UNIVERSITÀ DEGLI STUDI DI MILANO

Scuola di Dottorato in Fisica, Astrofisica e Fisica Applicata

Dipartimento di Fisica

Corso di Dottorato in Fisica, Astrofisica e Fisica Applicata

Ciclo XXXI

High Performance Computational Intelligence for Coherent Diffraction Data Analysis and Imaging

Settore Scientifico Disciplinare FIS/03

Supervisore: Professor Davide Emilio GALLI

Coordinatore: Professor Francesco RAGUSA

Tesi di Dottorato di:

Alessandro COLOMBO

Anno Accademico 2017-2018

External Referees:

Marchesini Stefano
Lawrence Berkeley National Laboratory, 1 Cyclotron Rd, Berkeley, CA 94720, USA
smarchesini@lbl.gov

Maia Filipe
Uppsala University, Department of Cell and Molecular Biology, Molecular Biophysics, Uppsala Biomedicinska Centrum BMC, Husarg. 3 751 24 Uppsala
filipe.maia@icm.uu.se

Commission of the final examination:

Prof. Rossi Giorgio
Università degli Studi di Milano - Dipartimento di Fisica, via Giovanni Celoria 16, 20133 Milano, Italy

Prof. Parola Alberto
Università degli Studi dell'Insubria - Dipartimento di Scienza e Alta Tecnologia, Via Valleggio 11, 22100 Como, Italy

Dr. Grisenti Robert
Institut für Kernphysik, Universität Frankfurt, Max-von-Laue-Str. 1, 60438 Frankfurt am Main, Germany

Final examination:

December 14$^{\text{th}}$ 2018

Università degli Studi di Milano, Dipartimento di Fisica, Milano, Italy

To my little Cecilia
(and to my wife Federica, otherwise she gets angry)

Cover illustration, internal illustrations and design:
Alessandro Colombo

MIUR subjects:
FIS/03

PACS:
42.30.Rx

Contents

List of Figures

Introduction

Since the beginning of the 19^{th} century, Science have been made many steps thanks to the ability to deeply investigate matter. Direct observations started with optical microscopes, able to resolve up to some hundreds of nanometers, while nowadays aberration-corrected Transmission Electron Microscopes go down to the ångström scale (Williams & Carter 1996; Lentzen et al. 2002). These direct imaging approaches are based on lenses systems that highly influence the microscope performance, due to aberration effects and imperfections intrinsic in the manufacturing process that limit the achievable resolution. Aberrations can be mitigated, for example, by means of aberration correctors (Batson et al. 2002; Spiller et al. 1994), while, when high energy photons are used (hard X-rays), the manufacturing of lenses becomes a hard, and sometimes impossible, task (Lengeler et al. 1999). These issues do not affect imaging techniques that don't exploit lenses, the so-called lens-less imaging techniques. Among them, Coherent Diffraction Imaging (CDI) (for some examples see Miao et al. 1999, 2002; Zuo et al. 2003; Marchesini et al. 2003) is revealing to be able to provide quantitative images of matter at a resolution depending only on the radiation wavelength. As its name well describes, CDI is an imaging technique that exploits the diffraction phenomena, i.e. the detected radiation scattered by the interaction with the specimen of interest, acquired in far–field condition (Born & Wolf 1999). This radiation must be coherent and monochromatic, which means that particles (photons or electrons) must have both the same wavelength and the same phase. Here the first issue of CDI resides: it requires special radiation sources, with respect to the simpler ones required for lens-based imaging. The second not trivial point resides in the diffraction phenomena. The detected radiation diffracted by the sample, i.e. the diffraction pattern, cannot be trivially linked to an image of the specimen. In particular, given the sample density distribution $\rho(\vec{x})$, the measured diffraction pattern $I(\vec{q})$, acquired in far-field conditions, is proportional to the square modulus of the Fourier Transform (FT) of $\rho(\vec{x})$, that is $I(\vec{q}) = C \times |\tilde{\rho}(\vec{q})|^2$. Directly performing an inverse FT to retrieve $\rho(\vec{x})$ isn't a viable way, because only the module of $\tilde{\rho}(\vec{q})$ is known, while its phase is lost. Thus, the phase must be retrieved by solving the so-called *phase retrieval problem* (Taylor 1981): the quality of CDI results strongly depends on the quality of the retrieved phase. In principle, given a diffraction pattern $I(\vec{q})$, there are an infinity of $\rho^i(\vec{x})$ such that the square modulus of their FT coincides with the experimental measure, causing the *phase retrieval problem* to be undetermined. In order to restore the possibility of retrieving the phase, a further knowledge on the sample must be added. For example, Holography exploits an a-priori knowledge in the real space (Chapman & Nugent 2010), achieved also with very sophisticated techniques (for example as done by Gorkhover et al. 2018). An other

example is Ptychography (Zheng et al. 2013; Yang et al. 2011), a scanning technique that exploits redundancies in the diffraction data.

The original CDI technique, instead, requires a further constraint to the problem, known as *support* constraint (Fienup 1987), which limits the extension of $\rho(\vec{x})$, exploiting the so–called *oversampling* method (Bates 1982). Under well-defined mathematical conditions, the phase retrieval problem has a unique solution. In this way, the correct solution is in principle defined as the one whose Fourier Transform is compatible with the experimental diffraction pattern and which totally resides inside the area defined by the support function. In mathematical terms, the aim of phase retrieval is to find the intersection between the set of densities that satisfy the *experimental constraint* and the set of densities that fulfill the *support constraint*. This standard CDI approach is among the few techniques capable to provide time-resolved imaging of matter with a resolution limited only by the probe wavelength (Barty et al. 2008), and it well suits the so–called *diffract and destroy* imaging experiments performed in the most recent X-ray Free Electron Laser facilities (Spence 2008). This work is, thus, focused on this *standard* CDI approach, and, from now on in this dissertation, the term CDI will refer to this last technique.

The *phase problem* in CDI is much harder than it may appear, mainly for two reasons: experimental errors, intrinsic to the diffraction measurement, and the large amount of unknowns to be retrieved. The first one, which can be addressed as *experimental issue*, makes those two constraints not intersecting, such that the *ideal* solution doesn't exist anymore, and it is redefined as the one which optimizes the distance between the two constraints. This distance is interpreted as the *error* of the reconstruction (for a review see Marchesini 2007). This turns the *phase retrieval problem* into an optimization problem of a quantity (the distance of the constraints) that assumes an unknown value at the global optimum, such that distinguishing the global minimum among local ones it's like to find a needle in a haystack. The latter issue, let's call it *dimensional issue*, gives a prohibitive size to this haystack. The usual number of parameters involved in phase retrieval goes from 10^5 to 10^8, slowing down both the search process and the solution identification.

The phase retrieval problem can be easily imagined as a golf course, where reaching the solution is as like as putting the ball in the golf hole. The golfer (or, better, the phase retriever) is a blind player, able to only measure the altitude of its position. The only knowledge he has about the hole (i.e. the solution) is that it is placed at the point with the lowest altitude.

If, despite everything, Coherent Diffraction Imaging exists, it is thanks to the early works of J. R. Fienup in 1970's, where he adapted the Gerchberg and Saxton's algorithm (Gerchberg & Saxton 1972), conceived for Electron Microscopy, for the reconstruction of a spatial distribution from its Fourier Transform amplitude. This algorithm, known as Error Reduction (ER), cyclically imposes the problem constraints on a given initial guess of the solution, and its name derives from its peculiarity to always reduce the error value. Even if, at a first sight, this approach seems to be the definitive solution to the problem, ER algorithm is a striking example of phase retrieval complexity. If we come back to the golf player, solving the problem via the ER approach is as like as trying to reach the hole by simply placing the ball on the ground and letting the gravity force act. Moreover, this ball has no momentum and exactly follows the gradient of the gravity potential. At this point, it is clear that this approach is not viable, unless we are really close to the solution or the playing field is more similar to a bathtub rather than a golf course.

For these reasons, along with the ER algorithm, Fienup proposed an enhanced approach called Hybrid Input-Output (HIO) (Fienup 1978). Its name derives from the fact that, along with the gradient value, also the position assumed in the previous step of the algorithm is exploited. Speaking in terms of golf, it's like the ball has now a sort "kinetic

energy", and it is able to escape from small depressions (or turn around in the case of a bathtub). This is clearly an improvement, such that HIO algorithm is one of the most exploited algorithms nowadays. However, not all that glisters is gold, and HIO has its drawbacks. First of all, small changes in the initial position may lead to very different final phases. Second, its ability to escape holes holds also for the solution, such that HIO can pass near the solution but then go away.

Many people managed to find out smarter golf balls (i.e. better phase retrieval algorithms), also with good results and also by alternating different algorithms during the retrieval procedures, but, at the end of the story, it turns out that the problem resides in the golf player rather than in the ball. In real cases, the golf course is too wide for a single blind player and information at his disposals are too few.

An usual way to come nearer to the solution is to recruit a team of golfers, scatter them randomly in the golf course, and let them throw their own golf balls. This approach is very close to what is commonly known as *random search* approach for optimization problems (Matyas 1965). Obviously, it has a cost, as the salary of the team grows with the number of members. In the phase retrieval world, this salary is translated into *computational cost*. This issue remained insurmountable for many years, until recent developments in computing hardware: nowadays High Performance Computing (HPC) hardware makes possible to perform many phase retrieval procedures in parallel, opening the door to the *random search* approach for *phase retrieval*. Here we have reached the state-of-the-art of phase retrieval, which is currently faced by performing many phase retrieval procedures with different starting guesses, and only the ones with the best performance are selected (and often averaged) to define a final result. Again, this is not the end of the story; *random search* approach doesn't give ultimate results and further developments are required for a real "full throttle" phase retrieval.

Coming back to the golfer team, their blindness has been already justified as their total ignorance about the solution direction. It has been assumed also, without having said it explicitly, that they are mute, such that they cannot communicate each other. Let's now imagine that these golfers have the chance to communicate the information they own (actually the altitude and the position of the ball in the field). It becomes clear that, if they were able to exchange news about their performance, at every time they would walk up towards the colleague that is supposed to be the nearest to the hole. Even if in the golf world the communication by voice is costs–free, in the phase retrieval one it implies a *communication cost*, that is hardware time spent for communication, instead of computation. Thus, this PhD thesis focuses on the designing, implementation and characterization of a "communication-capable" *heuristic optimization algorithm* for the *phase retrieval problem* in Coherent Diffraction Imaging, called Memetic Phase Retrieval. Its conceptual structure, which belongs to the category of Evolutionary Algorithms, will be described, along with details on its implementation, able to exploit the most recent HPC hardware. Satisfactory results will be shown for 2D and 3D Coherent Diffraction Imaging, on both simulated data and experimental diffraction patterns.

Let's proceed to give a voice to blind golfers.

Thesis Outline

Chapter 1 starts with a general discussion about Coherent Diffraction Imaging and its motivations. After a detailed description of the *phase retrieval problem* and the related issues, the speech will introduce the most common phase retrieval algorithms. The last section discusses about the usual way to use these algorithms and its limitations, which are the main motivations of this work.

The following Chapter 2 is the core of the work, and it is dedicated to a description of the Memetic Phase Retrieval approach, which represents a new way to exploit the existent phase retrieval algorithms. After a formal depiction, the speech focuses on its practical implementation and the description of its relevant parameters. The last part deals with the parallel programming techniques exploited to get the best performances on High Performance Computing hardware and some scaling tests, performed on the most recent computing infrastructures, are presented.

Chapter 3 provides a characterization of Memetic Phase retrieval, performed by using simulated data. In the first part, Memetic Phase Retrieval behavior is studied as function of its parameters. The second part shows the performance of MPR when treating data that presents some critical issues, like noise or lack of information. Comparisons with the results of the standard approach to the phase problem are shown.

Chapter 4 focuses on Memetic Phase Retrieval results on real experimental data. The first part is dedicated to 2D Coherent Diffraction Imaging results, with some comparisons with the standard approach. The second part, instead, describes imaging results on three dimensional diffraction data, showing how the approach can face the critical issues that affect this kind of diffraction patterns.

The last Chapter 5 doesn't deal with Memetic Phase Retrieval. Instead, it focuses on side works that were done during the PhD, still on diffraction data analysis. In particular, the first part describes a software that was developed with the aim to provide an online analysis on a diffraction experiment performed at a Free Electron Laser, in order to give a "live" view on the progress of the experiment. The second part, which is related to the previous one, consists in the description of a software developed for the purpose of correcting artifacts introduced in the diffraction patterns by the presence of a Micro-Channel Plate detector.

Chapter 1

Coherent Diffraction Imaging

1.1 Why Coherent Diffraction Imaging

The history of Coherent Diffraction Imaging is strictly bounded to the history of Crystallography. The great success of the latter led one of its pioneers, David Sayre, to ask if the same technique used for crystallography could be applied to non-periodic objects (Sayre 1952). The possibility to realize such idea would led to the imaging of matter at a resolution which is limited by the angle of diffraction, throwing away all the issues concerning lenses (mainly lens aberrations). The idea of Sayre was actually a relatively trivial one. In fact, crystallography requires the measurement of all the diffraction spots produced by a crystal. Sayre argued that, if one is able to measure the diffraction at, at least, twice that sampling rate, this would provide the necessary information for the reconstruction of the whole object. This paper by Sayre was just an hint, and he didn't go inside the mathematical issues of his proposal.

A first step after Sayre work was made when scientists started to develop imaging methods in the so-called *water window*, a spectral region in the soft X-rays domain where there are appreciable differences in absorption between carbon and oxygen atoms, allowing the imaging of biological samples in their natural environment. In this framework, an algorithm to treat diffraction data was proposed by Gerchberg and Saxton (Gerchberg & Saxton 1972). This approach, directly inherited from electron microscopy, is able to retrieve a phase distribution of a sample, given its diffraction pattern in far field and its intensity distribution in direct space. This algorithm converges to a solution, but the strong limitation of this approach derives from the need of an intensity distribution in the real space. The latter is available only when other direct imaging techniques are possible, which is not the case for small objects.

In 1982, Bates suggested that a sample density distribution is uniquely determined by its autocorrelation function, which actually is the inverse Fourier Transform of the diffraction pattern, if its spatial extension is known (the so-called support function). Bates also pointed out the aspects that cannot be recovered, i.e. some situations where the diffraction pattern is the same (i.e. have the same modulus of the Fourier Transform): among them, a translation of the sample or a global phase shift. All of these ambiguities, that will be discussed later on, are actually not important: the absolute phase, for example, has no significant physical meaning. Concerning the uniqueness of the solution, apart some peculiar cases (Fienup 1982), Bates also proved that it is very unlikely that two objects with the same support function produce the same diffraction pattern.

Fienup proposed a practical algorithm (Fienup 1978) to exploit the support function as constraint to get a unique solution, by modifying the Gerchberg and Saxton approach (Gerchberg & Saxton 1972). If, from the theoretical side, the Fienup's approach is capable to reach convergence, some issues emerged from the experimental side. The first point

regards the amount of the scattered radiation. In fact, when using X-rays, the scattered photons are much less than the unscattered ones. This means that the transmitted beam, once recorded by the detector, will hide the weak scattered radiation. An experimental solution to the problem is to insert the so-called *beam-stopper*, that hides the central part of the detector from the unscattered beam. If this trick mitigates the described effect, it introduces an issue concerning the data analysis, because the central part of the diffraction pattern, which corresponds to the low frequencies of the power spectrum, is missing.

A first attempt to overcome the issue was to extrapolate those low-resolution information from a low-resolution image of the sample, as done by Miao et al. (1999). Their trick was to substitute the missing data with the Fourier Transform of an image of the same sample acquired via a Transmission Electron Microscope. This work, which experimentally demonstrated the feasibility of the approach, can be considered the birth of Coherent Diffraction Imaging (CDI). From that moment on, the interest of the scientific community towards CDI rapidly grew. Even if successful, Miao's result was however affected by the need of spatial information on the sample, in order to get a complete diffraction pattern.

A second boost was given to CDI by the development of X-rays Free Electron Lasers (XFELs). Their goal is to provide an amount of photons able to give a usable diffraction pattern for the imaging of molecules at atomic resolution. XFELs produce a radiation with a brilliance (merely speaking, the number of photons) which is billion of times the one provided by synchrotrons. In the latter, the radiation is provided by electrons circulating in a ring, and the beam is available only in defined sections of the facility. Instead, in a XFEL, bunches of electrons are linearly accelerated and pass through a set of magnetic ondulators. The bunches size is, in the first part, greater than the produced radiation wavelength, thus the radiation emitted by electrons is uncoherent, due to the fact that there is no correlation among the spatial position of those electrons. If ondulators are properly tuned with respect to the radiation wavelength, the emitted radiation interacts with the electron bunches, causing the so-called *microbunching*: the electrons organize themselves such that they progressively emit coherent radiation. Those "synchronized" electrons produce a radiation which is exponentially amplified in intensity, up to a saturation level. This peculiar process is called Self-Amplified Spontaneous Emission (SASE) (Bonifacio et al. 1984) and the recent implementations provide short flashes (in the order of tens of femtoseconds), high peak intensity and full transverse coherence.

The main issue concerning XFELs is that the amount of radiation is such that every sample hit by the beam is doomed to be totally destroyed. However, Chapman et al. (2006) experimentally demonstrated the so-called *diffract and destroy* approach, which means that the scattered radiation useful for CDI contains the information of the intact sample, i.e. the destruction of the sample happens in a time scale which is much lager than the flash duration. The very short pulses opens to possibilities to perform also time resolved imaging via a pump and probe experiment, firstly experimentally demonstrated by Barty et al. (2008). This approach makes use of a second laser source, usually in the optical or ultraviolet spectrum, and characterizes the sample properties as function of the time delay between the *pump* beam and the XFEL *probe*.

However, the true reason for why XFELs represent a revolution in the physics of matter is the possibility to perform single molecule imaging. This "dream", however, is hampered by some technical problems, and the main one still remains the low amount of scattered light. To overcome this issue, and to perform 3D imaging, many diffraction patterns from the same specimen must be acquired. As long as we deal with XFELs, speaking about the "same specimen" means speaking about different samples of the same nature: as told before, the radiation energy is so high that the sample is damaged. Thus,

in this situation, the only way to get sufficient diffraction data to perform a reliable 3D CDI is to acquire many diffraction patterns on similar samples and then recombine them with suited algorithms which are able to perform a "smart" averaging of the diffraction data, considering that it is acquired at random orientations (Bogan et al. 2010; Gaffney & Chapman 2007; Ekeberg et al. 2015).

