for the q range to be searched for peaks, and, later, for confirming permitted mesophase assignments.

fig : A switch for whether a graph is produced at the end of the finding routine, showing the data with overlaid lines of the positions in q of where the peaks in the returned array have been found. The figure will only be returned if this value is 1.

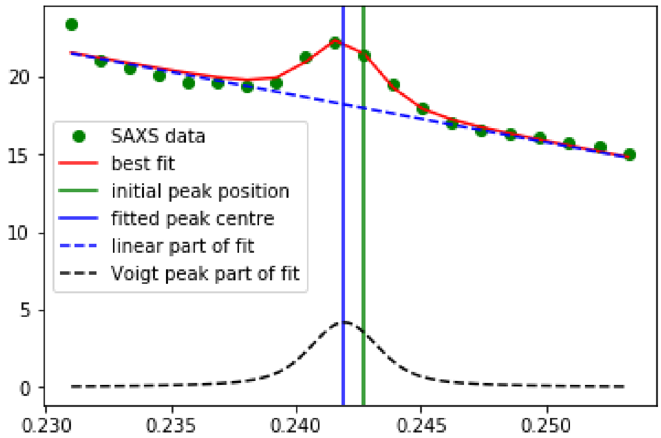


Figure 1: An example of the fitting performance of a Bragg peak. Axes of q vs. I(q). As labeled in the figure, the vertical green line is the initial peak position found by the *detect_peaks.py* script. This initial peak position is passed to the fitting function, along with the data shown as discrete green points. The data are fitted with a convolution of a linear background and a voigt peak, the individual contributions of which are shown as dotted black lines, and shown as a convolution in solid red. The new, 'true' centre of the peak to be returned in an array of total peak positions, at the centre of the fitted Voigt peak, is shown as a vertical solid blue line.

The programme has two functions, finder and fitting. The variables for the fitting function are entirely determined by the routine of the finder. In the first instance, the function uses the *detect_peaks.py* script, written by Marcos Duarte, and made available at http://nbviewer.jupyter.org/github/demotu/BMC/blob/master/notebooks/DetectPeaks.ipynb. These are subsequently refined so that the peaks for consideration only lie within the user-defined accepted q range, as given by the **lo_lim** and **hi_lim** variables. However, as the function only finds peaks in discrete values of q, and not always successfully. Therefore, the key aspect of the finding script is using a range of points around the discrete point to fit the data using a peak and background model. The fitting is performed using the *lmfit* library [2]. *lmfit* is not installed as a default library on some common distributions of python (eg. Anaconda), so should be installed or checked for installation before the scripts are used.

The peak and background model used is a convolution of a Voigt peak model, and a linear background, as demonstrated in Fig. 1. The peak model has been found not to work well on its own, due to the fact that the form factor of the SAXS pattern means there is a significant background aspect to the data. Additionally to note is the fact that the gamma parameter of the Voigt peak fit is set to equal the sigma parameter by default. These parameters are used to determine aspects of the peak width in the fit of the data. They can, however, be set to not equal each other to improve the fit, but extensive testing of the free variation of the gamma parameter has not proved fruitful in improving the fitting of SAXS data. Most commonly, it has resulted in the centre of the peak being found very far away from where the true centre is. When the free variation of the gamma parameter is turned off, the data are usually found to fit very well. For more information of the fit, the *lmfit* results table can be returned at each instance of fitting, if so desired, by changing the value of the final **plot** variable in the fitting function, as called in the finding function, to 1. The fitting function then simply returns where the centre of the peak has been found to be.

The set of initial peaks are fitted over a range of fitting widths, so to try to maintain as much information as possible about the data. The set of peak centres found in this way are subsequently tested for degeneracy, to remove peaks very close to each other. The peaks established in this way are then used as the final set of peaks for the data passed to the module, returned as a numpy array. At this point, if the **fig** variable has been set to 1, then a figure of the data with the peak positions overlaid as vertical lines is shown.

The peaks obtained by this module can now be used by the *Phase_ID.py* script to try to determine the mesophase of the sample from which the SAXS data came.

