X-ray coherence: imaging techniques and applications

Julio Cesar da Silva
email: julio-cesar.da-silva@neel.cnrs.fr
website: https://sites.google.com/view/jcesardasilva

March 24, 2021

Abstract

The goal of this tutorial is to introduce the basics of coherent X-ray imaging, including isolated-specimen Coherent Diffraction Imaging (CDI) and Ptychography. We will learn how to design an experiment, how to check the sampling conditions and some basics of the phase retrieval algorithms. Since tutorial is limited in time, the present notes provide additional information in a very brief way. For educational use only, basic simulations to assist our understanding of the concepts are provided in **Jupyter Notebooks in Python3**.

1 Starting Jupyter Notebook with Python3

Throughout this tutorial, we will run some python scripts using Jupyter Notebook [1]. The website [1] shows how to install it and to start it in the different Operational Systems (Linux/Mac OS, Windows).

Additionally, some Python packages are required: Numpy, IPython, Matplotlib, and scikit-image. If you do not have such packages installed, I recommend the installation via pip install:

```
pip3 install --user numpy, ipython, matplotlib, scikit-image
```

The Jupyter Notebooks we will use in this tutorialare available through my ESRF Gitlab repository:

https://github.com/jcesardasilva/tutorialHercules

You can either download the files and copy them in a folder of your choice or, if you prefer, you can clone my repository by typing:

```
git clone https://github.com/jcesardasilva/tutorialHercules.git
```

Once the files are in your computer, you can open Jupyter Notebook by:

- Windows: click on the Jupyter Notebook icon installed in the start menu.
- Linux/Mac OS: open the terminal and type `jupyter notebook` at the prompt.

You can also use MyBinder and run the notebooks there:

```
https://mybinder.org/v2/gh/jcesardasilva/tutorialHercules.git/v0.1
```

There three notebooks there, which we will run in this order: 1) `Oversampling.ipynb`; 2) `CDI_ER_HIO.ipynb`; and 3) `PIE_ptycho.ipynb`.

It is important to mention that the Python codes in notebook 1 and 2 are inspired by the "Tutorial in Diffraction Imaging" by Gösta Huldt and Filipe Maia (originally in MATLAB), which was

available until recenlty at http://xray.bmc.uu.se/~ekeberg/molbiofys/tutorial.pdf. Modifications have been made on the original code for educational reasons and Python compatibility. The Python implementation in notebook 3 is inspired by the MATLAB code available in the Appendix A of the Diploma thesis by Dr. Martin Dierolf, which is available here: https://www.psi.ch/sls/csaxs/PublicationsEN/thesis_dierolf.pdf.

The present notes are intended for educational reasons only. They are not be perfect, but keep been gradually improved. Apologies for possible mistakes and missing references. Additionally, the codes in the Python notebook are not intended to be efficient and should not be used for goals other than educational. There are other more efficient and complete codes available, such as Ptypy (ptycho.github.io/ptypy/) and PyNX (www.esrf.eu/computing/scientific/PyNX/README.html).

2 Oversampling of the speckle patterns

The concept of oversampling of the speckled diffraction pattern is the key to allow the phase retrieval in coherent diffraction imaging. In this section, we will discuss the size of the speckles, how they relate to the sample, and the sampling requirements to oversampling the speckles.