A recent variant of CDI is called Electron Diffraction Imaging (EDI) (Zuo et al. 2003; Huang et al. 2009; De Caro et al. 2010), which exploits coherent electron beams instead of photons, performed in Transmission Electron Microscopes. EDI experiments are able to provide quantitative maps of the atomic potential down to a sub-angstrom resolution(De Caro et al. 2012, 2013).

The very last part of the CDI data analysis chain, after the experiment setup, the data acquisition and the data restoration, is the data inversion step, on which all the success of a CDI experiment depends. The data inversion step consists in recovering the density distribution of the sample of interest from the diffraction data, which merely corresponds to the square modulus of the Fourier Transform of the density. Some mathematical aspects of the problem were previously mentioned, while a deeper view will be provided in the next sections.

1.2 The *phase problem*

Before venturing into a mathematical description of the topic, it's worth providing its definition in the most concise and clear way possible. Following Fienup's words of 1975 (Fienup 1975), solving the *phase retrieval problem* means *"reconstructing a general object from the modulus of its Fourier transform"*. A direct translation of this sentence in mathematical speech could be:

$$\text{Find } \rho(\vec{x}) : |\mathscr{F}[\rho(\vec{x})](\vec{q})| = M(\vec{q}) \tag{1.1}$$

where M is the (known) modulus, $\mathscr{F}$ stands for *the Fourier Transform of* and $\rho(\vec{x})$ is the density distribution of the *general object*, i.e. a function which gives a single value for any given position $\vec{x}$ in the space. Finding $\rho(\vec{x})$ is equivalent to retrieving the correct set of phases $\phi(\vec{q})$, such that $\rho(\vec{x}) = \mathscr{F}^{-1}[M(\vec{q})e^{\iota\phi(\vec{q})}](\vec{x})$, and for this reason the *phase retrieval problem* assumes its name. From now on, the greek letter ι (iota) will be used to identify the imaginary unit, i.e. $\iota = \sqrt{-1}$, while the letter i has to be interpreted as an index.

Given $\rho(\vec{x})$, its Fourier Transform is defined as:

$$\tilde{\rho}(\vec{q}) = \mathscr{F}[\rho(\vec{x})](\vec{q}) = \int_{-\infty}^{-\infty} \rho(\vec{x}) e^{\iota 2\pi \vec{q}\cdot\vec{x}} d\vec{x}. \tag{1.2}$$

where $\vec{x}$ and $\vec{q}$ are spatial coordinates in the direct and in the Fourier space respectively.

In practice, $\rho(\vec{x})$ and $\tilde{\rho}(\vec{q})$ are no more functions, but D-dimensional matrices with N^D entries. Thus, the Fourier Transform assumes the discrete form

$$\tilde{\rho}(\vec{q}) = \mathscr{F}[\rho(\vec{x})](\vec{q}) = \sum_{\vec{x}=0}^{N-1} \rho(\vec{x}) e^{\iota 2\pi \vec{q}\cdot\frac{\vec{x}}{N}} \tag{1.3}$$

where $\vec{x}$ and $\vec{q}$ are discrete coordinates identifying pixels, that range from 0 to $N-1$ in each of the D dimensions. For example, in the 2D case the Discrete Fourier Transform

Figure 1.1: On the left, the real space image of a matrix obtained by combining the amplitudes of a picture representing the number 3 with the phases of a picture representing the number 1. The same thing have been performed on the right, by using the phases extracted from a picture representing the number 2. Even if both images have the same Fourier amplitudes (the ones of the number 3) they appear much more similar to the information stored in their Fourier phases.

can be written as:

$$\tilde{\rho}_{k,l} = \mathscr{F}[\rho]_{k,l} = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} \rho_{i,j} e^{\iota \frac{2\pi}{N}(ki+lj)} \tag{1.4}$$

From this moment on, $\mathscr{F}$ will be intended in its discrete form, ρ will be considered a D-dimensional matrix with N entries for each dimension, while vectors, denoted by the symbol $\vec{\cdot}$, have to be interpreted as D-dimensional arrays containing pixel coordinates.

At this point, despite the problem may appear clear, an issue arises: Fig.1.1 depicts two images of two different numbers, 1 and 2. These figures have the same Fourier modulus, and this amplitude is extracted from a picture representing the number 3. The first information provided by this trivial example is that phases carry many information (surely more than the amplitudes). The second implication is that the problem, as posed in Eq.(1.1) is undefined: given a Fourier amplitude, there is an infinite set of solutions to Eq.(1.1). Let's call $\mathcal{M}$ the set of these solutions, that is:

$$\mathcal{M} = \{\rho(\vec{x}) : |\mathscr{F}[\rho(\vec{x})](\vec{q})| = M(\vec{q})\} \quad . \tag{1.5}$$

This fact becomes clearer if Eq.(1.1) is explicitly rewritten in the discrete form, that is:

$$\left| \sum_{\vec{x}=0}^{N-1} \rho(\vec{x}) e^{\iota 2\pi \vec{q}\cdot\frac{\vec{x}}{N}} \right| = M(\vec{q}) \quad . \tag{1.6}$$

This equation is, actually, a system of N^D equations and the aim is to solve this system for each pixel of $\rho(\vec{x})$. In the case of complex valued $\rho(\vec{x})$, $2N^D$ values are unknown with only N^D equations (the dimension of the matrix M). If, instead, $\Im[\rho(\vec{x})] = 0$ (where $\Im$ represents the imaginary part), $M(\vec{q}) = M(-\vec{q})$ and the number of independent equations drops down to $\frac{N^D}{2}$. Both in the complex and in the real case, the unknowns are twice the

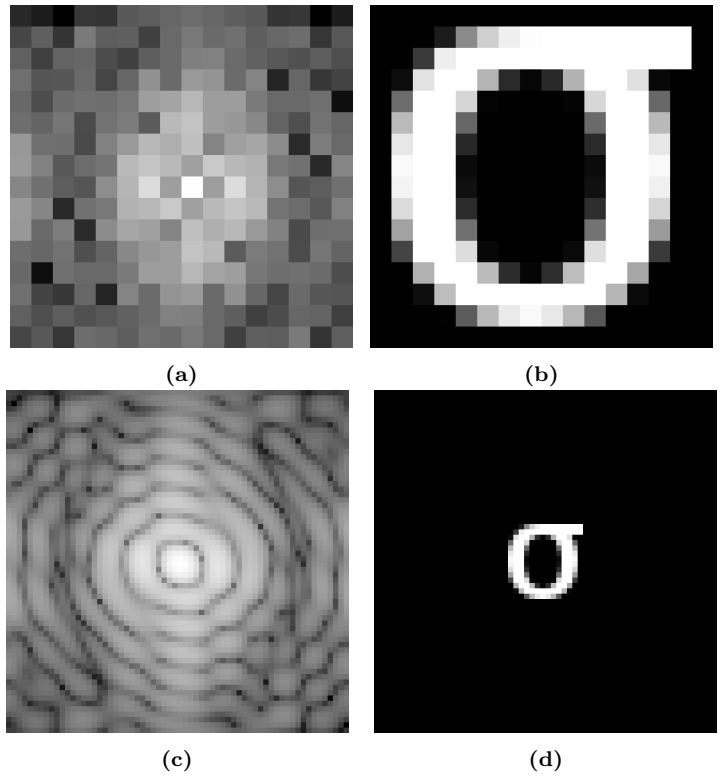


Figure 1.2: Example concerning the oversampling degree. In (a), a 16x16 pixels (simulated) diffraction pattern, which is the amplitude of the Fourier Transform of (b): the real space image occupies almost the whole 16x16 matrix, such that the value of σ in Eq.(1.7) is particularly low. When amplitudes are sampled in a finer way, let's say 4 times finer as depicted in (c), the resulting real space sample (d) preserves its resolution (i.e. its extension in pixels) but its extension with respect to the whole matrix is just a small fraction of the total number of pixels. In this way σ increases its value and the number of known pixels (i.e. the entries of the matrix defining the diffraction pattern) is much higher than the unknowns (i.e. the real space area where the sample assumes values different from 0).

number of equation, leaving the problem undetermined. The indeterminateness of the problem holds as long as the number of known values (N^D) is less than twice the number of unknown-valued pixel, that is the ratio

$$\sigma = \frac{N^D}{\text{number of unknown pixels}} \tag{1.7}$$

is less than 2 (Miao et al. 1998).

Even if $\mathcal{M}$ cannot represent the solution to the *phase problem*, it can be said with certainty that this solution belongs to $\mathcal{M}$, i.e. $\rho(\vec{x})_{\text{sol}} \in \mathcal{M}$. Further constraints are thus needed to uniquely identify $\rho(\vec{x})_{\text{sol}}$, or, in other words, further equations must be added to (1.6).

The usual way to increase the σ value is to add information that increase the value of Eq.(1.7) via the so-called *oversampling* method (Bates 1982), which aims to increase the number of equations keeping the number of unknowns constant. Let's imagine to

perform a (really trivial) measurement of a 2D Fourier amplitude $M(\vec{q})$ which gives as result a 16x16 pixels matrix depicted in Fig.1.2a. This amplitude is the one belonging to Fig.1.2b. If the experiment is repeated, by measuring the amplitudes four times finer (i.e. a matrix of 64x64 pixels) depicted in Fig.1.2c, those amplitudes refer to a 64x64 pixels image in the real space where, actually, the total number of unknowns is the same of the 16x16 pixels case (Fig.1.2d). Pixel outside the central 16x16 pixels square of Fig.1.2d are known to be equal to zero.

At this point, it's useful to introduce the so-called *support function* $S(\vec{x})$ which assumes 0 values where $\rho(\vec{x})$ is known to be 0, 1 otherwise. In the case of Fig.1.2d it could be, for example, a function which assumes the value 1 in the central 16x16 pixels square. The support function allows to add further equations to the system defined in Eq.1.6, in particular

$$S(\vec{x}) \cdot \rho(\vec{x}) = \rho(\vec{x}). \tag{1.8}$$

This system provides a set of equations that drops the number of unknown pixels down to $A \cdot N^D$, where $A \in [0, 1]$ is the number of pixels belonging to the support normalized with the total number of pixels in the matrix. Substituting this value inside Eq.1.7 it turns out that $\sigma = A^{-1}$. Thus, the *phase retrieval problem* becomes determined ($\sigma > 2$) when the area defined by the support function is less than the half of the total area. Finding out the correct support function isn't a trivial task at all and it will be topic of discussion in the next sections.

In order to easily visualize the problem, it's useful to introduce a set $\mathcal{S}$ as the one of all the functions $\rho(\vec{x})$ satisfying Eq.1.8. As like as we did for the set $\mathcal{M}$, we now define

$$\mathcal{S} = \{\rho(\vec{x}) : S(\vec{x}) \cdot \rho(\vec{x}) = \rho(\vec{x})\}. \tag{1.9}$$

Ensuring that *the phase retrieval problem* is determined, i.e. it has a unique solution, is equivalent to assert that the intersection between $\mathcal{M}$ and $\mathcal{S}$ is unique, and, if it exists, it is the solution:

$$\mathcal{M} \cap \mathcal{S} = \{\rho_{\text{sol}}(\vec{x})\} \tag{1.10}$$

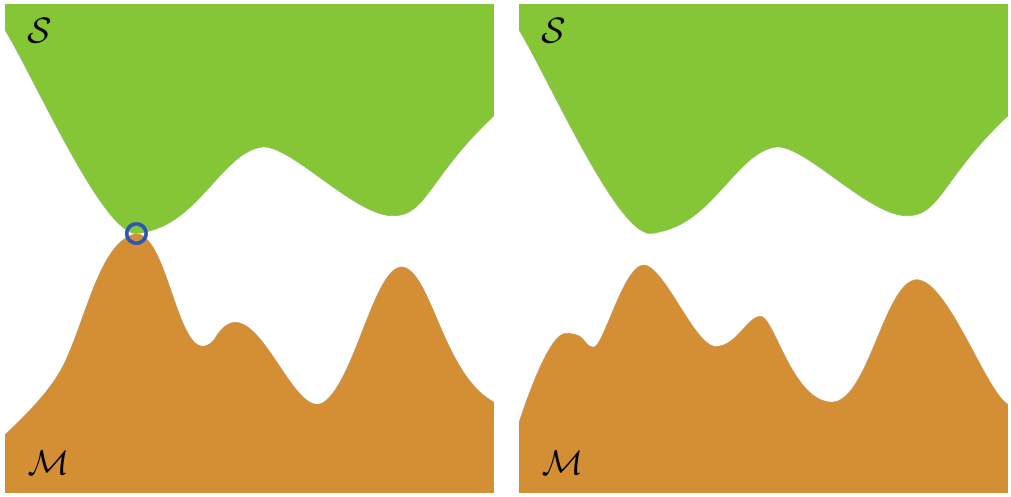
Even if it could appear the end of the story, here is a list of facts that may tickle the reader:

1. $\mathcal{M} \cap \mathcal{S} = \varnothing$ almost always, i.e. the solution defined by (1.10) almost never exists.
2. if a solution exists, an infinite number of equivalent solutions exists.
3. assuming that the solution exists, is it possible to reach it? If yes, how?

Next sections will be dedicated to (try to) answer these questions.

1.3 The non-trivial definition of the solution

The sets $\mathcal{M}$ and $\mathcal{S}$ represent the two constraints that a solution should satisfy. A visual description is given by Fig.1.3a, where the two sets intersect in a point (the solution) highlighted with a blue circle. The first issue about this description arises when we consider the noise on the measured amplitudes M, which derives from the measured intensities intrinsecally following a Poisson distribution. This noise, that is intrinsic of the measurement process and, thus, unavoidable, recasts the shape of the set $\mathcal{M}$, as exemplified by Fig.1.3b. It turns out that the intersection $\mathcal{M} \cap \mathcal{S}$ gives an empty set, and the definition of what a solution is, described by Eq.1.10, holds no more.



(a) The ideal situation **(b)** The noisy case

Figure 1.3: Exemplification of the constraints relationship in the ideal case (a) when the diffraction pattern is noiseless and artifacts-free, and in the real case (b), when noise and artifacts modify the shape of the set $\mathcal{M}$.

A redefinition of the term *solution* urges, and its new form now takes into account the distance among those two sets. The following description follows the one proposed by Marchesini (2007).

First of all, we need to define two operators that act on a given guess $\rho(\vec{x})$:

$$P_{\mathcal{M}}\rho(\vec{x}) = \mathscr{F}^{-1}[M(\vec{q})e^{\iota \arg[(\tilde{\rho}(\vec{q})]}] \tag{1.11}$$

$$P_{\mathcal{S}}\rho(\vec{x}) = S(\vec{x}) \cdot \rho(\vec{x}) \tag{1.12}$$

It's trivial to show that these operators are *projectors*, by simply noting that

$$P_{\mathcal{M}}P_{\mathcal{M}}\rho(\vec{x}) = P_{\mathcal{M}}\rho(\vec{x}) \tag{1.13}$$

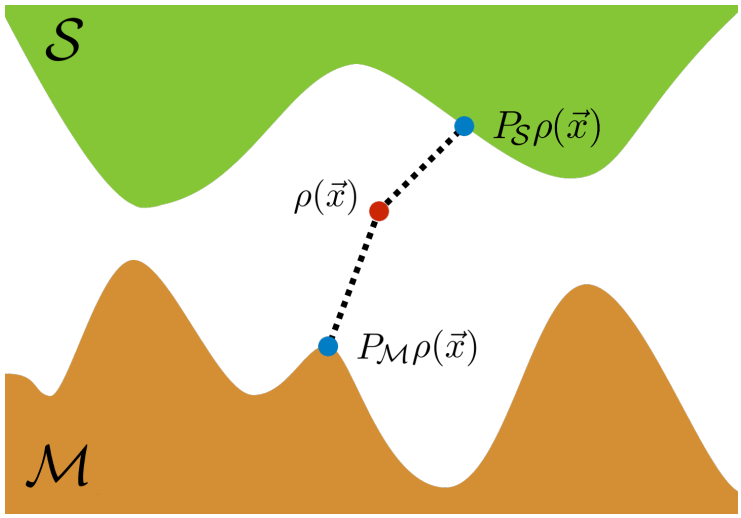
$$P_{\mathcal{S}}P_{\mathcal{S}}\rho(\vec{x}) = P_{\mathcal{S}}\rho(\vec{x}) \quad . \tag{1.14}$$

The first one, $P_{\mathcal{M}}$, projects $\rho(\vec{x})$ on the set $\mathcal{M}$, by ensuring that its Fourier amplitudes are the measured ones. The second, $P_{\mathcal{S}}$, projects $\rho(\vec{x})$ on the set $\mathcal{S}$, by ensuring that the pixels of $\rho(\vec{x})$ are null where $S(\vec{x}) = 0$. Their actions are exemplified by Fig.1.4. As it can be easily noted, when the guess $\rho(\vec{x})$ is far from the solution, its projections on the two sets, i.e. $P_{\mathcal{M}}$ and $P_{\mathcal{S}}$, are very far from each other. It turns out that, given that $\mathcal{M} \cap \mathcal{S} = \varnothing$, a new definition of solution could be based on the information about the *distance* between the two sets. Following this idea, first of all a definition of *distance* is needed. Many kind of *distances* can be proposed, but two of them have peculiar importance.

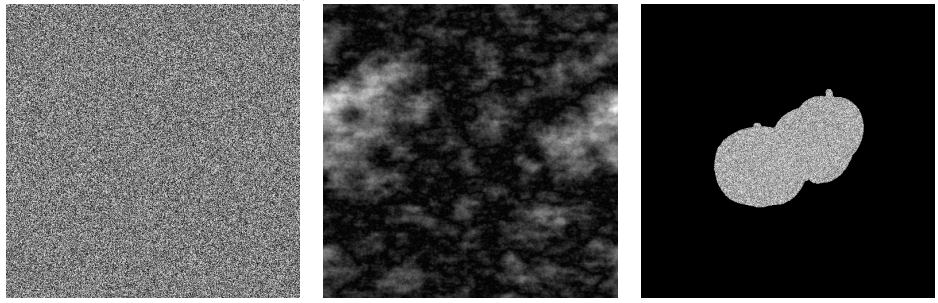
The first proposed *distance* between two configurations $\rho^A(\vec{x})$ and $\rho^B(\vec{x})$ is defined as follows:

$$D_{\mathcal{M}}[\rho^A(\vec{x}), \rho^B(\vec{x})] = \sum_{\vec{q}} | \parallel \tilde{\rho}^A(\vec{q}) \parallel - \parallel \tilde{\rho}^B(\vec{q}) \parallel |^2 \quad . \tag{1.15}$$

$D_{\mathcal{M}}$ has the peculiarity to assign a null distance to all the elements that have the same amplitude of their FT: in particular, it means that every couple $\rho^A(\vec{x})$ and $\rho^B(\vec{x})$ belonging to $\mathcal{M}$ have $D_{\mathcal{M}} = 0$.