3 Phase Identification

The 5 most common mesophases described earlier have characteristic peak spacings like so:

Lamellar ($\mathbf{L}_\alpha$) : 1,2,3

Inverse Hexagonal ($\mathbf{H}_{II}$) : 1, $\sqrt{3}$, $\sqrt{4}$

Primitive Cubic ($\mathbf{Q}^{P}_{II}$) : $\sqrt{2}$, $\sqrt{4}$, $\sqrt{6}$, $\sqrt{8}$, $\sqrt{10}$, $\sqrt{12}$, $\sqrt{14}$

Diamond Cubic ($\mathbf{Q}^{D}_{II}$) : $\sqrt{2}$, $\sqrt{3}$, $\sqrt{4}$, $\sqrt{6}$, $\sqrt{8}$, $\sqrt{9}$, $\sqrt{10}$, $\sqrt{11}$

Gyroid Cubic ($\mathbf{Q}^{G}_{II}$) : $\sqrt{6}$, $\sqrt{8}$, $\sqrt{14}$, $\sqrt{16}$, $\sqrt{20}$, $\sqrt{22}$, $\sqrt{24}$

If a set of peaks can be said to belong to a particular mesophase, then the lattice parameter (that is, repeat unit cell spacing) of the mesophase can be subsequently calculated:

Lamellar : $\mathrm{a}_{[hkl]}=\frac{2\pi}{q_{[hkl]}}$

Inverse Hexagonal $\mathrm{a}_{[hk]}=\frac{2}{\sqrt{3}}\frac{\sqrt{h^2+k^2-hk}}{q_{hk}}$

Cubic phases $\mathrm{a}_{[hkl]}=\frac{\sqrt{h^2+k^2+k^2}}{q_{[hkl]}}$

Indeed, most easily, a mesophase can be identified by finding a set of peaks which, when used for the above calculations, and indexed correctly, produce the same value. However, and especially in the case of coexisting systems, this is a laborious method to tackle the identification problem. It is therefore useful to have a process of automatically indexing peaks to identify the mesophase.

Two facts are useful to note at this point:

- The cubic phases more readily scatter to higher-order Bragg peaks, and thus it is useful to know the characteristic ratios to higher order
- Within the characteristic ratios of the cubic phases, there is a good degree of overlap with regards to the Miller indicies that peaks may be assigned.

With respect to these factors, the first step that is taken is to discriminate the possibilities of the phase to assign to any set of peaks by the number of peaks provided to the phase_ID.py programme. If 3 or fewer peaks have been identified, then it is more likely that the data are from a L_α or H_{II} phase than a cubic one[1]. As noted previously, distinguishing between cubic mesophases is a greater challenge than other possibilities, and so the method used for clarification there will be described first.

The entire *phase_ID.py* script can be called from the main function. main expects two variables:

peaks : An array of peak positions

lo_q : The same lo_q value previously used to define the region in which to search for Bragg peaks.

main firstly discriminates what kind of phase to search for by the number of peaks that have been passed to it. This is run in a *while* loop. The *while* loop accounts for the possibility of cubic/non-cubic phase coexistence. If such a situation has arisen, then a greater number of peaks will have been found, of which a subset may be successfully assigned as cubic, but leaving the non-cubic ones. Once the number of peaks is used to determine which sort of phases to search for, the separate functions are called, whose variables and methods are detailed in the following sections. Subsequently, if the number of peaks left unassigned through this search is still greater than 1, the remaining peaks are re-investigated, until either all peaks have been assigned, or until the *while* loop has been run 10 times. Ultimately, the main programme returns a dictionary of the phases found, along with any unassigned peaks.

[1] Of course from single crystalline mesophase structures, such as those of the Kim and coworkers [3], there is the possibility of systematic Bragg extinctions, resulting in fewer peaks present anyway

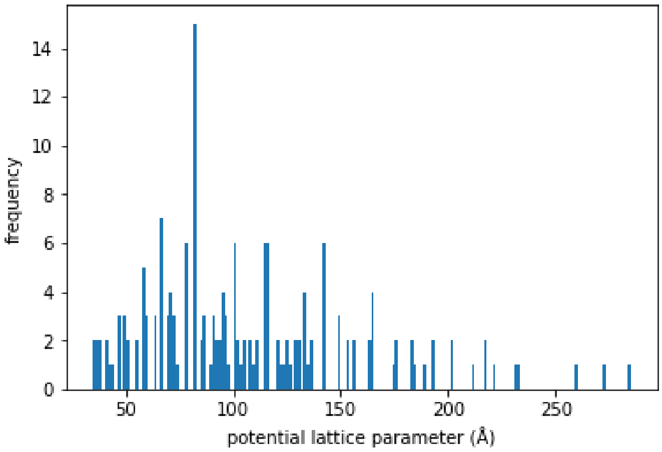


Figure 2: A histogram of all possible lattice constants of a set of input peaks, as a result of indexing every peak with every possible index. There are higher peaks where the same lattice constant has been found more often: one of these will be the correct indexing of the peaks provided.