2.1 Size of the speckles

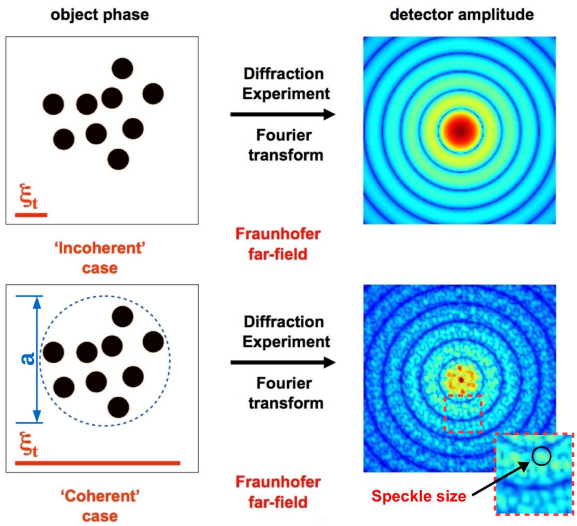

Figure 1: Top: incoherent scattering. Bottom: coherent scattering

Figure 1 illustrates the difference between incoherent scattering and coherent scattering. We see that when the transverse coherence ξ_t of the incident beam is smaller than the sample size, no speckles are visible on the detector. On the other hand, when ξ_t is larger than the sample, we can record the speckle pattern on the detector.

The angular opening of each speckle is given by:

$$\Delta\theta = \frac{\lambda}{a}, \tag{1}$$

where λ is the the wavelength of the incident beam and a is the size of the support of the sample. Thus, the size of a speckle at a sample-to-detector distance z is given by:

$$\Delta S = \frac{\lambda z}{a}. \tag{2}$$

2.2 Sampling requirements

Figure 2 illustrates the oversampling of the intensities. The Shannon-Nyquist sampling tells the minimum sampling rate of a signal such that it can be well represented in the discrete world. This means, the minimum sampling rate that allows us applying a Discrete Fourier Transform (DFT) to a signal and recover it back by applying an Inverse Discrete Fourier Transform (IDFT).

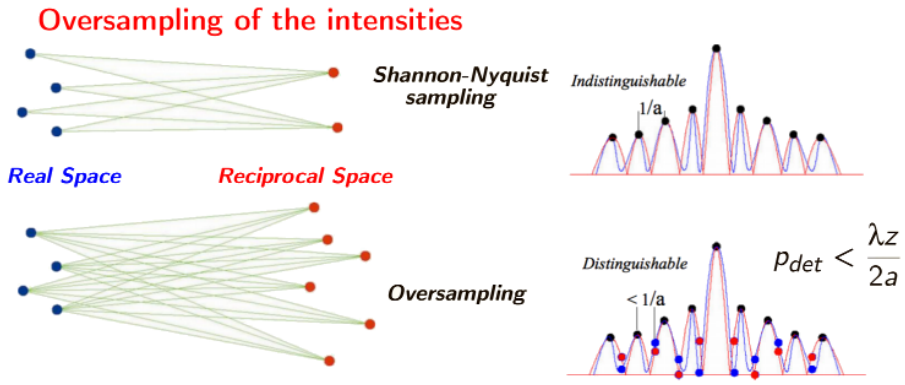

Figure 2: Oversampling of the intensities.

This indicates that the intensities must be sampled with a pitch of the size of a speckle. However, we are not able to sample well each speckle by doing so. Thus, we need to increase the number of correlated points between the real and the reciprocal space by oversampling the diffraction pattern.

Such an oversampling also provides additional information that allows us retrieving the lost phases from the intensities. Therefore, we need to have at least two pixel (or more) per speckle size, which means that the detector pixel size must be:

$$p_{det} < \frac{\lambda z}{2a} \tag{3}$$

where λ is the wavelength, z is the sample-to-detector distance, and a is the size of the support of the sample.

Jupyter Notebook: `Oversampling.ipynb`

Let us start using the Jupyter Notebook. The notebook `Oversampling.ipynb` illustrates the oversampling of the diffraction patterns.

1. Launch the Jupyter Notebook.
2. Open `Oversampling.ipynb`.

The notebook tries to be self-explanatory and you can go through it. To run one cell, type `Shift+Enter`. Remember of running cell after cell from the top to bottom, because it happens that a block of code at a certain cell depends on the results from a previous cell. Here we are going to play with a molecule, in particular the caffeine. Experimentally, we cannot perform a single

molecule CDI experiment, but we hope one day this will be possible. For now, we are going to suppose it is possible.

When running the notebook, you can play with the with the oversampling factor and see how the diffraction pattern is sampled. Some task you can do are:

1. Try to find the size of the support of the sample. What is the width of the caffeine molecule?

2. Try to see how the diffraction patterns look like when the oversampling factor is 1. Then, you can gradually increase and watch the diffraction pattern. What changes can you observe in the diffraction pattern by doing this? Can you see better the speckles?