(a) Visual representation of projectors



(b) $\rho(\vec{x})$ **(c)** $P_{\mathcal{M}}\rho(\vec{x})$ **(d)** $P_{\mathcal{S}}\rho(\vec{x})$

Figure 1.4: Visual representation of the action of the projectors (a) and their effect on a real image (c,d).

The second proposed *distance* is

$$D_{\mathcal{S}}[\rho^A(\vec{x}), \rho^B(\vec{x})] = \sum_{\vec{x}} [1 - S(\vec{x})] \cdot |\rho^A(\vec{x}) - \rho^B(\vec{x})| \quad . \tag{1.16}$$

$D_{\mathcal{S}}$ assigns a distance equal to 0 to all the couples $\rho^A(\vec{x})$ and $\rho^B(\vec{x})$ that has the same density distribution outside the support area. It's worth noting that all the elements belonging to $\mathcal{S}$ have a null distance each other.

Given a guess $\rho(\vec{x})$, it's now clear that $D_{\mathcal{M}}[\rho(\vec{x}), P_{\mathcal{M}}\rho(\vec{x})]$ can be interpreted as the distance between $\rho(\vec{x})$ and the set $\mathcal{M}$, while $D_{\mathcal{S}}[\rho(\vec{x}), P_{\mathcal{S}}\rho(\vec{x})]$ as its distance from the set $\mathcal{S}$. Moreover, the distances $D_{\mathcal{M}}[P_{\mathcal{S}}\rho(\vec{x}), P_{\mathcal{M}}\rho(\vec{x})]$ and $D_{\mathcal{S}}[P_{\mathcal{S}}\rho(\vec{x}), P_{\mathcal{M}}\rho(\vec{x})]$ can be now interpreted as two different expressions for the distance between the two sets.

Given a solution guess $\rho(\vec{x})$ we can now assign an *error* value, which represents how much are $\rho(\vec{x})$ projections on the two sets far from each other. The error value $E_{\mathcal{M}}$, concerning the amplitude constraint $\mathcal{M}$, is given by merging Eq.(1.11), Eq.(1.12) and

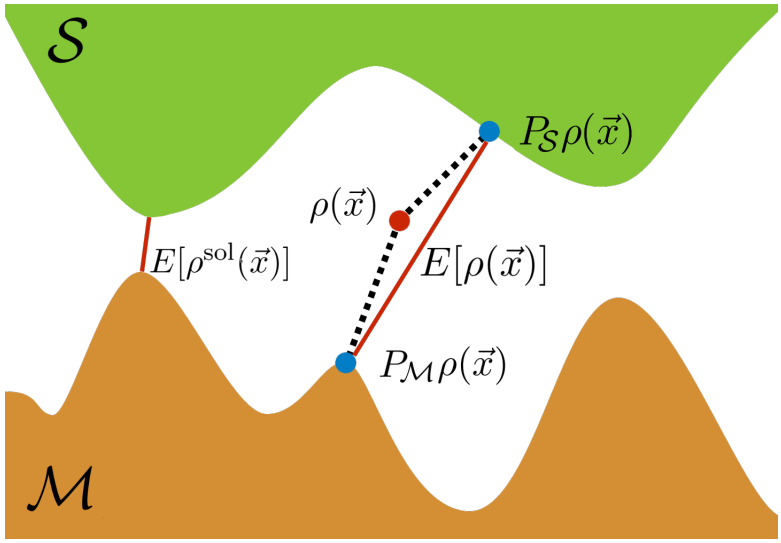


Figure 1.5: Visual exemplification of the meaning of the error functionals in Eq.1.17 and Eq.1.18.

Eq.(1.15) as follows:

$$E_{\mathcal{M}}[\rho(\vec{x})] = D_{\mathcal{M}}[P_{\mathcal{S}}\rho(\vec{x}), P_{\mathcal{M}}\rho(\vec{x})] = \sum_{\vec{q}} | \parallel \mathscr{F}[S(\vec{x}) \cdot \rho(\vec{x})](\vec{q}) \parallel - M(\vec{q}) \parallel |^2 \quad . \tag{1.17}$$

On the other side, the error value $E_{\mathcal{S}}$ is provided by the merging of Eq.(1.11), Eq.(1.12) and Eq.(1.16):

$$E_{\mathcal{S}}[\rho(\vec{x})] = D_{\mathcal{S}}[P_{\mathcal{S}}\rho(\vec{x}), P_{\mathcal{M}}\rho(\vec{x})] = \sum_{\vec{x}} [1 - S(\vec{x})] \cdot |\mathscr{F}^{-1}[M e^{\iota \arg[\tilde{\rho}(\vec{q})]}](\vec{x})| \quad . \tag{1.18}$$

Eq.(1.17) and Eq.(1.18) provide two different forms for the *error functional*. Given a guess $\rho(\vec{x})$, its error is the distance between its projections on $\mathcal{M}$ and $\mathcal{S}$. $E_{\mathcal{M}}$ computes the distance based on the metric defined by (1.15), while $E_{\mathcal{S}}$ exploits the one in Eq.(1.16). Fig.1.5 depicts the geometric meaning of the error value, where $E[\rho(\vec{x})]$ refers to the error of a given guess $\rho(\vec{x})$, while the error of the solution $E[\rho^{\text{sol}}(\vec{x})]$ is the one that minimizes the distance between the two sets.

In the *ideal case* depicted in Fig.1.3a, where $\mathcal{M}$ and $\mathcal{S}$ intersects, they are equivalent, because the solution requires that $E_{\mathcal{M}} = E_{\mathcal{S}} = 0$. In the real case (Fig.1.3b) this situation exists no more, and the solution is defined as the one that minimizes the *error functional* $E[\rho]$. This new definition of solution can be stated as follows:

$$\rho^{\text{sol}} := \left\{ \hat{\rho} : E[\hat{\rho}] = \min_{\rho} E[\rho] \right\} \tag{1.19}$$

where $E[\rho]$ can be equal to Eq.(1.17) or Eq.(1.18). A colloquial translation of this definition can be: "the solution is a function such that the distance beween its projection on the two constraints has the minimum value".

This sentence turns *the phase retrieval problem* into an *optimization problem*. The next section will investigate how hard this problem is.

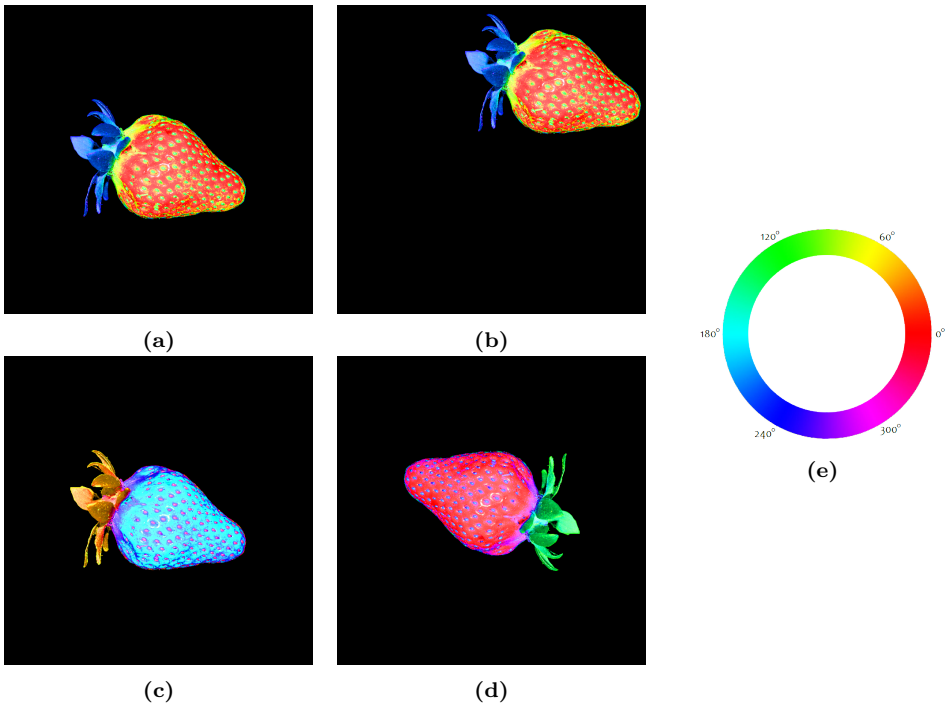


Figure 1.6: Ambiguities in the phase retrieval problem. In particular, (b) is a translation of (a), (c) is a global shift in the phase and (d) is obtained by inverting the coordinates and the imaginary part (complex conjugation). These images are created by assigning the amplitude of the density $|\rho(\vec{x})|$ to the lightness and the phase $\arg[\rho(\vec{x})]$ to the hue, following the color map depicted in (e).

A last digression in this section is dedicated to the analysis of some ambiguities in the phase retrieval problem, as mentioned in the introduction of this chapter. In Fig.1.6, four different $\rho(\vec{x})$ are depicted. Even if they looks different, they actually have the same FT amplitude, such that if one of them is a solution to the phase problem, all of them are a correct solutions. In fact, the equality $|\tilde{\rho}(\vec{k})| = |\tilde{\sigma}(\vec{k})|$ holds for the following definitions of σ:

- **Translation:** $\sigma(\vec{x}) = \rho(\vec{x} + \vec{x}_0)$, where $\vec{x}_0$ is a constant
- **Phase shift:** $\sigma(\vec{x}) = \rho(\vec{x})e^{\iota\phi_0}$, where ϕ_0 is a constant
- **Coordinate inversion:** $\sigma(\vec{x}) = \rho^*(-\vec{x})$, where * denotes the complex conjugate.

Moreover, the equality holds also for a combination of these transformations (Bates 1982). This fact means that, if a solution of the phase retrieval problem exists for a given diffraction data, actually there are an infinity of solutions. These equivalent solutions are called *trivial characteristics* of $\rho(\vec{x})$ (Miao et al. 1998). They don't represent a big issue in the identification of the solution, because once that a *trivial characteristic* is found, the solution is reached. On the other side, these ambiguities often cause stagnation issues of phase retrieval algorithms, as well shown by Fienup & Wackerman (1986).

1.4 The optimization problem

Coming back to the introduction, solving the *phase problem* was compared to a blind golfer trying to hit the hole in the golf course, with the only knowledge of its altitude and that the hole resides in the lowest point of the lawn. It's now clear why this comparison well describes our problem: the phase retriever has to find the solution, and he is able to measure only the error of a given solution guess.

The main issue concerns the ignorance about the solution error, such that the golfer should visit all the altitude minima of the golf course before addressing one of them as "the lowest one".

A second issue concerns the dimensionality of the problem, such that this golfer moves in an N-dimensional space (instead of a common 2D one), where N commonly runs from 10^5 to 10^8 for CDI.

A third issue is that the golfer requires a long time both to move and to evaluate its altitude: in fact the error evaluation, defined in the previous section, involves the computation of a Discrete Fourier Transform (DFT).

We have previously mentioned that all the mathematical formulas have to be intended in a discrete space. This means that, first of all, the experimental amplitudes $M(\vec{q})$ is actually a D-dimensional matrix M. For sake of simplicity, formulas will be presented for the bi-dimensional case ($D = 2$): the extension to the 3-dimensional case is trivial. Thus, for $D = 2$, amplitudes are nothing more than a real-valued matrix $M_{i,j}$ with $i, j = 0, ..., N-1$. $M_{i,j}$ represents the first constraint to the problem. The second constraint is given by the support function, which is, again, a $N \times N$ matrix $S_{i,j}$ of boolean values. Finally, the density $\rho(\vec{x})$ is a $N \times N$ matrix $\rho_{i,j}$ of (usually) complex values.

Again, Eq.(1.19) holds, but some translations in a discrete form are needed. The first one concerns the error value. In particular, Eq.(1.17) in its discrete form follows:

$$E_{\mathcal{M}}[\rho] = \sum_{k,l=0}^{N-1} | \parallel \mathsf{DFT}[S \circ \rho]_{k,l} \parallel - M_{k,l}|^2 \tag{1.20}$$

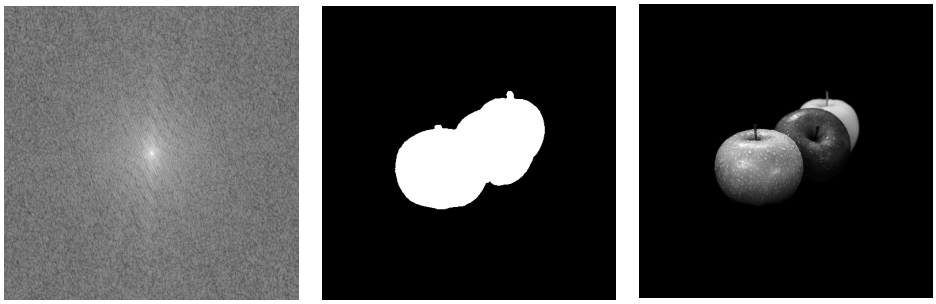
where DFT explicitly refers to the discrete form of the Fourier Transform. The operator $\circ$ stands for the *Hadamard product* between two matrices, where $(A \circ B)_{i,j} = A_{i,j} \cdot B_{i,j}$. Defining the solution as the one that minimizes Eq.(1.20) means that ρ^{sol} is, among the densities that resides inside the support area, the one that better fits the experimental amplitudes M. An example of the two constraints, M and S, and the solution ρ^{sol} is provided by Fig.1.7.

Under the discrete point of view, it's clear that the unknowns to be retrieved are merely the pixels with coordinates i, j such that $S_{i,j} = 1$. Those pixels represent the free parameters of the *phase retrieval* optimization problem. The greater the number of those pixels, the heavier the optimization problem. Their quantity can be easily estimated by computing $\sum_{i,j}^{N-1} S_{i,j}$. In this framework, it's worth defining a quantity O_d, called *oversampling degree*, that is strictly bound to σ, declared in Eq.(1.7):

$$O_d = \frac{\sigma}{2} = \frac{N^D}{2\sum_{i_1,...,i_D}^{N} S_{i_1,...,i_D}} = (2A)^{-1} \tag{1.21}$$

where A is the support extension as fraction of the total number of pixels in the matrix, i.e.:

$$A = \frac{\sum_{i_1,...,i_D}^{N} S_{i_1,...,i_D}}{N^D} \overset{D=2}{=} \frac{\sum_{i,j}^{N} S_{i,j}}{N^2}. \tag{1.22}$$



(a) $M_{i,j}$ **(b)** $S_{i,j}$ **(c)** $\rho^{\mathrm{sol}}_{i,j}$

Figure 1.7: Example of matrices for a 512x512 pixels bi-dimensional case. In (a), the simulated diffraction pattern (in logarithmic scale), obtained by the Fourier Transform of (c); in (b), an example of support function for this data.

O_d has to be greater than 1 to allow the existence of a unique solution.

For 2-dimensional CDI, the experiment is usually set up such that $O_d \gtrsim 5$. This condition means that the total number of unknowns is about one order of magnitude less than the dimensions of the acquired diffraction pattern $I_{i,j} = M^2_{i,j}$. Common sizes for experimental data (merely the detector pixels) are about the megapixel, i.e. $N \approx 10^3$ and $N \times N \approx 10^6$ values. Thus, pixels that have to be retrieved ($\sum_{i,j}^{N} S_{i,j}$) are around 10^5.

For what concerns 3-dimensional CDI, a much higher value for O_d is required, which is commonly one order of magnitude greater than the $D = 2$ case. The reason for this distinction will be clarified in the next sections, and mainly derives from the high amount of unknown values in the measured amplitudes $M_{i,j,k}$. In this case, experimental data has sizes up to $N \times N \times N = 10^9$, such that the number of unknowns, i.e. the number of free parameters involved in the optimization procedure, goes up to 10^7.

1.5 Short history of phase retrieval algorithms

Coming back again to the golfer metaphor, we have seen so far why the hole stays in the lower point of the field and why the golfer is blind and knows only his altitude. In this section we are going to face the strategies the golfer can adopt to find the hole, which influence the way he explores the golf course starting from an initial position.

As stated in the introduction, the very first approaches to the *phase retrieval problem* for CDI were proposed by Fienup (1978). They can be described in different ways: the one I chose for this work is probably the least widespread in literature and it is based on the projections operators defined in Eq.(1.11) and Eq.(1.12) (Marchesini 2007). It may be useful to recall and rewrite them in terms of matrices:

$$P_{\mathcal{M}}\rho = \mathsf{DFT}^{-1}[M \circ e^{\iota \arg[(\tilde{\rho}]}] \tag{1.23}$$

$$P_{\mathcal{S}}\rho = S \circ \rho \tag{1.24}$$

where $\tilde{\rho} = \mathsf{DFT}[\rho]$.

$P_{\mathcal{M}}$ acts on a matrix ρ by going in the reciprocal space, imposing that the amplitudes are the experimental ones (without changing the phases) and coming back to the real space. It is a projector just because applying $P_{\mathcal{M}}$ many times has the same effect as

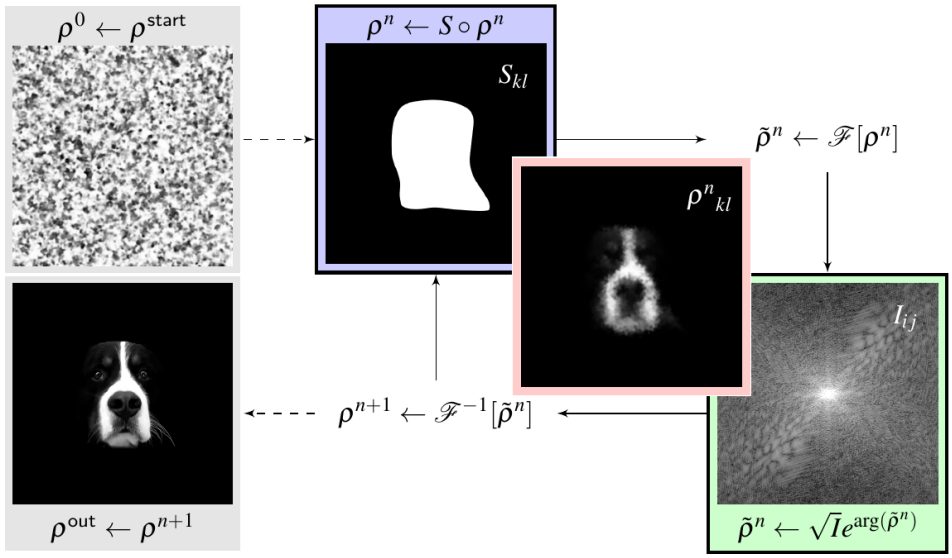


Figure 1.8: Description of the Error Reduction algorithm. An initial, usually random, guess ρ^{start} is provided to the algorithm. After that, the algorithm goes back and forth from the real space to the reciprocal one and cyclically imposes the two constraints via the projector operators. The procedure stops when the desired error is reached or when the desired number of iterations have been performed, providing ρ^{out} as result.

applying it once. The same consideration holds for $P_{\mathcal{S}}$, which acts in the direct space by suppressing the pixels of ρ that lay where $S_{i,j} = 0$. It's worth introducing also other two operators, based on these projectors, that will result useful for the description of the algorithms:

$$R_{\mathcal{M}}\rho = (2P_{\mathcal{M}} - \mathbf{I})\rho \tag{1.25}$$

$$R_{\mathcal{S}}\rho = (2P_{\mathcal{S}} - \mathbf{I})\rho \tag{1.26}$$

where $\mathbf{I}$ is the identity operator.