3.1 The cubic phases

(NB for this section: if a programme variable is described as 'x_', x is being used to describe the set of identical variables at the same stage of the method to separate the P, D, and G variables.)

The function Q_main is used to identify cubic mesophases from the SAXS Bragg peaks found. The Q_main function runs two other functions: Q_possible_phases, and Q_projection_testing, both of whose variables are passed from the Q_main. Q_main expects four variables:

peaks : The array of peaks found in the data, from as determined by the

bin_factor : A factor to determine the widths of the bins in a historgram used for finding lattice parameters of cubic phases. Doesn't have to be big, 2 should suffice.

threshold : A value for which a histogram bin has to be populated more than for confirming that a phase is valid.

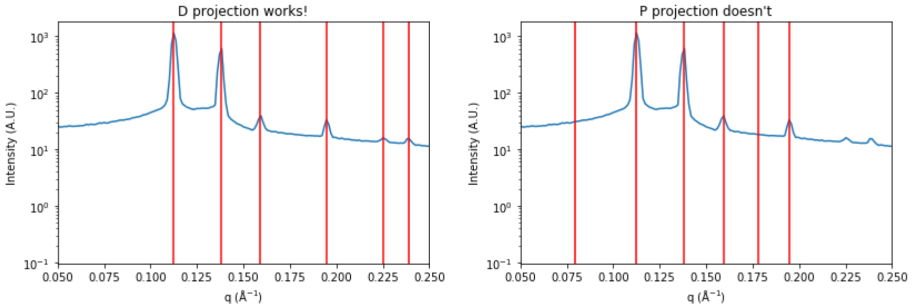
lo_q : should be the same as the lo_q value in the finder.py.

Q_main begins by passing all the **peaks**, **bin_factor**, and **threshold** variables to the Q_possible_phases function. Q_possible_phases begins exploring the entire range of possible lattice parameters that could be found from the data, by creating x_init: a set of multidimensional arrays which simultaneously index all peaks as all x indices. These arrays are then concatenated together. Additionally, n_x are created, along with n. These are arrays of integers running the length of the individual arrays, and the concatenated array. They are later used in order to track back where values in the x_init have originated from: the specific peak and index from which the value arose.

At this point, the frequency of the possible lattice parameters are examined by a histogram: this is where the variable **bin_factor** is used. **bin_factor** is used to determine the widths of the bins, which is defined as the product of the number of possible values and the variable. A value of 2 should suffice for this purpose. As Fig. 2 demonstrates, when a peak has been correctly indexed to calculate the lattice parameter, there are more peaks present in the bin, and therefore the frequency count is higher. It is for this reason a **threshold** variable is passed to Q_possible_phases: if the number of potential lattice parameters in a bin is lower than the threshold, then the bin is discarded, as the indexing has not been correct.

If the number of values in the bin exceeds the **threshold** parameter, then the contents of the bin are examined further. The factors contributing to the height of the bin are traced back to the phase array, peak, and indexing from which they arose. That is to say, across the three cubic phases, it is inevitable that the 'correct' index has been used for a peak: as noted previously, the factors $\sqrt{6}$ and $\sqrt{8}$ appear in all three cubic phase indexings, and consequently, regardless of whichever cubic phase the set of peaks has arisen from, they will both make a 'correct' contribution.

The individual factors are collected separately. At this point, the number of contributing correct indices is used as the comparison for which phase it is most likely to have arisen from. As a result



(a) The projected peaks from a Q_{II}^D phase proposed by Q_possible_phases match those found in the data. Note here that for this example, the $\sqrt{4}$ peak was not actually found by the finder.py module, and nor has it been found by the projection shown in this figure, which must be taken care of when considering the acceptable threshold of missing peaks.

(b) The projected peaks of a Q_{II}^P phase fall into gaps in the data, and do not index the final two peaks at all.

Figure 3: Projections of peaks from the proposed phases by the Q_possible_phases function across the data being tested.

of the high coincidence between the factors of the Q_{II}^D and Q_{II}^P phases, they are examined slightly separately. Firstly, it is assumed that 4 peaks is the minimum number required to assign a phase by this point, for any phase. Subsequently, if this condition is met by both the Q_{II}^D and Q_{II}^P contributing factors, they are examined. It would be expected that:

1. The correct assignment of the peaks contains more correct peak indices.
2. The incorrect assignment of index and peak pairs is entirely contained in the correct assignment.