3. When you play with different oversampling factors, what can you observe with respect the size of the field-of-view relative to the sample size?

Finally, after playing with the notebook, the last cell allow you to save your intensities and your density map ($\rho_{original}$). Save the files, because we will use them in the next section. Remember that the intensities need to be over-sampled. (Hint: use an oversampling factor of 6 and a cutoff of 4. Previous tests showed those values are very efficient for what we will do).

Optional: If you have a computer with enough RAM memory, try to repeat the steps above with the molecule of Lysozyme. Do not try to do it in a computer with not enough memory because it can get stuck.

3 The phase retrieval algorithms in isolated specimen CDI

The iterative algorithms are the heart of the coherent diffraction imaging techniques. Many different algorithms have been proposed, but the fundamental ones are the Error Reduction (ER) and the Hybrid Input-Output (HIO) algorithms.

3.1 Error reduction (ER) algorithm

The Error Reduction algorithm is a classical iterative algorithm. It was proposed by Gerchberg and Saxton in the 70's and thus is also know as Gerchberg-Saxton algorithm [2]. Figure 3 illustrates the engine of the ER algorithm. Briefly, ER works as follows:

1. We start with a complex function filled with random numbers (or ones) representing the object first guess, which obviously does not look like the object at this point.

2. We then apply a Fourier Transform to such a function, which will provide a reciprocal-space function that does not correspond yet to the Fourier transform of our object.

3. We now apply the **reciprocal-space constraint**: We know that the square root of the intensity correspond to the amplitude of such a function in the Fourier (reciprocal space). We then replace the amplitude of our reciprocal-space function by the square root of the experimental intensities, but keep the phases untouched.

4. At this point we have an updated reciprocal-space function which is a bit closer to the experimental data. We then apply an inverse Fourier transform to such a function to obtain a new version of the real-space function describing our object.

5. We now apply the **real-space constraints**: Since the object needs to be isolated for CDI to work, we have zeros outside the object support. Additionally, we know that the electron density of the object must be positive. Therefore, we apply both constraints to the real-space function and obtain an update function a bit better than the initial guess.

6. We then continue to cycle between Real and Reciprocal spaces through Fourier Transforms while imposing the constraints of each space until convergence is reached. The resulting real-space function will be the one better describing the object or, at certain extent, a very good approximation of it.

The tricky part is to find the support of the sample. *A priori* information about the object can help at this point. Otherwise, we can use the autocorrelation function (Patterson function) to obtain a guess of such a support as we will see in the next section.

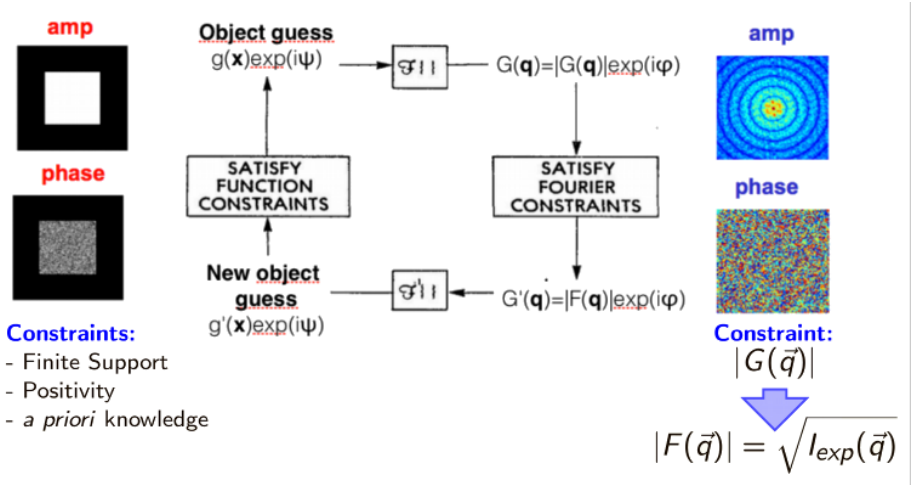

Figure 3: Error reduction iterative algorithm.