If $P_{\mathcal{M}}$ and $P_{\mathcal{S}}$ are projectors on the sets $\mathcal{M}$ and $\mathcal{S}$ respectively, the latter are reflectors. In fact, it can be easily verified that $R^2 = (2P - \mathbf{I})^2 = 4P^2 + \mathbf{I} - 4P = 4P + \mathbf{I} - 4P = \mathbf{I}$.

Our digression about phase retrieval algorithms starts with the conceptually simplest one, the Error Reduction (ER) algorithm. It is based on the simple idea of recursively jumping back and forth from the set $\mathcal{M}$ to the set $\mathcal{S}$, described by Fig.1.8.

Starting from an initial guess ρ^0, the support constraint and the amplitude constraint are iteratively imposed: the guess ρ is refined step by step towards the solution.

A step of ER algorithm is trivially described by the following relation:

$$\rho^{n+1} = P_{\mathcal{M}}P_{\mathcal{S}}\rho^n \tag{1.27}$$

Thanks to this description, its way of acting can be depicted in terms of sets, as shown in Fig.1.9.

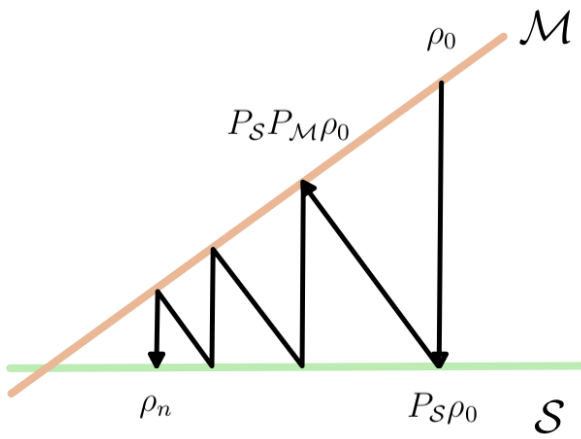


Figure 1.9: Geometric representation of the Error Reduction algorithm.

As the algorithm moves back and forth from a point in a set to the nearest point belonging to the other set, the following relationship turns out to be easy to understand:

$$E[\rho^{n+1}] \leqslant E[\rho^n] \tag{1.28}$$

where E is the error functional of Eq.(1.20). If, at a first glance, Eq.(1.28) may appear as a good news, actually it represents the main limitation of this approach. Let's consider again Fig.1.9, and place the golfer at position labeled with ρ_0. Let's consider $\mathcal{M}$ the golf course, such that its height corresponds to the distance with the set $\mathcal{S}$. Let's also consider the different steps of the algorithm belonging to $\mathcal{M}$ as the subsequent positions assumed by the golf ball after being thrown by the golfer from ρ_0. It is evident that if there was a local minimum of the distance of the two sets between ρ_0 and the solution, the ball would remain stuck there. If we carry this example from the 2 parameters case of Fig.1.9 to the CDI situation of more than 10^5 parameters, and if we try to imagine how many local minima there could be between ρ_0 and the solution, it appears clear that the Error Reduction algorithm alone is doomed to fail its mission. ER, however, has a really *stable* behavior. This means that, if it approaches a minimum, it is able to precisely identify it. Moreover, its *stability* ensures that it correctly identifies the solution once that its convergence basin is reached.

With the aim to overcome these limitations, along with the ER approach, Fienup proposed the Hybrid Input Output algorithm (HIO) (Fienup 1978). HIO keeps the same scheme of ER, but it introduces a novelty with respect to how the support constraint is applied. This method takes into account the error of the current estimation of ρ thanks to a *feedback parameter* β. Following Fienup's original description, its action is described by the following relation:

$$\rho^{n+1} = \begin{cases} P_{\mathcal{M}}\rho^n, & \text{if } S_{i,j} = 1 \\ (1 - \beta P_{\mathcal{M}})\rho^n, & \text{if } S_{i,j} = 0 \end{cases} \tag{1.29}$$

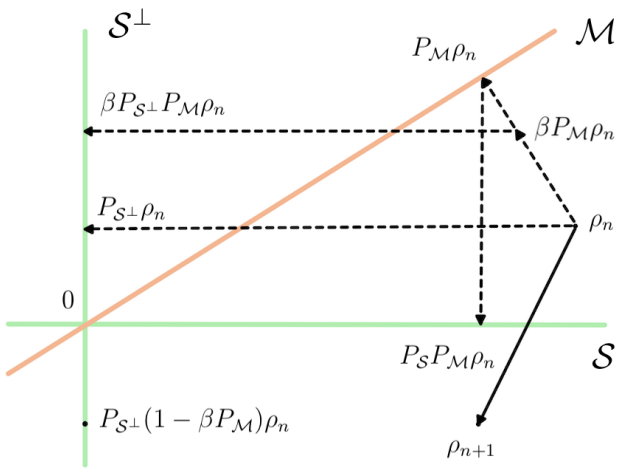


Figure 1.10: Geometric representation of the Hybrid Input Output algorithm.

In particular, the more precise the estimation of the support shape is, the nearer the value of β to 1. In order to graphically visualize its way of acting, it is worth introducing a linear definition of its dynamics:

$$\rho^{n+1} = \left[P_\mathcal{S} P_\mathcal{M} + P_{\mathcal{S}^\perp}(1 - \beta P_\mathcal{M})\right] \rho^n \tag{1.30}$$

where the operator $P_{\mathcal{S}^\perp} = 1 - P_\mathcal{S}$ is the projector on the set $\mathcal{S}^\perp$, as easily perceptible from Fig.1.10.

The HIO algorithm speeds up convergence with respect to ER and overcome a lot of the issues concerning stagnation in local minima. It strongly perturbs the guess ρ from a step to the following: for this reason HIO is particularly *ergodic*, which means that it is able to widely explore the space. In the context of golf, a ball launched with the HIO algorithm travels much more than the ER one before stopping in a minimum. The price to pay for this *ergodicity* is that the ball may pass very near to the solution and then run away.

Both HIO and ER belong to the category of *iterative projection algorithms*, a set which nowadays includes many different approaches, all of them based on the cyclical imposition of the constraints $\mathcal{M}$ and $\mathcal{S}$ through the use of *projectors*. All of them were born as attempts to find out the better compromise between the *stability* of ER and the *ergodicity* of HIO. The following subsections are just a list of the most common algorithms developed after HIO and ER, with a brief description and a geometric representation depicted at the end of this digression.

1.5.1 Solvent Flipping

The *Solvent Flipping* (SF) algorithm (Abrahams & Leslie 1996) is the simplest one after the ER, and it derives from the latter by replacing the projector on the set $\mathcal{S}$ with its reflector $R_\mathcal{S}$. Thus, a step of SF is trivially described by:

$$\rho^{n+1} = P_\mathcal{M} R_\mathcal{S} \rho^n \tag{1.31}$$

1.5.2 Difference Map

The Difference Map (Elser 2003) is actually a set of algorithms, all described by the following equation:

$$\rho^{n+1} = \{1 + \beta P_{\mathcal{S}} [(1 + \gamma_m)P_{\mathcal{M}} - \gamma_m] - \beta P_{\mathcal{M}} [(1 + \gamma_s)P_{\mathcal{S}} - \gamma_s]\} \rho^n \tag{1.32}$$

Ideal values for the best convergence are $\gamma_s = -\beta^{-1}$ and $\gamma_m = \beta^{-1}$. With this values, Eq.(1.32) turns into a simpler form:

$$\rho^{n+1} = \{1 + P_{\mathcal{S}} [(\beta + 1)P_{\mathcal{M}} - 1] - P_{\mathcal{M}} [(\beta - 1)P_{\mathcal{S}} + 1]\} \rho^n \tag{1.33}$$

It's worth noting that the operation $P_{\mathcal{M}}$, which involves the computation of one FT and one inverse FT, is required here twice, such that the computational cost for the DM algorithm is nearly twice than, for example, the HIO.

1.5.3 Averaged Successive Reflections

The Averaged Successive Reflections (ASR) (Bauschke et al. 2002) is strictly bounded to the HIO. In particular, it is a HIO algorithm where the feedback parameter $\beta = 1$, i.e.:

$$\rho^{n+1} = \frac{1}{2} [R_{\mathcal{M}} R_{\mathcal{S}} + 1] \rho^n \tag{1.34}$$

1.5.4 Hybrid Projection Reflection

Hybrid Projection Reflection (HPR) (Bauschke et al. 2003) derives from a relaxation of ASR:

$$\rho^{n+1} = \frac{1}{2} \{R_{\mathcal{S}} [R_{\mathcal{M}} + (\beta - 1)P_{\mathcal{M}}] + 1 + (1 - \beta)P_{\mathcal{M}}\} \rho^n \tag{1.35}$$

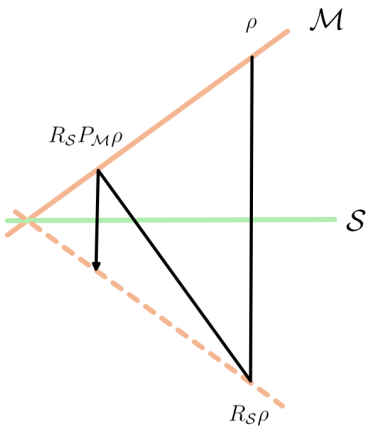
It is mathematically equivalent to HIO if the positivity of the retrieved function is not imposed.

1.5.5 Relaxed Averaged Alternating Reflectors

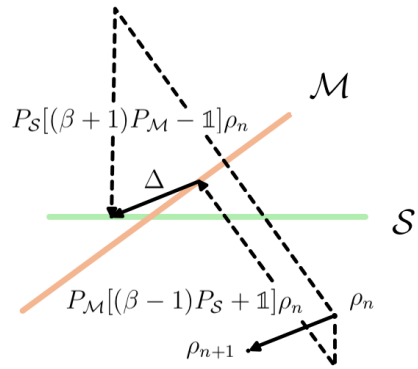
The Relaxed Averaged Alternating Reflectors (RAAR) algorithm (Luke 2004) gets the following form:

$$\rho^{n+1} = \left\{ \frac{1}{2} \beta [R_{\mathcal{S}} R_{\mathcal{M}} + 1] + (1 - \beta)P_{\mathcal{M}} \right\} \rho^n \tag{1.36}$$

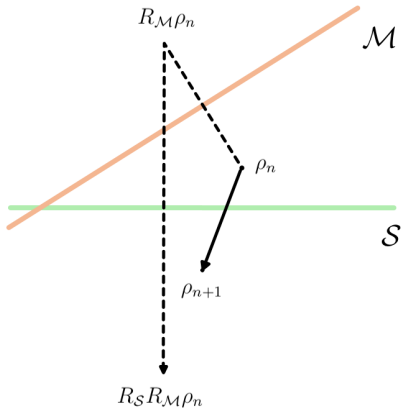
It's worth noting that, in case of $\beta = 1$, HPR, ASR and RAAR coincide.



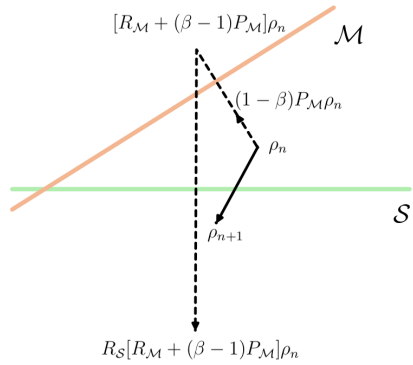
(a) SF



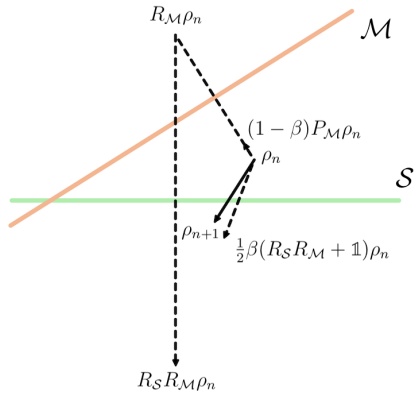
(b) DM



(c) ASR

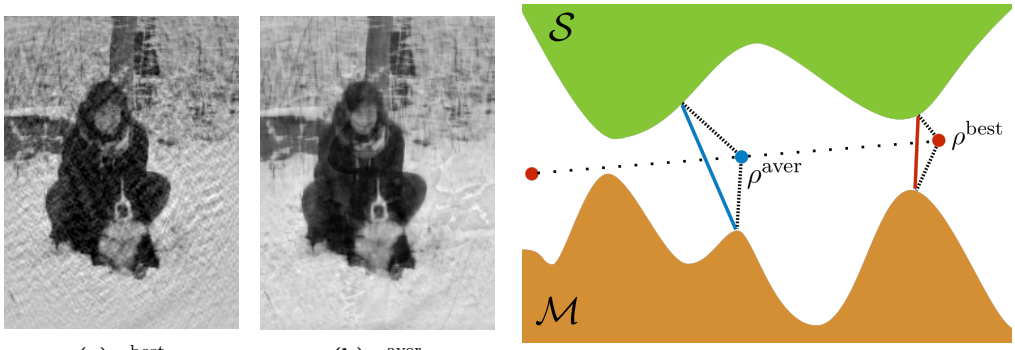


(d) HPR



(e) RAAR

Figure 1.11: Geometric representation of the most common iterative projection algorithms



(a) ρ^{best} **(b)** ρ^{aver} **(c)**

Figure 1.12: The best reconstruction (a) and the average one (b). In (c), a simplified geometric representation of (a) and (b), which is the average between the two red dots.

1.6 The state of the art (and its issues)

Algorithms presented in the previous section are the most widespread in literature. However, it is still very common to see high-level CDI results where the phase retrieval procedure has been performed by simply alternating HIO and ER algorithms. This means that, despite decades of development in phase retrieval algorithms, none of them has represented a concrete revolution in this field. It can be stated without doubts that the main improvements in CDI results are, instead, the outcomes of remarkable enhancements in the data acquisition phase and in computing hardware. In particular, the latter allowed scientists to perform many phase retrieval runs on the same dataset, getting a multitude of different results from the same diffraction pattern. Coming back to the golfer metaphor, it's as like as we have at our disposal a team of golfers composed by R players, and each of them is randomly placed in the golf course. These golfers hit their balls, and these balls, moving in the field, approach to a point of minimum altitude. At this point, the solution can be computed in different ways. The first is to label as solution the ball that reached the lowest point among the R ones, that is the phase retrieval run ρ^{best} which reached the lowest error, defined by Eq.(1.20). The risk correlated to this approach is that this solution could be really far from the true one, if the reached point is a local optimum. This situation is very common in CDI, where both low knowledge about the support function and noise in acquired data produce many deep local optima far away from the solution. The second way is to consider as solution the average position, ρ^{aver}, computed on a set of best results R_b among the R ones. This approach is commonly undertaken because it protects from the risk mentioned before. On the other side, averaging the best results R_b doesn't ensure to decrease the error value nor to come closer to the solution.

Fig.1.12 well depicts these situations. In particular, it represents two results on a trivial example, where $R = 1024$. Fig.1.12a represents the best retrieval ρ^{best}, i.e. the one with the lowest error. Fig.1.12b shows, instead, the average ρ^{aver} on the best R_b reconstructions. The latter appears clearer and less noisy, even if its error, computed via Eq.(1.20), is more than four times higher with respect to ρ^{best}. This situation is well exemplified by Fig.1.12c: the density with the lowest error is, actually, far from the solution. The average between two results ($R_b = 2$ in this trivial representation) is instead much closer to the solution, but its error is higher.

Even if, at a first glance, the high error of the average doesn't look as an issue, it is

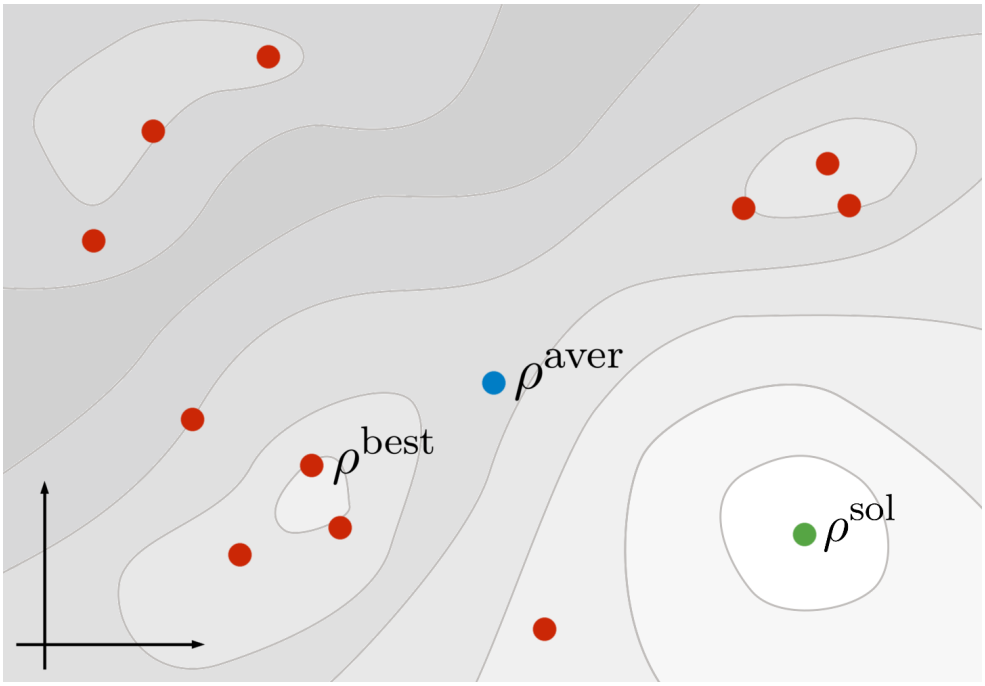


Figure 1.13: An pictorial description of a typical phase retrieval result. The gray value in the background is proportional to the error value. Each red dot in the plot represent a result of an independent phase retrieval procedure: the one labeled as ρ^{best} corresponds to the result with the lowest error. The blue point is obtained by averaging the coordinates of the red points. The green dot highlights the solution, which is placed where the error value reaches its global minimum. It represents a quite common situation when facing the phase retrieval problem, where many independent phase retrieval procedures started from different random starting guesses are performed: the best reconstruction is not the nearest to the solution and the average is nearer than the best but has a higher error value.

actually a concrete problem to deal with.