These factors are tested for, and the correct assignment between the Q_{II}^D and Q_{II}^P phases is made based on which conditions are met.

The function described above, Q_possible_phases, returns a dictionary of possible phase assignments back to the Q_main function of the *phase_ID* script. At this point, another check is made, cross referencing the set of assigned peaks with the peaks actually present in the data, by the Q_projection_testing function. The function firstly generates a set of peaks based on the lattice parameter and assigned phase, as worked out from the first Q_possible_phases routine. The function then firstly checks that the first peak of this generated set of peaks falls within the user-determined q search range as initially defined. Secondarily, the function cross references the generated set of peaks (all or most of which will be completely coincidental with the 'real' peaks in the data) with the peaks present in the data.

That is, there are two sets of peaks to consider: one from the actual data itself, and a set of peaks that can be generated by knowing both the phase and lattice parameter of an assigned set of peaks. These latter set of peaks are tested against the peaks that have been similarly assigned to the phase. By identifying where the generated peaks match up to the data, the number of peaks that have been correctly assigned to the phase can be identified. This is apparent in Fig 3, where it can be seen the there is a single Q_{II}^D phase in the data, which has been identified as either Q_{II}^D or Q_{II}^P by the Q_possible_peaks function. The $\sqrt{2}$, $\sqrt{3}$, $\sqrt{6}$, $\sqrt{8}$, and $\sqrt{9}$ peaks were found in the data by the finder.py script, and have been correctly indexed in the case of the Q_{II}^D possible assignment. In the Q_{II}^D projection, the $\sqrt{4}$ peak has additionally been projected, so that visually it matches up in Fig 3a, but is not actually identified for comparison purposes. However, as the first peak is within the accepted search range, and all but one of the peaks are present in the data, the phase assignment (complete with peak indexing) has been confirmed as correct. In the case where the Q_{II}^P phase has been tested, as shown in Fig 3b, it can be seen the there are visual matches on four of the peaks, whilst the proposed $\sqrt{10}$ peak of the Q_{II}^P phase falls at a point where the isn't a peak either in the data, let alone by visual inspection. Notably in this case, it is the $\sqrt{4}$, $\sqrt{6}$, and

$\sqrt{16}$ peaks that have been correctly assigned - and matching up with the correctly assigned peaks a factor of $\frac{1}{\sqrt{2}}$ less.[2]

Therefore, the entire set of possible cubic phases have been tested and checked for correct assignment, so if any are present in the dataset of peaks provided to the respective functions, then they should have hopefully been identified by this point.

3.2 Lamellar and Hexagonal phases

As previously highlighted, it might reasonably be expected that for powder scattering, the number of peaks identified in the data will be fewer for non-cubic phase patterns than for cubic-phase structures. If this condition is met in the first instance, then instead of the cubic-phase routines being carried out, a shorter, separate, routine for identifying either L_α or H_{II} mesophases is used. The La_HII_possible_phases function expects only two variables:

peaks : The array of peak positions

bin_factor : A factor to determine bin width of the histogram used for finding lattice parameters of phases.

Overall, the function uses a similar statistical frequency method to the cubic phase routines, but simply distinguishes between which phase has been used by which is the more common value. The incidence between the possible L_α and H_{II} peak assignments coincide such that this statement should always be true if the phase provided to the routine is indeed one of those phases.

4 Acknowledgements

CB thanks Adam Squires and Tim Snow for their invaluable feedback into the development of the methodology.

References

[1] C. V. Kulkarni, W. Wachter, G. Iglesias-Salto, S. Engelskirchen, and S. Ahualli, "Monoolein: a magic lipid?," *Phys. Chem. Chem. Phys.*, vol. 13, pp. 3004–3021, 2011.

[2] M. Newville, T. Stensitzki, D. B. Allen, and A. Ingargiola, "LMFIT: Non-Linear Least-Square Minimization and Curve-Fitting for Python," Sept. 2014.

[3] H. Kim, Z. Song, and C. Leal, "Super-swelled lyotropic single crystals," *Proceedings of the National Academy of Sciences*, vol. 114, no. 41, pp. 10834–10839, 2017.

[2] In fact, in the case of these data, the Q^P_{II} test failed at the first hurdle: the first peak was outside of the search range.