3.2 Sample support from autocorrelation function

If we Fourier transform the diffraction pattern intensities, we obtain the autocorrelation of the electron density, which is also called Patterson Function:

$$\mathscr{F}\{I(\vec{q})\} = (\rho \star \rho)(\vec{r}) = \int \rho(\vec{r}\prime + \vec{r})\rho(\vec{r}\prime)d\vec{r}\prime \tag{4}$$

The autocorrelation function is common among crystallographers and can help to solve small molecules. However, here we are not dealing with crystalline samples, but the autocorrelation function can help us in a different way. First, we can obtain it by simply applying a Fourier transform to the intensities. And second, the support of the autocorrelation function is twice larger than the support of the each function being correlated, which is the electron density ρ with itself in our case. Thus, we can obtain an initial guess of our sample support by applying a threshold to the autocorrelation function. We will soon see how this is implement in practice when we will play with the next Jupyter Notebook.

3.3 Hybrid Input-Output (HIO) algorithm

The ER algorithm is a great algorithm, but it is prone to stagnation and it tends to converge very slowly, sometimes requiring tens of thousands of iterations to reach an accurate result. The problem comes mostly from the fact that we bring the value outside the sample support directly to zero at each iteration of ER.

Fortunately, a very smart modification of algorithm can speed up the convergence significantly. Rather then applying the support constraint directly, we reduce smoothly the values outside of the support constraint as each iteration. This is the idea behind the Hybrid Input-Output algorithm (HIO) [3,3]. This is the basic scheme of the HIO algorithm:

$$\begin{aligned} F_k &= \mathscr{F}(\rho_k) \\ F'_k &= FourierConstrain(F_k) \\ \rho'_k &= \mathscr{F}^{-1}(F'_k) \\ \rho_{k+1} &= RealSpaceConstrain(\rho_k) \end{aligned} \tag{5}$$

In the HIO algorithm, the last step above is modified as follows:

$$\rho_{k+1} = \begin{cases} \rho'_k, & \text{where } \rho'_k \text{ satisfies the constraints} \\ \rho_k - \beta\rho'_k, & \text{where } \rho'_k \text{ does not satisfies the constraints.} \end{cases} \tag{6}$$

where β is a constant that will play a role in the convergence and stability of the the algorithm and typically $0 < \beta < 1$.

Jupyter Notebook: `CDI_ER_HIO.ipynb`

Let us start using the Jupyter Notebook. The notebook `CDI_ER_HIO.ipynb` provides some simulation of the CDI using the ER and HIO algorithm.

1. Launch the Jupyter Notebook.
2. Open `CDI_ER_HIO.ipynb`.

Again, the notebook tries to be self-explanatory and you can go through it. Please, remember that the block of code at a certain cell depends on the results from a previous cell.

You will implement the ER and the HIO algorithms using the over-sampled intensities you saved to file at the end of the section 2 after playing with the first Jupyter Notebook. For the ER algorithm, we keep simple and obtain the sample support by applying a threshold to the $\rho_{original}$ we previously saved to file. For the HIO algorithm, we will use the support obtained from the autocorrelation function as explained in the subsection 3.2. The HIO constraints from equation 6 are implemented in the function `getViolations`.

Some task you can do are:

1. Run ER and HIO algorithm one after the other. Can you observe any difference in the results? Do you think the results are both good or one is better than the other?
2. Try to increase the number of iterations in each algorithm. Do you see any improvement in the results? Is there again any difference between the results obtained with ER and the ones obtained with HIO?
3. Is the support obtained from the autocorrelation function good enough?

4. Come back to the first notebook, `Oversampling.ipynb`, try a different oversampling factor, save the files and run again the ER and HIO. What do you observe when the oversampling factor decreases or increases?