Fig.1.13 is a visualization of a hypothetical phase retrieval where the matrix ρ has only two entries. A single point in a 2D cartesian plot uniquely defines ρ, where its coordinates (x, y) are the values ρ_1 and ρ_2. The grayscale background depicts the error of those reconstructions. Red points are the R_b results, i.e. the balls launched by the golfers team that have reached the lowest altitude. As said in the previous sections, the solution ρ^{sol} resides in the point with the lowest altitude, that is the point with the lowest error. Fig.1.13 well depicts some issues to be faced during phase retrieval:

- The point nearest to the solution is not the one with the lowest error.
- Many reconstructions have some coordinates really near to the solution ones, while the other coordinates are totally wrong (see the groups below on the left and up on the right).
- The average has an error higher than both the nearest point to the solution and the point with lowest error.

- The average has lost the "good parts" of the single reconstructions. Even if some reconstructions well retrieved the first component and some others the second, their average has a worse value for both the two components.

After these considerations, issues concerning the average start to appear. In particular, the average of a set of reconstructions $\{\rho^r\}_{r=0,..,R-1}$ keeps their common features. On the other side, pixels whose values vary inside $\{\rho^r\}$ are suppressed, as previously depicted by Fig.1.13.

Fig.1.14a shows a phase retrieval result started from a random initial guess, after 100, 500 and 1000 iterations of HIO respectively. Fig.1.14b shows, instead, their discrepancy with the diffraction pattern $I_{i,j}$. These images represents, for every coordinate (i, j), the value $\frac{||\tilde{\rho}_{i,j}|-M_{i,j}|}{M_{i,j}}$. White pixels represent a discrepancy greater than 33%. Fig.1.14b well describes the behavior of a standard phase retrieval procedure: the information firstly retrieved by algorithms is the one concerning low resolution features (the central part of the diffraction pattern), while the final reconstruction error $E[\rho]$ is mainly provided by a discrepancy with the experimental data at high resolution.

Fig.1.14c shows, instead, the discrepancy between the phase of the retrieved density and the phase of the solution, known in this artificial CDI example. For every coordinate (i, j), the value $|\arg[\tilde{\rho}_{i,j}] - \arg[\tilde{\rho}^{\text{sol}}_{i,j}]|$. White pixels represent a discrepancy of π.

Comparing Fig.1.14b and Fig.1.14c, the correlation between the two kind of discrepancy looks striking. The only one that can be computed on a real CDI experiment is the discrepancy with the diffraction pattern, but, as this example shows, there is a reasonable certainty that a wrong estimated amplitude carries with him a wrong retrieved phase.

At this point, let's have a look on the effects that this consideration has on the computation of the average. The computation of the arithmetic average can be performed both in the direct and in the reciprocal space, i.e.:

$$\rho^{\text{aver}}_{i,j} = \frac{1}{R}\sum_{r=0}^{R-1} \rho^r_{i,j} \tag{1.37}$$

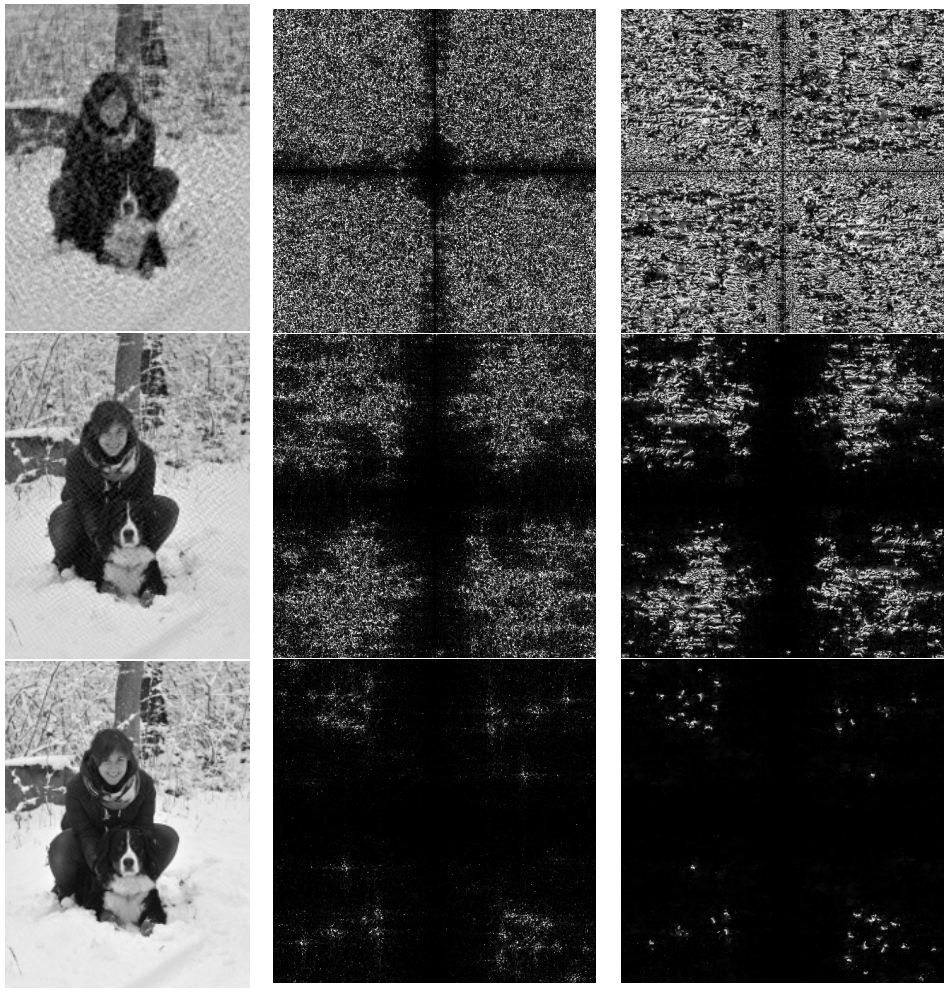
$$\tilde{\rho}^{\text{aver}}_{i,j} = \frac{1}{R}\sum_{r=0}^{R-1} \tilde{\rho}^r_{i,j}. \tag{1.38}$$

The linearity of the Fourier Transform ensures that Eq.(1.37) and Eq.(1.38) give the same result:

$$\mathsf{DFT}[\rho^{\text{aver}}] = \tilde{\rho}^{\text{aver}} \tag{1.39}$$

which means that the Fourier Transform of the average is equal to the average of the Fourier Transforms.

When the average is computed on the set on independent reconstruction results $\{\rho^r\}_{r=0,..,R-1}$, it keeps the common features, while it tends to suppress peculiar details of each element. In particular, the common features inside the set $\{\rho^r\}$ correspond to those coordinates whose value is well retrieved by all of the $\{\rho^r\}$, such that $\rho^r_{i,j} \approx \rho^{\text{aver}}_{i,j}$ and $M_{i,j} \approx |\rho^{\text{aver}}_{i,j}|$. Those coordinates whose value has, instead, a high discrepancy with the experimental data, carry also a wrong estimation of the phase, as we have previously seen. Wrong estimated values differs from a reconstruction to an other, due to the fact that the retrieval results $\{\rho^r\}$ stared from different configurations. It means that, when computing the average on those coordinates, the result given by Eq.(1.38) tends to be similar to the arithmetic average of R random complex numbers, whose value goes to 0. Thus, in this case, $|\rho^{\text{aver}}_{i,j}| \ll M_{i,j}$. Due to the fact that, as previously stated and



(a) **(b)** **(c)**

Figure 1.14: Characterization of a typical phase retrieval process. Fig.1.14a are the real space images of the reconstruction after 100, 500 and 1000 iterations respectively: only the pixels inside the support function are displayed. Fig.1.14b is the discrepancy between the amplitude of their Fourier Transform and the amplitude of the solution. Fig.1.14c is instead the discrepancy between the Fourier phases of the reconstruction and the phases of the solution. The algorithm retrieves the correct density starting from the low frequencies (i.e. the central part of diffraction data) which correspond to low resolution features in real space. It is evident, by comparing Fig.1.14b and 1.14c, the strong correlation between the error on the amplitudes (which is the discrepancy between the reconstruction amplitude and the diffraction pattern) and the error on the phase (which is unknown for a real CDI experiment).

showed in Fig.1.14b, the error of the reconstructions resides in those pixels that provide high-resolution details, the average is characterized by lower values at high frequency with respect to the solution. At the end of the story, the average ρ^{aver} tends to look like a solution ρ^{sol} subjected to a *low-pass filter*.

This is the reason why the average, at the end, does not enhance the reconstruction

quality by avoiding artifacts, but it simply sweeps them under the carpet.

An better approach to exploit these set of results $\{\rho^r\}_{r=0,..,R-1}$ is, thus needed. Maybe, a better communication among the golfers team could help.

Chapter 2

Memetic Phase Retrieval

2.1 A smarter way

Recalling what has been told at the end of the previous chapter, a smarter way to exploit the results of many phase retrieval runs is needed. This set of results carries inside a lot of information which a simple arithmetic average cannot fully exploit.

The *state of the art* previously described requires a set of R phase retrieval procedures starting from a set of different (usually random) initial configurations $\{\rho^r_{\text{start}}\}_{r=0,..,R-1}$. Each of them, after a defined number of iterations of phase retrieval algorithms, becomes a solution guess, such that a set of final states $\{\rho^r\}_{r=0,..,R-1}$ is taken into account to elect the solution.

Under the *optimization* point of view, this approach is really near to what is called *random search* (Rastrigin 1963). This stochastic optimization method is trivial, and consists in proposing many solution guesses that are randomly built. Their *error* is then evaluated and a final solution is elected among them. The standard approach to the phase problem is a sort of *random search* for *phase retrieval*, enhanced by the use of iterative algorithms that refines the randomly proposed solutions. Thus, from now on, we will refer to this method as *Random Phase Retrieval* (RPR).

It is well known that optimization via a random search approach is very general (it needs only the optimization target, without requiring its continuity or differentiability) (Matyas 1965). The price to pay for this generality is its high inefficiency (Sarma 1990). This limitation derives from the total independence among the different results. Some first attempts to overcome the issue were performed by Marchesini et al. (2005) and Chen et al. (2007). These methods, called respectively *Averaged Relaxed Average Alternating Reflections* and *Guided Hybrid Input-Output* represent a first step beyond the mere RPR approach, because they exploit the output of a RPR procedure to feed a subsequent one.

As seen before, speaking about Fig.1.13, it is common in phase retrieval that each ρ^r provides, along with some "bad features", some good ones. Following the trivial example in the figure, if we create a new ρ by combining the first component of a ρ^r on the top right and the second component of a ρ^r on the lower left, this new solution guess will be very close to the actual solution to the phase problem. Despite it appears trivial, the main problem is that, in a realistic case, we do not have any idea about the coordinates of the solution, nor its direction. This implies that we can only compute how good a ρ^r is, but we cannot discriminate the well retrieved coordinates from the bad ones.

The only way to combine different coordinates is, at the end of the story, to perform it randomly, hoping that a combination of two good ρ^r can give life to a better one. This means to create a new set of solution guesses $\{\hat{\rho}^r\}_{r=0,..,R-1}$ from the previous set of final

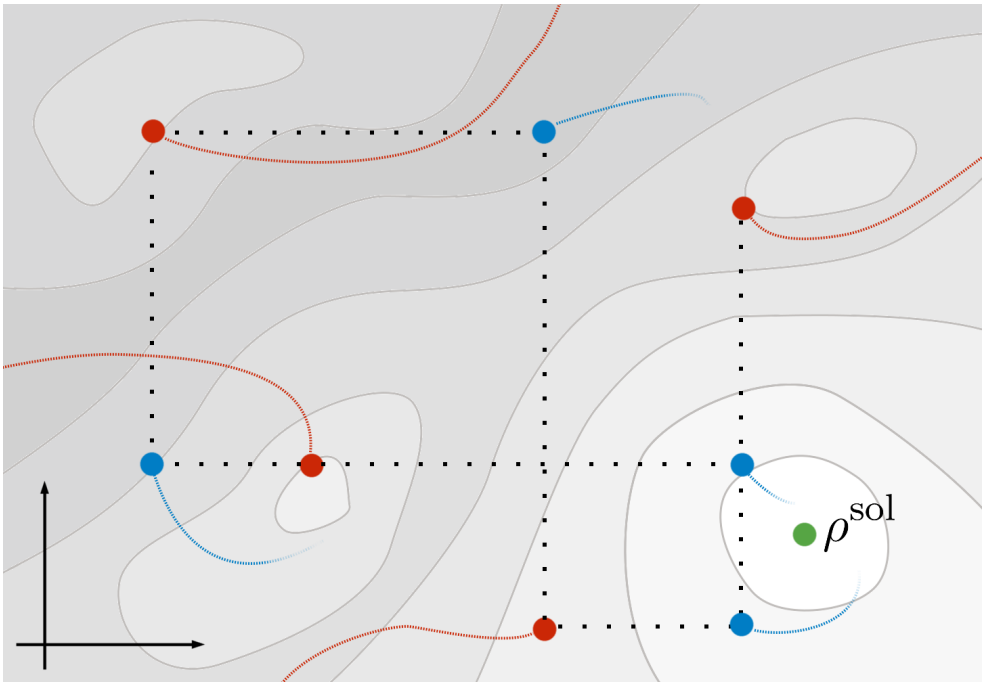


Figure 2.1: Graphic representation of the combination of 4 different phase retrieval results (red dots) to generates 4 new guesses that are used as a new starting point for iterative algorithms (blue dots).

states $\{\rho^r\}_{r=0,..,R-1}$, such that:

$$\dot{\rho}^r_{i,j} = \begin{cases} \rho^{r_1}_{i,j}, & \text{for some } (i,j) \\ \rho^{r_2}_{i,j}, & \text{for the others} \end{cases} \tag{2.1}$$

where r_1 and r_2 are indexes of two "good" elements in $\{\rho^r\}$.

A trivial representation of this operation, where $\rho = (\rho_1, \rho_2)$, is presented in Fig.2.1. Here, red dots represent $\{\rho^r\}_{r=0,..,3}$ while the red trajectories describes their previous movement in the space thanks to iterative algorithms. Blue dots are, instead, the elements of $\{\dot{\rho}^r\}_{r=0,..,3}$. The new densities, due to the operation described in Eq.(2.1), are no more in a local optimum of the error function. Thus, it is convenient to use them as a new starting guess for a new set of phase retrieval procedures, whose action is exemplified by blue lines of Fig.2.1. This means that the blind golfers in the team, after having recombined their position, launch again their balls.

If the operation of mixing the components (Eq.(2.1)) is repeated, i.e. it is iteratively applied to a set of solution guesses $\{\rho^r\}_{r=0,..,R-1}$, this produces an *evolution* of the elements inside this set. This *evolution* is peculiar of a class of stochastic optimization algorithms, called Evolutionary Algorithms (EAs) (Bäck et al. 1997; Eiben et al. 2003; Ashlock 2006). This approaches imitates the natural evolution process by treating every free parameter of the optimization , in our case $\arg[\tilde{\rho}_{ij}]$, as a gene. Thus, a complete set of values for these genes is provided by a single guess ρ, which, in this context, can be interpreted as the *genetic pool*.

EAs, following the action of nature on genomes, treat a set of genetic pools (in our

case the set $\{\rho^r\}_{r=0,..,R-1}$) as a population of individuals. This population evolves thanks to the action of the *Selection* and *Crossover* operators, which recursively create a new population (blue dots of Fig.2.1) by mixing the genes of the old one (red dots). Within this formalism, we will refer to $\mathcal{P}^{\text{parents}}$ as the set $\{\rho^r\}_{r=0,..,R-1}$ (red dots) and $\mathcal{P}^{\text{sons}}$ as the newly created set $\{\dot{\rho}^r\}_{r=0,..,R-1}$ (blue dots).

In nature, the *Selection* operator plays the role to promote the reproduction for those individuals that are better suited for the environment they live in. A famous example are giraffes: due to their need to eat leaves from tall trees, the individuals with a longer neck had an advantage with respect to the others because they had the opportunity to eat more, be more strong and healthy. These features made those individuals more fertile and more attractive for reproduction. In our case, it is suddenly clear that the role of this attractiveness is played by the error function $E[\rho]$, defined by Eq.(1.20). The individuals that better suits the "phase retrieval environment" correspond to the ones with a lower error value, or, to be more "naturalistic", an higher *fitness* value F defined as follows:

$$F[\rho] = E[\rho]^{-1} \tag{2.2}$$

Thus, the role of the *Selection* operator is to assign to every $\rho^r \in \mathcal{P}^{\text{parents}}$ a fitness value and to make a ranking[1]. A common way to perform this ranking is to rearrange the set $\mathcal{P}^{\text{parents}}$ into an ordered set $\mathcal{P}^{\text{selected}}$ with the same elements inside, but sorted from the best to the worst. This operation can be written as follows:

$$\hat{\mathsf{S}}\,\mathcal{P}^{\text{parents}} = \mathcal{P}^{\text{selected}} \,:\, F[\rho^i] \geq F[\rho^j] \;\; \forall \rho^i, \rho^j \in \mathcal{P}^{\text{selected}}, i < j \tag{2.3}$$

The second fundamental operation that allows the natural evolution is the *Crossover*. This operation provides a new individuals by combining two belonging to the population. In mathematical terms, the crossover is an operator $\hat{\mathsf{C}}$ which, generically speaking, acts on a set of genetic pools as follows:

$$\hat{\mathsf{C}}\,\mathcal{P}^{\text{parents}} = \mathcal{P}^{\text{sons}} \tag{2.4}$$

Its detailed action will depend on what is considered as the *genetic pool* of the individuals. The main feature is, however, that parents with higher ranking are preferred for the generation of a new individual.

Another operation, never mentioned before but intrinsic of natural evolution, is the *Mutation* one. Under the biological point of view, it turns out to be the fundamental step responsible of the speciation phenomenon, and the differentiation of races inside the same species. The mutation is a random modification of a genetic pool which happens independently from other factors. Without this operation, a population of different genetic pools is doomed to collapse to a unique one. Thus, the *Mutation* operation plays a main role in the adaptation because it introduces new features in the population. Some mutations are penalizing (like a giraffe with a shorter neck), some others rewarding (a longer neck), as the *Selection* step will judge. In our case, given a population of candidate solutions $\mathcal{P}$ the Mutation operator $\hat{\mathsf{M}}$ acts as:

$$\hat{\mathsf{M}}\,\mathcal{P} = \mathcal{P}^{\text{mutated}} \tag{2.5}$$

where the elements inside $\mathcal{P}^{\text{mutated}}$ are similar to the ones in $\mathcal{P}$, but randomly modified in some of their components.

[1] In the context of Evolutionary Algorithms, the *Selection* operator includes also the choice of the parents for each element that will compose the next generation. In this description, however, this role is demanded to the *Crossover* step.

These three operators, i.e. *Selection*, *Crossover* and *Mutation*, if iterated, give life to a particular EA, called Genetic Algorithm (GA) (Goldberg 1989; Koza 1994; Schmitt 2001). This approach is known to be a very general stochastic optimization method, due to the only need of the *fitness value* to optimize a quantity, without the requirement, for example, of the evaluation of its gradient.

In this digression we have temporary shelved the *iterative projection algorithms* presented in the previous chapter, such as Hybrid Input Output and Error Reduction. At a first sight, these approaches seems to be alternative to the presented GA. However, HIO and ER can be easily included in a *random search* approach as we said before, giving rise to what we called Random Phase Retrieval in the first lines of this chapter.

Following the same idea, those *iterative projection algorithms* can be easily included inside the EA framework. To do so, let's introduce a new operator $\hat{\mathsf{I}}$, called *Self-Improvement*. This operator acts on each ρ in $\mathcal{P}$ independently, by subjecting it to some iterations of projection-based algorithms. Calling $\mathsf{IP}[n]\rho^{(0)} = \rho^{(n)}$ the application of n iterations of a standard projection algorithm on a given guess ρ, the Self-Improvement operator acts as follows:

$$\hat{\mathsf{I}}\,\mathcal{P} = \{\mathsf{IP}[n]\rho^{(0)}{}^{0}, \mathsf{IP}[n]\rho^{(0)}{}^{1}, ..., \mathsf{IP}[n]\rho^{(0)}{}^{R-1}\} = \\ = \{\rho^{(n)}{}^{0}, \rho^{(n)}{}^{1}, ..., \rho^{(n)}{}^{R-1}\} = \mathcal{P}^{\text{improved}} \quad (2.6)$$

If the operator of *Self-Improvement* is added to *Selection*, *Crossover* and *Mutation*, this quartet of operators defines a peculiar kind of EA, called Memetic Algorithm (MA) (Moscato et al. 1989, 2004; Moscato & Cotta 2003; Ong et al. 2010). This type of stochastic optimization procedure is conceptually identical to a GA, but introduces a further dynamic beyond the purely-stochastic genetic one. In fact, each individual belonging to the population of candidate solutions $\mathcal{P}$ has the opportunity to improve itself (*Self-Improvement*) before *Selection*. Under a biological point of view, the single individual can develop a "culture" during its life that enhances its probability or reproduce. As the dictionary says, a *Meme* is "an element of a culture or system of behaviour passed from one individual to another by imitation or other non-genetic means": for this reason, this approach is addressed as *Memetic*.

Defining a MA framework for phase retrieval is now trivial, because it only needs the correct placing of Selection (Eq.(2.3)), Crossover (Eq.(2.4)), Mutation (Eq.(2.5)) and Self-Improvement (Eq.(2.6)).

Given a starting population of candidate solutions to the problem $\mathcal{P}^{(0)}$, a generation of *Memetic Phase Retrieval* (MPR) (Colombo et al. 2017) algorithm is described as:

$$\mathcal{P}^{(n+1)} = \hat{\mathsf{I}}\,\hat{\mathsf{M}}\,\hat{\mathsf{C}}\,\hat{\mathsf{S}}\,\mathcal{P}^{(n)} \quad (2.7)$$

Within the same formalism, just for comparison, it is possible to describe the *Genetic Phase Retrieval* (GPR):

$$\mathcal{P}^{(n+1)} = \hat{\mathsf{M}}\,\hat{\mathsf{C}}\,\hat{\mathsf{S}}\,\mathcal{P}^{(n)} \quad (2.8)$$

and the Random Phase Retrieval (RPR):

$$\mathcal{P}^{(n+1)} = \hat{\mathsf{I}}\,\hat{\mathsf{S}}\,\mathcal{P}^{(n)} \quad (2.9)$$

The flowchart in Fig.2.2 outlines the MPR operations defined by Eq.(2.7): also RPR and GPR are represented. If, from one side, the RPR approach corresponds to the standard one, some words should be spent about the GPR one. Pure Genetic Algorithms

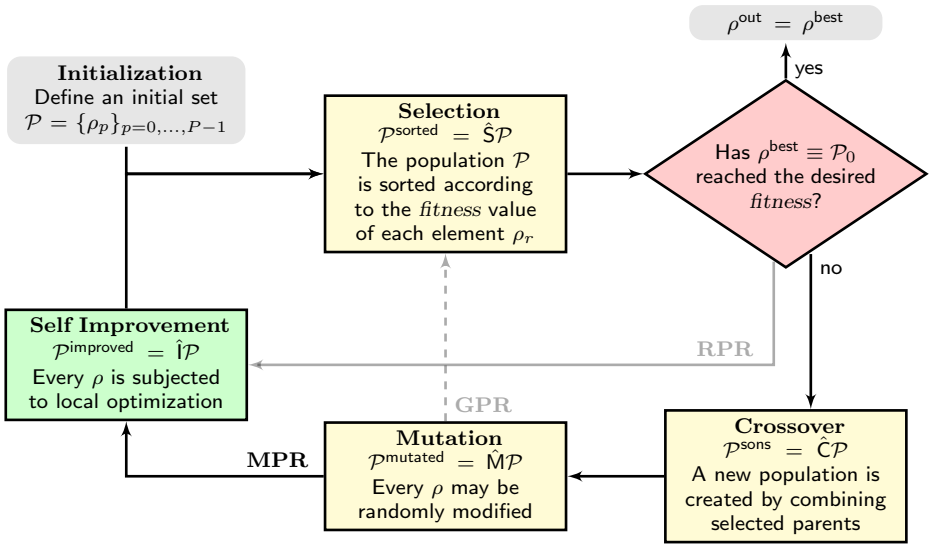


Figure 2.2: The Memetic Phase Retrieval algorithm flowchart, which is composed by the iteration of the Selection, Crossover, Mutation and Self-Improvement steps. It's worth noting that, inside this scheme, it is possible to visualize also the flowchart of the Random Phase Retrieval (RPR) approach, which lacks the Crossover and Mutation operators, and the Genetic Phase Retrieval approach, which lacks the Self-Improvement step.

are known to be very ductile, as said before. The price to pay is the inefficiency: it means that the higher the number of free parameters of the optimization problem, the higher the number of individuals of the population. Such a high dimensionality optimization problem like *phase retrieval* would require an unacceptable computational cost, concerning both memory occupancy and computation weight. For this reason, at the moment and with the current computing hardware a pure stochastic GA isn't a viable way (even if it could be in a, probably far, future).

After this last comment, it's time to completely focus on the title of this chapter (as well as the main topic of this work), that is the Memetic Phase Retrieval approach.

2.2 Implementation

In this section implementation aspects of MPR will be deepened. In particular, its steps will be described using pseudocodes. Just to become familiar with pseudocode notation[2], let's describe the trivial operations of the projection operators defined by Eq.(1.23) and Eq.(1.24).

Given these two implementations, it's now trivial to implement the function that performs n_{it} iterations of Error Reduction algorithm, as described by Alg.2.

A little more complex implementation is required by the Hybrid Input Output algorithm, due to the fact that the support constraint is no more a simple projection operation. The following Alg.3 describes the implementation of the HIO approach.

[2]In the following pseudocodes, **for** loops like the one at line 3 of Alg.1 must be interpreted as nested loops where the two indexes assume values from 0 to $N-1$.

Algorithm 1 Implementation of the two projectors $P_{\mathcal{M}}$ and $P_{\mathcal{S}}$

```
function PM(ρ)
    ρ̃ ← DFT(ρ)
    for i, j ← 0 to N do
        ρ̃[i, j] ← M[i, j] ∗ exp(√−1 ∗ arg(ρ̃[i, j]))
    end for
    ρ ← DFT^{−1}(ρ̃)
    return ρ
end function
```

```
function PS(ρ)
    for i, j ← 0 to N do
        if S[i, j] = 0 then
            ρ[i, j] ← 0
        end if
    end for
    return ρ
end function
```

Algorithm 2 Error Reduction

```
function ER(ρ, n_it)
    for i ← 0 to n_it do
        ρ ← PM(ρ)
        ρ ← PS(ρ)
    end for
    return ρ
end function
```

At this point, we can proceed with MPR implementation. A first consideration is that MPR is implemented in C++ language. C++, like other high-level programming languages (like Fortran, Java, Python), is *object-oriented*. This, put simply, means that the user its free to create new data types (objects) by collecting the fundamental types. In C++, those objects are called *Classes*. The flexibility of objects allows us to define the single properties of an *individual* of the population, declared by Alg.4.

Then, an array of individuals is declared, such that the value of the matrix ρ at coordinates (i,j) of the individual with index r inside $\mathcal{P}$ is accessed through $\mathcal{P}[r]\,.\,\rho[i,j]$. The error value of the same individual with index r in $\mathcal{P}$ is $\mathcal{P}[r]\,.\,E$. Let's now proceed to show the implementation of each step of Fig.2.2, starting from the **Initialization** one.

2.2.1 The *Initialization* step

The **Initialization** step is responsible of the creation of the starting population $\mathcal{P}$, i.e. the array of individuals. The starting guesses for the population are randomly made, starting from a single given ρ^{start} as Alg.5 describes.

Here, the coefficient $C_{RF} \in (0, 1]$, let's call it *random fill coefficient*, depends on the confidence level we have in the starting guess ρ^{start}. If ρ^{start} is known to be far from the solution, C_{RF} will have values near or equal to 1. This means that InitDensity(ρ^{start}) returns a matrix ρ with totally random values, due to the assignment of fully random phases in the reciprocal space. On the other side, the nearer ρ^{start} to the solution, the lower

Algorithm 3 Hybrid Input Output

```
function CS(ρ,ρ′)
    for i, j ← 0 to N do
        if S[i, j] = 0 then
            ρ[i, j] ← ρ′[i, j] − β ∗ ρ[i, j]
        end if
    end for
    return ρ
end function
```

```
function HIO(ρ, n_it)
    for i ← 0 to n_it do
        ρ′ ← ρ
        ρ ← PM(ρ)
        ρ ← CS(ρ,ρ′)
    end for
    return ρ
end function
```

Algorithm 4 Implementation of the population $\mathcal{P}$

```
1: Class INDIVIDUAL
2:     ρ                                          ▷ The matrix of complex values
3:     S                                          ▷ The support function
4:     M                                          ▷ The matrix containing the measured amplitudes
5:     E                                          ▷ The assigned error value
6:     F                                          ▷ The assigned fitness value
7: EndClass
8: P = INDIVIDUAL[R]          ▷ C-style declaration of an array of R elements of type INDIVIDUAL
```

Algorithm 5 The Initialization step

```
1: function INITDENSITY(ρ^start)
2:     ρ ← ρ^start
3:     ρ̃ ← DFT(ρ)
4:     for i, j ← 0 to N do
5:         Phase ← arg(ρ̃[i, j]) + C_RF * RAND(−π, π)
6:         ρ̃[i, j] ← M[i, j] * exp(√−1 * Phase)
7:     end for
8:     ρ ← DFT^−1(ρ̃)
9:     return ρ
10: end function
```

```
1: function INITIALIZATION(P, ρ^start)
2:     for r ← 0 to R do
3:         P[r] . ρ ← INITDENSITY(ρ^start)
4:     end for
5:     return P
6: end function
```

the value of C_{RF}. Fig.2.3 well exemplifies the concept, where two different populations $\mathcal{P}$ have been initialized via the function INITIALIZATION with different values of C_{RF}.

2.2.2 The *Selection* step

As previously said, the *Selection* step has the role to promote those elements inside $\mathcal{P}$ that reached the highest fitness values. As mentioned before, this step is actually composed by two sub-steps: the error evaluation and the sorting. The first is based on Eq.(1.20), while the latter sorts the population $\mathcal{P}$, which is implemented as an array, such that the first element $\mathcal{P}[0]$ is the one with the highest fitness value (Eq.(2.2)). The description of this step is provided by Alg.6.

Algorithm 6 The Selection step

```
1: function GETERROR(ρ)
2:     ρ̃ ← DFT(ρ)
3:     Error ← 0
4:     Norm ← 0
5:     for i, j ← 0 to N do
6:         Error ← (M[i, j] − |ρ̃[i, j]|)^2
7:         Norm ← M[i, j]^2
8:     end for
9:     return √(Error/Norm)
10: end function
```

```
1: function SELECTION(P)
2:     for r ← 0 to R do
3:         P[r] . E ← GETERROR(P[r] . ρ)
4:         P[r] . F ← 1/P[r] . E
5:     end for
6:     P ← SORT(P)
7:     return P
8: end function
```

The function GETERROR provides indeed a normalized version of Eq.(1.20), as can be easily seen at line 9, where the normalization is the sum of all the square amplitudes (line 7). After this *Selection* step, an **if** clause will determine the end of the whole procedure if the desired error value has been reached by $\mathcal{P}[0]$, such that **if** $\mathcal{P}[0] \,.\, E \leq E^{\text{desired}}$ **then** EXIT().

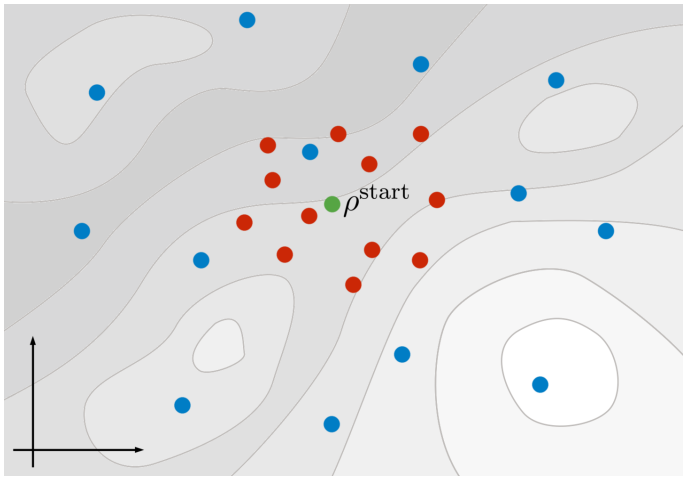


Figure 2.3: Exemplification of the Initialization step. Given a starting guess (green dot), blue dots depicts a population $\mathcal{P}$ initialized with a high value for C_{RF}, while a lower value produces a less spread population (red dots).

2.2.3 The *Crossover* step

The *Crossover* operation is the core of any EA. Its role is the creation of a new population of candidate solutions to the problem by mixing the genes of parents. In our case, those genes are the single entries of the matrix ρ. Let's proceed, dividing its implementation in smaller sections. The first operation for a crossover is the retrieval of the parents. As said before, these parents must be extracted from the population $\mathcal{P}$, preferring the ones with lower index (due to the action of the *Selection* operator). This extraction is accomplished by a *rigged roulette* method, defined in Alg.7.

Algorithm 7 The Crossover step: the Rigged Roulette

```
1: function ROULETTE(P)
2:     index ← ⌊ RAND(0,1)^{C_R} ∗ (R − 1)⌋
3:     return P[index] . ρ
4: end function
```

The operation encoded at line 2 first of all extracts a random number with a flat distribution in the range $[0, 1)$. Then, this value is elevated to the power C_R, called *rigged roulette coefficient*. When $C_R > 1$, $\text{RAND}(0, 1)^{C_R}$ turns into an unbalanced distribution, where values near 0 are more likely than values near to 1. Thus, because all is multiplied by $(R-1)$, the integer part of $\text{RAND}(0, 1)^{C_R} * (R-1)$ will become an integer distribution in $[0, R-1]$ where the closer the value to 0, the easier it is to be extracted. In this way, the function ROULETTE will provide an element of the population, which will be more likely to come from the first part of the array $\mathcal{P}$.

The second component of the *Crossover* operator concerns the way in which the genetic pool is mixed. As said before, the genetic pool is represented by the values ρ_{ij} at each coordinate (i, j) of the matrix ρ. A "standard" crossover requires two parents ρ^{p_1} and ρ^{p_2}, and the result of their merging ρ^{son} has values ρ^{son}_{ij} which are randomly inherited from ρ^{p_1} or from ρ^{p_2}. In MPR a peculiar type of crossover has been implemented instead, called *Differential Crossover* (Storn & Price 1997), which requires four parents, ρ^{p_1}, ρ^{p_2},

ρ^{p_3} and ρ^{p_4}, instead of two. Its implementation is declared by Alg.8.

Algorithm 8 The Crossover step: Part II

```
 1: function REPRODUCTION(ρ^p1, ρ^p2, ρ^p3, ρ^p4)
 2:     ρ
 3:     for i, j ← 0 to N do
 4:         if RAND(0,1) > C_GB then
 5:             ρ[i, j] ← ρ^p1[i, j]
 6:         else
 7:             ρ[i, j] ← ρ^p2[i, j] + C_D * (ρ^p3[i, j] − ρ^p4[i, j])
 8:         end if
 9:     end for
10:     return ρ
11: end function
```

This implementation depends on two parameters. The first one, C_{GB}, is called *Genetic Balance Coefficient*, and defines how many pixels of the newly created ρ are chosen from ρ^{p_1} or from the combination of the other three parents. Its value runs from 0 to 1: 0.33 means, for example, that about two thirds of the values are taken from ρ^{p_1}, while for $C_{GB} = 0.5$ about the half. The second parameters C_D is called *Differential Coefficient*. Possible values go from 0 (in this case it is equivalent to a "standard" crossover) up to, commonly, 2, while a standard value is $C_D = 1$. The role of this *Differential* approach is to better explore those coordinates (i, j) whose value varies a lot from an individual to an other. If two parents ρ^{p_3} and ρ^{p_4} are selected, whose values at coordinates (i, j) differ a lot, the generated son ρ^{son} will have a value ρ^{son}_{ij} really different from both ρ^{p_2}, ρ^{p_3} and ρ^{p_4}. Thus, this approach aims to better explore the space of configurations on those coordinates that are held responsible for the missing achievement of the solution.