5. How could you optimize the convergence of the algorithm, avoiding local minimas?

4 The ptychographic phase retrieval

Although a fantastic technique, CDI has some limitations such as the need of isolated specimen, which restricts its application to small samples only, the algorithms are prone to stagnation and it requires detectors with pixel sufficiently small to over-sample the diffraction patterns. The oversampling factor needs to be most times much larger than 2. Therefore, extended samples cannot be imaged by CDI.

Different from CDI, Ptychography is a coherent imaging technique that can be applied to extended samples. Although proposed in the 70's, about 20 years before CDI that appears at the end of the 90's [5, 6], only when the iterative phase retrieval algorithms of CDI appears and inspired the use of iterative algorithms in Ptychography that such a technique became practical. This happened around 2004 [7], but only in 2007 that the first demonstration of Ptychography was made in synchrotrons [8].

There are already several different iterative phase retrieval algorithms for Ptychography, but the most common ones are the Difference Map (DM), the Ptychographic Iterative Engine (PIE), and the Maximum Likelihood (ML). At this tutorial, we will focus only at the PIE algorithm for simplicity.

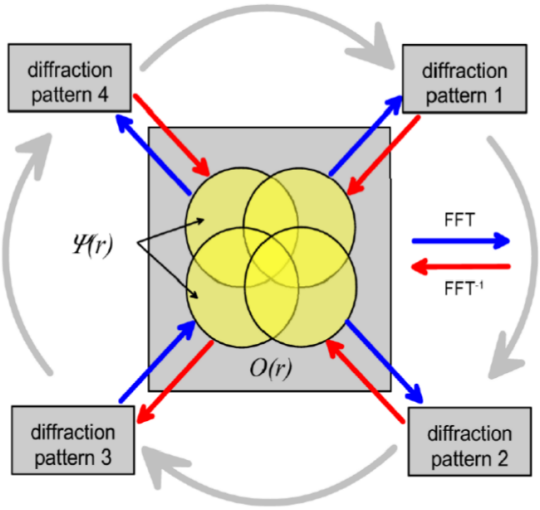

Figure 4: The Ptychographic Iterative Engine (PIE).

Figure 4 illustrates the PIE algorithm. The algorithm reinforces the consistent between the each scan position and the corresponding diffraction pattern, as well as the consistent between adjacent overlapping positions. The constraints in the Fourier space are applied similar to the CDI at each scan position. The overlap constraint ensures that the overlapping regions agree and the incident wave field is unique.

Since there is overlap between the adjacent scan positions, there is redundant information which is exploited to yield not only the phases by also the amplitudes. Therefore, different from

CDI, ptychography provides the information about the complex refractive index of the sample $n(\vec{r}) = 1 - \delta(\vec{r}), + i\beta(\vec{r})$, where $\delta(\vec{r})$ is the refractive index decrement, related to the phases, and $\beta(\vec{r})$ is the absorption index, related to the amplitudes.

4.1 The ptychographic sampling requirements

The ptychographic sampling requirements are different from those of isolated-specimen CDI. The additional overlapping constraints adds robustness to the phase retrieval algorithm. Thus, the sampling of the ptychographic data needs to be seen jointly between the real and the reciprocal spaces.

Figure 5 shows the joint real-reciprocal phase space we use to derive the sampling requirements of a ptychographic experiment. The sampling of the diffraction pattern is the similar to the one in equation 3. However, if such a sampling is not respected, as long as there is no gaps in the joint real-reciprocal phase space, Ptychography will still work, because the upsampling of the Fourier space can be compensated by the overlapping in the real space.

Let us define the scanning steps R for ptychography as a fraction α of the beam size D at the sample position, such that:

$$\alpha = \frac{R}{D}. \tag{7}$$

Let us also define a factor β that represent the inverse of the oversampling factor $\mathcal{O}$ as follows:

$$\beta = \frac{1}{\mathcal{O}} = \frac{p_{det} D}{\lambda z}, \tag{8}$$

where p_{det} is the detector pixel size and z is the sample-to-detector distance.