Once we have seen how parents are extracted and how they are combined to give life to a new candidate solution, its now the time to show how those functions are exploited to generate a new population of candidate solutions from the previous one. Alg9 provides the implementation of the whole Crossover operator.

Algorithm 9 The Crossover step: Part III

```
 1: function CROSSOVER(P)
 2:     P^sons = INDIVIDUAL[R]
 3:     if F_FC = 1 then
 4:         for r ← 0 to R do
 5:             P[r] . ρ ← DFT(P[r] . ρ)
 6:         end for
 7:     end if
 8:     for r ← 0 to R do
 9:         ρ^p1 ← ROULETTE(P)
10:         ρ^p2 ← ROULETTE(P)
11:         ρ^p3 ← ROULETTE(P)
12:         ρ^p4 ← ROULETTE(P)
13:         P^sons[r] . ρ ← REPRODUCTION(ρ^p1,ρ^p2,ρ^p3,ρ^p4)
14:     end for
15:     P ← P^sons
16:     if F_FC = 1 then
17:         for r ← 0 to R do
18:             P[r] . ρ ← DFT^−1(P[r] . ρ)
19:         end for
20:     end if
21:     return P
22: end function
```

The core of this step runs from line 8 to line 14, which is a loop on each element of $\mathcal{P}^{\text{sons}}$. Inside this loop, every $\mathcal{P}^{\text{sons}}[r]$ is generated by combining via Alg.8 four *parents* extracted by Alg.7. The only parameter in this block is the *Fourier Crossover Flag* F_{FC}. If false ($F_{FC} = 0$), the genetic pool is is considered to be the entries of ρ_{ij} in direct space.

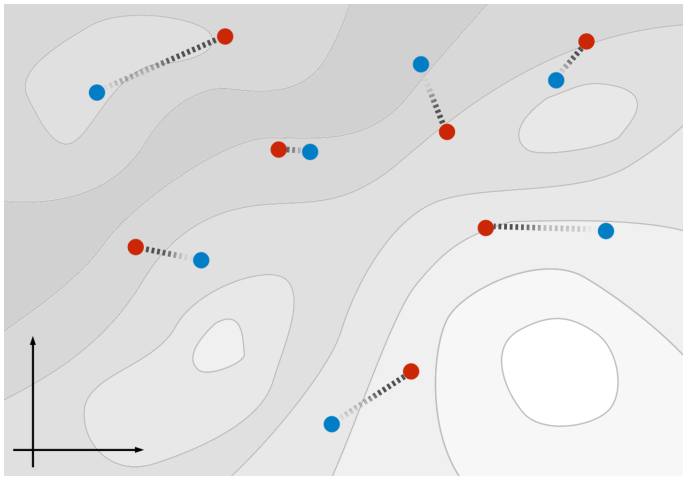


Figure 2.4: Exemplification of the Mutation step. Given the population $\mathcal{P}$ (blue dots), the Mutation operator randomly moves every element of $\mathcal{P}$, generating a new population $\mathcal{P}^{\text{mutated}} = \hat{\mathsf{M}}\mathcal{P}$ (red dots).

If true, the crossover acts instead on the Fourier representation of ρ, i.e. the genetic pool becomes the entries of $\mathsf{FT}[\rho]_{ij}$.

2.2.4 The *Mutation* step

Let's proceed now with the implementation of the mutation operator. In a standard GA, this operator is the true responsible of the GA success in optimizing the *fitness* values. In fact, as exemplified in Fig.2.4, this operator randomly moves each $\mathcal{P}[r]$ in the space of configuration by modifying the values $\mathcal{P}[r] \cdot \rho_{ij}$.

Its implementation is shown in Alg10 and depends on three parameters, C_{MP}, C_{MF} and C_{ML}.

Algorithm 10 The Mutation step

```
1:  function MUTATION(P)
2:      for r ← 0 to R do
3:          if RAND(0,1) < C_MP then
4:              ρ̃ ← DFT(P[r] · ρ)
5:              for i, j ← 0 to N do
6:                  if RAND(0,1) < C_MF then
7:                      phase ← arg[ρ[i,j]]+RAND(−C_ML π, C_ML π)
8:                      ρ[i,j] ← |ρ[i,j]| ∗ exp(√−1 ∗ phase)
9:                  end if
10:             end for
11:             P[r] · ρ ← DFT^−1(ρ̃)
12:         end if
13:     end for
14:     return P
15: end function
```

The first one, the *Mutation Probability Coefficient*, defines how likely a mutation is. For example, $C_{MP} = 0.2$ means that about one individual every five will be subjected to mutation.

The second parameter C_{MF}, called *Mutation Fraction Coefficient*, determines instead how many genes (i.e. coordinates (i, j)) will be mutated, such that, if $C_{MF} = 0.5$, about

the fifty percent of them will be modified.

The latter C_{ML}, the *Mutation Level Coefficient*, tunes the entity of the mutation. For example, if $C_{ML} = \frac{1}{3}$, the phase will be shifted by a random value in the interval $[-\frac{\pi}{3}, \frac{\pi}{3}]$. As consequence, $C_{ML} = 1$ means that, for each coordinate elected for mutation, the phase will get a fully random value.

2.2.5 The *Self-Improvement* step

The operators seen so far are the ones that defines what we have called *Genetic Phase Retrieval*. This further *Self-Improvement* step, as said before, turns this approach into a Memetic Algorithm.

As previously said, this step is nothing more than the application of *iterative projection algorithms* on every candidate solution inside $\mathcal{P}$. Its implementation is trivial. The tricky point resides in the fact that any sequence of algorithms can be provided to the code, such that there will be a list of N_{alg} algorithm names $\mathsf{A}_{\mathsf{name}}$ and the relative list of the number of their iterations, A_{it}.

Algorithm 11 The Self-Improvement step

```
function SelfImprovement(P)
    for r ← 0 to R do
        for n ← 0 to N_alg do
            if A_name[n] = ER then ER(P[r].ρ, A_it[n])
            else if A_name[n] = HIO then HIO(P[r].ρ, A_it[n])
            else if A_name[n] = RAAR then RAAR(P[r].ρ, A_it[n])
            else if A_name[n] = DM then DM(P[r].ρ, A_it[n])
            else if ... then ...(P[r].ρ, ...)
            end if
        end for
    end for
    return P
end function
```

If $\mathsf{A}_{\mathsf{name}} = \{\text{HIO}, \text{DM}, \text{ER}\}$ and $\mathsf{A}_{\mathsf{it}} = \{20, 40, 10\}$, every $\mathcal{P}[r].\rho$ will undergo 20 iterations of HIO, 40 of DM and 10 of ER.

2.2.6 Monitoring the process

A routine dedicated to the control of the procedure is necessary. In particular, the tools useful to provide information about the behavior of the phase retrieval procedure are the errors of the individuals ($\mathcal{P}[r].E$), the image of the best individual and, as we will show later, the image of the "average citizen" of this population. For this reason, the function described by Alg.12 is inserted in the main loop of MPR.

Algorithm 12 The Monitor function

```
function Monitor(P)
    I^best ← P[0]
    I^aver ← GetAverage(P)
    PrintErrors(P)
    PrintImage(I^best)
    PrintImage(I^aver)
    return I^best, I^aver
end function
```

This function retrieves the best individual $\mathcal{I}^{\text{best}}$ of the population and the average one, $\mathcal{I}^{\text{aver}}$, computed via a simple arithmetic mean. After that, their images, $\mathcal{I}^{\text{best}} \cdot \rho$

and $\mathcal{I}^{\text{aver}}\,.\,\rho$, are saved to the disk and returned by the function. Also the errors of every element in $\mathcal{P}$ is printed to a file.

2.2.7 The MPR algorithm

At this point we are ready to build the house with the bricks we have created. The (very simple) MPR main loop is depicted by Alg.13.

Algorithm 13 The MPR main loop

```
 1: function MPR
 2:     R, N, C_RF, C_MP, C_ML, ... ← ReadParameters      ▷ Parameters are read from a configuration
                                                            file and their values are assigned
 3:     P ← Individual[R]                                 ▷ An array P of R Individuals is declared
 4:     M, S, ρ^start ← ReadData                          ▷ The matrices representing the Module M
                                                            and Support S constraint are read, along
                                                            with a starting guess, if provided
 5:     P ← Initialization(P, ρ^start) ReadData           ▷ The array P is initialized
 6:     n_step ← 0
 7:     loop
 8:         P ← Selection(P)
 9:         I^best, I^aver ← Monitor(P)
10:         if P[0].E < E^target ∨ n_step = N_steps then
11:             return I^best.ρ                           ▷ The algorithm ends if the desired error
                                                            value or if the maximum number of steps
                                                            are reached. The best individual is re-
                                                            turned as output
12:         end if
13:         P ← Crossover(P)
14:         P ← Mutation(P)
15:         P ← SomeOtherStuff(P)
16:         P ← SelfImprovement(P)
17:         n_step ← n_step + 1
18:     end loop
19: end function
```

The first two functions called before the main loop have to role to read data from the disk. The first one, ReadParameters, reads form a configuration file the value for those parameters. The second function, ReadData, loads two matrices of size $N \times N$, M_{ij} and S_{ij}, that represent the two constraints of the *phase retrieval problem*, and a starting guess for the density, if given. After the initialization step at line 5, the procedure enters in the main loop, whose exit condition is the **if** clause at line 9. All of the functions called in the main loop have been previously described, except the procedure SomeOtherStuff. This function collects inside some operations that are peculiar of the treated experimental data. Those operations will be described when results will be presented. Among them there is, however, a noteworthy one, which in some sense can be interpreted as part of the Self-Improvement step. Its name is *Shrinkwrap algorithm* (Marchesini 2007) and aims to refine the estimation of the support function S, when the provided one is too inaccurate.

2.2.8 The MPR version of the *Shrinkwrap* approach

The main query concerning CDI that could have risen to the reader resides in the support function. As said in the previous sections, the support function is a necessary constraint to ensure the existence of a solution to the *phase problem* and provide sufficient information to *phase retrieval algorithms*. The support function is, in practice, the shape of ρ, as clearly depicted in Chapter I by 1.7b.

Defining an a-priori support function is nothing simple, because a pre-existent knowledge on the sample is required. When, in 1999, the first CDI with X-rays was accomplished (Miao et al. 1999, see) the shape of the support was provided by a low-resolution image

Figure 2.5: A graphical exemplification of the action of the Shrinkwrap algorithm on the support constraint $\mathcal{S}$. Step by step, the *Shrinkwrap* algorithm tends to shrink the support constraint, as the steps from light green to dark green describe.

of the sample from a X-ray microscope. This need of a-priori information was a strong limitation: first of all, a non-perfect estimation of the support worsens a lot the algorithm convergence performance. The second issue is much general, and resides in the fact that a low-resolution image of the sample is, in most cases, unattainable. Thus, until 2003, CDI could be interpreted just as a method to enhance resolution instead of a standalone imaging technique.

Things, however, changed thanks to the work of Marchesini et al. (2003), when a self-consistent method to extrapolate a support function was proposed, making CDI a true high-resolution standalone imaging approach. The proposed method, subsequently referred as *Shrinkwrap*, exploits the current estimation of $\rho^{(n)}$, which has undergone, let's say, n iteration of HIO algorithm, to provide a new estimation of the support S_{ij}, which will be used as the new support constraint for the next n iterations.

This method is formed by two main steps:

1. Smooth a given ρ by convolving it with a Gaussian function characterized by a given σ. The higher the σ, the smoother the result.

2. Apply a threshold to the smoothed ρ^{smooth}, such that the new support is $S_{ij} = 1$ if $\rho_{ij}^{\text{smooth}} > \tau$, $S_{ij} = 0$ otherwise.

This operation, repeated every n iterations of iterative projection algorithms, sharpens the support constraint. Under a conceptual point of view, this approach works well because, shrinking the set $\mathcal{S}$ in Fig.2.5, it makes the gradient of the distance between the two constraints much more marked, helping in this way the convergence of projection algorithms.

On the other side, as Fig.2.5 depicts, this approach often tends to move away the two sets also in the area near the solution. This phenomenon derives from the use of a threshold value for the estimation of the support: it commonly throws away some low intensity pixels in the direct space which, instead, should be retrieved. This effect is usually neglected, due to the fact that those low intensity pixels are artifacts caused by noise in the diffraction data.

If, for a standard approach, this doesn't represent a big issue, it is instead a problem for the MPR approach. In fact, as can be understood again from Fig.2.5, a badly retrieved ρ with a loose support function can provide an error lower than a good ρ with a tailored support.

It turns out that the error value, provided by Eq.(1.20) and computed via Alg.6, cannot be intended as an absolute parameter for the evaluation of the solution. Thus, if the support function is independently updated for every individual $\mathcal{P}[r]$, the resulting support functions $\mathcal{P}[r] \centerdot S$ will have different shapes and extensions. If the error value is computed imposing supports with different extensions $A_S = \sum_{ij} S_{ij}$, it means that the resulting errors are not comparable. If these errors are not comparable, the MPR Selection step will fail to identify the best reconstructions and the procedure will go away from the correct solution.

For this reason, the *Shrinkwrap* algorithm implementation inside MPR is more tricky. In fact, for every MPR step, it is necessary that all the support functions $\mathcal{P}[r].S$ assigned to each individual have the same area, in order to let the Selection provide a fair evaluation. Alg.14 describes this implementation.

Algorithm 14 The MPR version of the *Shrinkwrap* algorithm: Part I

```
1: function EvolveSupport(P, I^best)
2:     A_S ← GetSupportArea(I^best • ρ)
3:     for r ← 0 to R do
4:         P[r] • S ← ShrinkWrap(P[r] • ρ, A_S)
5:     end for
6:     return P
7: end function
```

Algorithm 15 The MPR version of the *Shrinkwrap* algorithm: Part II

```
1: function GetSupportArea(ρ)
2:     ρ^smooth ← Convolve(ρ, G(0, σ))
3:     A_S ← 0
4:     for i, j ← 0 to N do
5:         if ρ^smooth[i, j] > τ* Max(ρ^smooth) then
6:             A_S ← A_S + 1
7:         end if
8:     end for
9:     return A_S
10: end function
```

This Shrinkwrap version is divided into two parts. The first part is represented by the GetSupportArea function, whose implementation follows in Alg.15.

This routine, given an input density ρ, computes the area that would have the support if it were updated via the *Shrinkwrap* algorithm. This area is computed on the best individual $\mathcal{I}^{\text{best}}$ and it is passed as an argument to the ShrinkWrap procedure which is called for every individual in $\mathcal{P}$.

The ShrinkWrap function of MPR, described by Alg.16, is a peculiar one. Instead of accepting a threshold value τ for the support update, it gets the desired area A_S. Starting from this value, the procedure extract a temporary threshold value τ_{Local} that will provide the given support area.

This function EvolveSupport (Alg.14) has to be intended as part of the routine SomeOtherStuff of Alg.13.

Algorithm 16 The MPR version of the *Shrinkwrap* algorithm: Part III

```
1: function ShrinkWrap(ρ, A_S)
2:     S^new
3:     Values
4:     ρ^smooth ← Convolve(ρ, G(0, σ))
5:     for i, j ← 0 to N do
6:         Values[i + N ∗ j] ← ρ^smooth[i, j]      ▷ Values of the smoothed density are assigned to a one-dimensional array
7:     end for
8:     Values ← Sort(Values)
9:     τ_Local ← Values[A_S]
10:    for i, j ← 0 to N do
11:        if ρ^smooth[i, j] > τ_local then
12:            S^new[i, j] ← 1
13:        else
14:            S^new[i, j] ← 0
15:        end if
16:    end for
17:    return S^new
18: end function
```

2.2.9 Table of MPR parameters

Fig.2.6 sums up the parameters on which an MPR phase retrieval procedure depends. These parameters highly characterize the behavior of the algorithm, by making it more *stable* or more *ergodic*. A section of the next chapter will be entirely dedicated to describe MPR behavior as function of these parameters.

2.2.10 Dimensional considerations

What we have seen so far has been all presented in the 2D formalism. Actually, all the showed pseudocodes can be trivially translated in the 3D case by performing few simple replacements:

- Matrix indexes, i.e. the coordinates, become three. For example, $\rho[i,j]$, $M[i,j]$, $S[i,j]$ become $\rho[i,j,k]$, $M[i,j,k]$, $S[i,j,k]$
- As consequence, the **for** loops on matrix indexes become **for** $i,j,k \leftarrow 0$ to N.
- The Discrete Fourier transform has to be intended in 3D instead of 2D.

If under the implementation side differences between 2D and 3D are almost non-existent, such that the same MPR code can run both in 2D and 3D, much more critical computational issues arise. Let's have a look on the computational weight of MPR approach, starting from its memory occupancy.

As shown by Alg.13, the array $\mathcal{P} \leftarrow \text{Individual}[R]$ is the main responsible of memory occupancy. The memory use of a single $\mathcal{P}[r]$ is mainly due to the complex matrix $\mathcal{P}[r].\rho$. Thus, the memory occupancy of the whole approach can be extracted by the following relation:

$$W = 2 \times N^D \times T \times R \tag{2.10}$$

where N is the number of pixels on a coordinate, D is the number of dimensions (2 or 3), T is the size in Bytes of a single value ρ_{ij}, R is the size of the population $\mathcal{P}$. The factor 2 derives from the Crossover step, where the population $\mathcal{P}$ and the new one $\mathcal{P}^{\text{sons}}$ must coexist. It's worth noting that the value W provides only a lower bound to memory occupancy, due to the fact that the matrices S (the support function) and M (the diffraction data) are neglected.