Therefore, if there is no gaps in the filling of the joint real-reciprocal phase space represented in Figure 5, the ptychography sampling requirements are respected. Since we are dealing with complex quantities (real and imaginary parts), we need twice more information, which means the filling must be done with some redundancy. Mathematically, this can be described by ensuring that the one-dimensional oversampling ratio, defined as $1/(\alpha\beta)$, respects the following condition:

$$\frac{1}{\alpha\beta} > 2. \tag{9}$$

Jupyter Notebook: `PIE_ptycho.ipynb`

The notebook `PIE_ptycho.ipynb` provides a simulation of a ptychography experiment using PIE for the phase retrieval. Note that this corresponds to the PIE algorithm, which only retrieves the object, and not to the extended-PIE algorithm, which retrieves the object and the probe simultaneously.

1. Launch the Jupyter Notebook.
2. Open `PIE_ptycho.ipynb`.

Again, the notebook tries to be self-explanatory and you can go through it. Please, remember that the block of code at a certain cell depends on the results from a previous cell.

Some task you can do are:

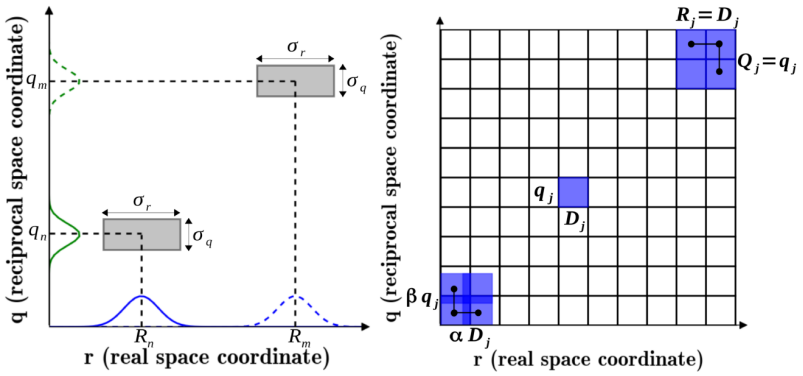

Figure 5: The joint real-reciprocal phase space for ptychography. Left: The boxes filling of the phase space respect the Heisenberg uncertainty's principle. Their size does not depend on their translation on the phase space. Right: the four boxes at the bottom-left show the case where the ptychographic sampling requirements are respected, while the ones at the top-right show the case where there is no overlapping between adjacent spots. D_j is the size of the beam at the sample position, q_j is the sampling pitch of the Fourier Space, R_j is the scanning steps, and Q_j is the sampling pitch of the Fourier space when there is oversampling of the diffraction patterns.

1. Try different beam sizes. Check if the sampling requirements are respected. What did you observe?
2. Try to under-sample the diffraction pattern, but keep condition 9 still valid. Can you still recover the image?
3. Change the number of iteration and see if it improves the reconstructed image.
4. When you change the beam size, does the pixel size of the reconstructed image also change?
5. Do you think the spatial resolution of the reconstructed image changes with the beam size at the sample position?
6. How does the pixel size of the reconstructed image calculated?
7. Which part of the code could be optimized for faster convergence? Could you please tell us how you would do?

References

[1] Jupyter/IPython Notebook. Website: https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/index.html. Accessed on January 25, 2019.

[2] R. W. Gerchberg and W. O. Saxton, Optik 35, 237 (1972).

[3] J. R. Fienup, Appl. Opt. 21, 2758 (1982).

[4] J. R. Fienup, J. Opt. Soc. Am. 3, 27 (1978).

[5] J. Miao, P. Charalambous, J. Kirz, and D. Sayre, Nature 400, 342 (1999).

[6] J. Miao, D. Sayre, and H. N. Chapman, J. Opt. Soc. Am. A 15, 1662 (1998).

[7] H. M. L. Faulkner and J. M. Rodenburg, Phys. Rev. Letter 93(2), 023903-1 (2004).

[8] J. M. Rodenburg, A. C. Hurst, A. G. Cullis, B. R. Dobson, F. Pfeiffer, O. Bunk, C. David, K. Jefimovs, and I. Johnson, Phys. Rev. Letter 98, 034801 (2007).

[9] J. C. da Silva and A. Menzel, Opt. Express 23(26), 33812 (2015).