Name	Range	Description
R	$[\mathbf{4}, \infty)$	**Number of individuals** of the population $\mathcal{P}$
C_{RF}	$(\mathbf{0}, \mathbf{1}]$	**Random Fill Coefficient**: it regulates how much the initial population is spread from the initial guess;
C_R	$(\mathbf{1}, 3]$	**Rigged Roulette Coefficient**: the higher the value, the more likely low error individuals are used for reproduction
C_{GB}	$(\mathbf{0}, \mathbf{1})$	**Genetic Balance Coefficient**: fraction of the genetic pool that will be inherited by the combination of the second, the third and the fourth parent, instead of the first, following the *Differential Crossover* scheme.
C_D	$[\mathbf{0}, 2]$	**Differential Coefficient**: it sets the strength of the differential part of the crossover operator. If 0, the crossover becomes of "standard" type
F_{FC}	$0, 1$	**Fourier Crossover Flag**: If 1, the crossover is performed in reciprocal space, otherwise in the real space.
C_{MP}	$[\mathbf{0}, \mathbf{1}]$	**Mutation Probability Coefficient**: it is the probability, for an element in $\mathcal{P}$, to undergo mutation
C_{MF}	$[\mathbf{0}, \mathbf{1}]$	**Mutation Fraction Coefficient**: it is the probability, for a given gene, to mutate
C_{ML}	$[\mathbf{0}, \mathbf{1}]$	**Mutation Level Coefficient**: it sets the strength of the mutation. $C_{ML} = 1$ means totally random.
N_{alg1} N_{alg2} ...	$[\mathbf{0}, 100]$	**Algorithm sequence:** the sequence of algorithms during the self-improvement step. E.g., $N_{\text{RAAR}} = 40, N_{\text{ASR}} = 20$ means that 40 iterations of RAAR and 20 of ASR are performed on each individual.

Figure 2.6: Main MPR parameters. The first column refers to parameters names. The last column, instead, is a description of the parameters. The column in the middle provides the usual range for parameters: intervals delimited by round brackets mean that the extreme of the interval is excluded, otherwise included (if denoted by square brackets). Limits in bold are *strong* limits, i.e. lower (or higher) values are forbidden, otherwise the limit is only suggested and the algorithm can accept also lower (or higher) values.

A common value for T is 8 Bytes, that is a complex value formed by two single-precision floating point values. However, also double-precision values are possible, raising T up to 16. In this case, we will assume single-precision, thus $T = 8$.

Common values for N depends on the experimental hardware, and in particular on the detector features. However, setting $N = 1024$ could provide a realistic value.

For what concerns R, that is the size of the population $\mathcal{P}$, there really isn't an usual value. It is usually chosen basing on the total memory available and/or the performances

of computing hardware. Obviously, the higher R, the better the performance of MPR. Let's set it to $R = 1024$, but we will see in the next chapters that its value can be lower or greater.

For the 2D case, i.e. $D = 2$, Eq.(2.10) provides a value of $W^{2\mathrm{D}} = 2 \times 1024^2 \times 8 \times 1024 \approx 17.2\mathrm{GB}$ for single-precision arrays. Even if some years ago this amount of memory was available only on specific machines, nowadays it is provided also by high-end laptops.

If we perform the same estimation for the three-dimensional case, assuming the same values for N, T and R, the result is three orders of magnitude higher, that is $W^{3\mathrm{D}} = 2 \times 1024^3 \times 8 \times 1024 \approx 17'600\mathrm{GB} = 17.6\mathrm{TB}$. Moreover, it's worth noting that the numerical complexity is dominated by the computation of the Fourier Transform. The Fourier Transform, in fact, is provided by the famous algorithm of Cooley & Tukey (1965), addressed as Fast Fourier Transform. Its computational complexity, which is strictly bound to the required computing time, scales at least with the following law:

$$C = N^D \log(N^D) \tag{2.11}$$

Other operations, which are mainly multiplications and additions inside loops on the coordinates, scale instead linearly with N^D. The ratio between the computational complexity of the 3D case C^{3D} and C^{2D} is equivalent to $\frac{C^{3D}}{C^{2D}} = \frac{3}{2}N$. If, as previously assumed, $N = 1024$, the computing time for the 3D case is about 1500 times more than the 2D one.

After this digression, it comes clear that this approach has an essential need of high performance computing hardware. Nowadays High Performance Computing (HPC) facilities can fulfill such requirements, but the implementation of MPR must be able to fully exploit those resources, which are distributed memory architectures with many level of computing parallelism.

2.3 Parallel implementation for High Performance Computing

Nowadays High Performance Computing hardware underwent many revolutions from the last years of the 20^{th} century until now. The first stage of its evolution was focused on the increasing of the computing speed of a single computing core (the so-called *clock frequency*). Then, the frequency of this hardware arrived near to its physical limit, or, saying it better, the limit at which the ratio between power consumption and performance was still acceptable. At this point the evolution of the hardware focused on distributed systems, where many computing cores, with their own memory, were interconnected by networks, opening the era of *computer clusters*. Also programming paradigms evolved in order to fully exploit those architectures, and those years saw the birth of the Message Passing Interface (MPI) protocol (Gropp et al. 1999), which still represents the main tool for parallel programming on distributed memory systems. Starting from the first years of this century, thanks to the increasingly miniaturization of transistors, the possibility to place more than one computing core on a single chip became true, giving life to the so-called *multi-core* systems. These system have the peculiarity that every computing core can access to the same memory address space: for this reason, multi-core processors follow the *shared-memory* paradigm. To fully exploit those systems and their strengths, many programming paradigms were born, and among them the OpenMP library (Dagum & Menon 1998). A third step was made in recent years (even if it was yet introduced at the dawn of digital computers era in 1960s), when the possibility to shrink the transistor size was approaching the physical limits. This last step involved the so-called *instruction level parallelism*, with the Single Instruction Multiple Data (SIMD) paradigm (Hennessy

& Patterson 2011). This parallelism is performed by introducing a vector computing unit, able to perform the same instruction on different inputs at the same time.

These three levels of parallelism, which can be addressed as *coarse-grained*, *fine-grained* and *instruction-level*, are not exclusive, such that modern HPC hardware provides all of them. Thus, anyone who wants to fully exploit this hardware must be aware of this concepts, and must know the necessary tools to get the maximum performance from it.

To sum up and clarify, modern HPC hardware commonly provides:

- N^{nodes} computing nodes, that are single stand-alone computing systems, with their own computing unit and their own memory, interconnected by means of a network. They represent the *coarse-grained* parallelism. A computing node cannot directly access to the memory of an other, and the only way is to broadcast data over the network.

- N^{cores} computing cores on each computing node. They represents the *fine-grained* level of parallelism. These cores has the access to the same memory, such that a core can read data written by an other one.

For what concerns instruction level parallelism, it is a tricky point which, to a first approximation, doesn't influence the algorithm design.

Let's now start to talk about the topic of this section, the MPR parallel implementation (Colombo et al. 2018). MPR is implemented in C++, and exploits the MPI library for distributed-memory parallelism and OpenMP library for shared memory parallelism. MPI requires that, on each computing node, a copy of the program is launched, such that N^{node} copies of the program are simultaneously run. Every copy of the program is identified by its rank $n = 0, ..., N^{\text{node}} - 1$. The operations on every computing node are, moreover, parallelized thanks to the *threads* managed by OpenMP, which are able to exploit all of the N^{cores} computing cores on each node.

As noted at the end of the previous section, especially for the three dimensional case, the memory occupancy of MPR, which is roughly the size of the array $\mathcal{P}$, can be in the order of hundreds of GigaBytes. Single computing nodes available nowadays can provide, at most, some tens of GBs. Thus, the first point is that the whole population $\mathcal{P}$ cannot be stored on a single computing node, but must be scattered among the N^{nodes} nodes. In this way, every instance of the program will store just a fraction of the population, that is:

$$\mathcal{P} \leftarrow \textsc{Individual}[R^{\text{local}}] \tag{2.12}$$

where $R^{\text{local}} = \frac{R}{N^{\text{nodes}}}$ is the dimension of the population fraction assigned to each node.

Fig.2.7 summarizes the *coarse-grained* parallelization of MPR implementation. MPR steps can be divided into *non-communicating* and *communicating* operations, basing on the fact the there aren't or there are communications among the team of MPI processes.

2.3.1 Data initialization and non-communicating operations

Before data initialization, the experimental data M_{ij} and the support function S_{ij} are read by the master MPI process (denoted here by the rank $\mathtt{r} = 0$), along with the parameters needed by the algorithm. The diffraction pattern and the support function are sent, together with the algorithm parameters, to other nodes via the `MPI_Bcast` function. This mean that the following lines should be inserted in Alg.13 after the function ReadData and before the call of Initialization.

The Mutation and Self-Improvement steps (along with the function SomeOtherStuff not displayed it Fig.2.7) do not require communications among nodes. Thus, their

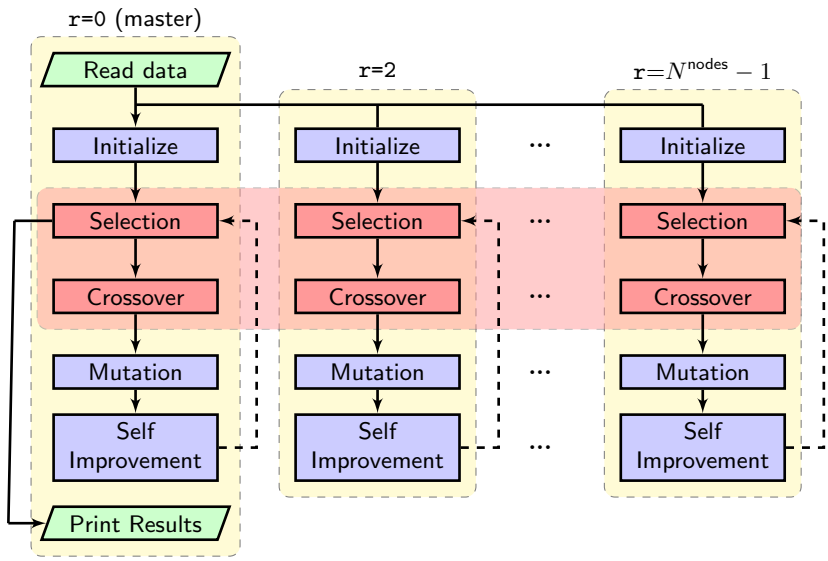


Figure 2.7: MPR *coarse-grained* parallelism. Yellow boxes indicate different MPI processes with ranks $\mathtt{r} = 0, ..., N^{\text{nodes}} - 1$. Steps requiring I/O operations are green colored. Steps not requiring MPI communications between nodes are blue colored. Steps surrounded by the red rectangle involve, instead, communication among nodes, and represent the real point of interest of this MPR implementation. Each of these steps have an *intra-node* parallelization via OpenMP.

Algorithm 17 The broadcasting of parameters

```
1: MPI_Bcast(M,0)      ▷ Send to all nodes the matrix M from process with rank 0 (master)
2: MPI_Bcast(S,0)      ▷ Send to all nodes the matrix S from process with rank 0 (master)
3: MPI_Bcast(C_RF,0)   ▷ Send to all nodes the parameters...
4: MPI_Bcast(C_ML,0)
5: MPI_Bcast(...,0)
```

implementation remains the same as the one shown before. The only point that should be noted is that the loops that previously run over R, now run up to R^{local}.

2.3.2 Communicating operations

The Selection and Crossover steps, which require communications among nodes, represent the key points of this parallel implementation. The most trivial way to perform those steps could be to gather the whole population $\mathcal{P}$ on the master node, perform Selection and Crossover, and then scatter it again on the N^{node} nodes. As previously said, this isn't a viable way, due to the high amount of memory required to store the whole population $\mathcal{P}$ which is much bigger than the one provided by a single computing node.

Thus, first of all the Selection operator cannot work directly on the population $\mathcal{P}$, but will act on an array of references defined below:

Every node will declare two arrays of this kind, i.e.:

Each element of these arrays contains an unambiguous reference to an individual of the population, by providing, along with its fitness value Reference.F, the rank of the node where it is stored, Reference.$Rank$, and its index Reference.$Index$ in the local population $\mathcal{P}$ on that node. For example, the individual 32 on node with rank 8 will be

Algorithm 18 Implementation of the reference type

```
1: Class Reference
2:     F                                   ▷ The assigned fitness value
3:     Rank
4:     Index
5: EndClass
```

Algorithm 19 The broadcasting of parameters

```
1: Ref^local ← Reference[R^local]
2: Ref^global ← Reference[R]
```

identified by a reference with $Rank = 8$ and $Index = 32$.

The parallel Selection

In this way, it is possible to sort the individuals in the population without directly sorting the array $\mathcal{P}$, but working on the much lighter reference array. For this reason, the Selection algorithm (Alg.6) has to be modified, as depicted by its new version in Alg.20.

Algorithm 20 The parallel implementation of the Selection step

```
1: function Selection(P)
2:     for r ← 0 to R^local do
3:         P[r] . E ← GetError(P[r] . ρ)
4:         P[r] . F ← 1/P[r] . E
5:         Ref^local[r] . F ← P[r] . F
6:         Ref^local[r] . Index ← r
7:         Ref^local[r] . F ← MPI_GetRank
8:     end for
       MPI_AllGather(Ref^local, Ref^global)
9:     Ref^global ← Sort(Ref^global)
10:    return Ref^global
11: end function
```

When looping on the local populatin $\mathcal{P}$, also the reference array $\text{Ref}^{\text{local}}$ is filled. Then, the function MPI_AllGather is called, which simply collects the single $\text{Ref}^{\text{local}}$ from each node in the global one $\text{Ref}^{\text{global}}$. Its mechanism is depicted by Fig.2.8.

When all the nodes have stored the global reference array $\text{Ref}^{\text{global}}$, they all call the Sort function, which now sorts this reference array basing on the value assigned to $\text{Ref}^{\text{global}}[r] \,.\, F$, that is the fitness value. The Selection function ends returning the sorted array $\text{Ref}^{\text{global}}$, where $\text{Ref}^{\text{global}}[r] \,.\, F \geq \text{Ref}^{\text{global}}[r+1] \,.\, F$.

The parallel Crossover

The parallel implementation of the Crossover operator is the most delicate one. In fact, when generating the new population $\mathcal{P}^{\text{sons}}$, the nodes cannot access directly to the whole population $\mathcal{P}$. Thus, the Roulette function defined by Alg.7 cannot directly return the density stored at the extracted index, but first of all it has to retrieve the array from the node that hosts that density.

This retrieval is performed exploiting a peculiar class of MPI calls, called Remote Memory Access (RMA)(Hoefler et al. 2015), introduced in the MPI-2 standard and extended in MPI-3. RMA communications are also called *one-sided* communications, in order to distinguish them from the standard *two-sided* ones.

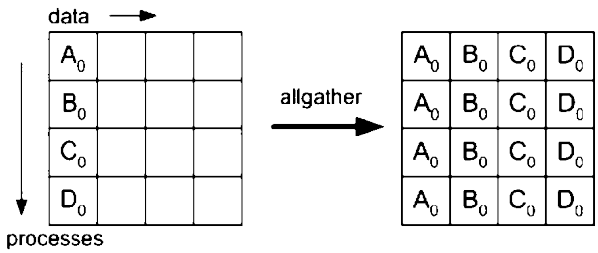


Figure 2.8: A graphical description of a MPI_AllGather call.

In a standard *two-sided* communication, both sender and receiver processes have to participate in data exchange operations explicitly, which requires synchronization between the processes. This means that, if the process A wants to receive some data from process B, A has to explicitly call the MPI_Recv and B has to call MPI_Send. The main issue is that, in this framework, not only A must know what ask to B, but it is mandatory that B knows *a-priori* what A will ask. Thus, when every node is performing the crossover step, all of them has to know which densities will be required by the others. Even if this approach is possible, it is highly inelegant and inefficient, due to the strong synchronization among nodes required by this method.

With *one-sided* communications, only the process A (the receiver) is involved, while B (the remote one) can continue to compute instead of explicitly sending data. Every process have to *expose* a defined block of memory, called *window*, in order to make it accessible by other processes with a RMA communication. In this way, a distributed memory system can be treated like in a shared memory paradigm. Advantages of RMA calls resides in their non-blocking nature, lowering the need of explicit synchronizations among processes and boosting the efficiency of the code. The use of RMA calls also produces a less intricate, easy to read source code[3].

Under the implementation point of view, the Crossover function requires also the reference vector $\text{Ref}^{\text{global}}$, which will be passed to the parallel version of the Roulette, listed below:

Algorithm 21 The parallel version of the Rigged Roulette

```
1: function Roulette(P, Ref^global)
2:     ind ← ⌊ Rand(0,1)^{C_R} * (R − 1)⌋
3:     ρ ← MPI_Get(Ref^global[ind] . Rank, Ref^global[ind] . Index)
4:     return ρ
5: end function
```

The third line of this algorithm, where the MPI_Get call appears, can be translated as "get the density with local index $\text{Ref}^{\text{global}}[\text{ind}] \,.\, Index$ from the node with rank $\text{Ref}^{\text{global}}[\text{ind}] \,.\, Rank$ and assign it to the matrix ρ".

Final remarks on the parallel implementation

The details on the implementation shown so far are clearly focused on what we have called *coarse-grained* parallelism, which exploits the MPI library. The calls to MPI routines,

[3]Details of Remote Memory Access MPI routines are not discussed here, but can be found at `www.mpi-forum.org`

during our digression, have been simplified by only providing the arguments useful for the implementation comprehension. In the concrete C++ implementation, those functions requires more arguments that in this description were neglected for sake of simplicity.

Moreover, details on *fine-grained* and *instruction-level* parallelism have been left out. These features do not impact on the implementation structure and can be easily summarized with the following general observations.

- *Fine-grained* parallelism, based on *threads*, has been performed on the loops that run on the population index r. This means that if an operation has to be repeated for all the individuals in the local population $\mathcal{P}$, the operations on different indexes r are performed in parallel. If, for example, the size of $\mathcal{P}$ is 16 and the available computing cores on a given node are 8, this means that the operations on the first 8 are done in parallel (one for each computing core), and then the same is performed for the next 8 individuals of the array $\mathcal{P}$[4]. Moreover, this *thread-level* parallelism is performed also for the computation of the Fast Fourier Transform[5], if a sufficient number of computing threads is available.

- *Instruction-level* parallelism, based on vector operations, is performed, instead, on the loops that cycle on the coordinates, i.e. those loops in the form **for** $i, j \leftarrow N$ **do**. This parallelism is commonly provided, unless some exceptions, by simply enabling the right flag at compilation time. For those cases where the compiler fails, the SIMD parallelism can be enforced thanks to OpenMP 4.0 SIMD directives[6].

2.4 A quick glance to performance

Although this section goes beyond what a PhD thesis in physics should be, it's mandatory to evaluate a bit the scaling performances of the code, just because it has been conceived to run on HPC hardware. Due to the "nerdy" nature of this section, a reader not interested in the topic can, without doubts, jump this part because it doesn't compromise the comprehension of the work.

Tests are performed on Marconi[7] cluster at CINECA[8] and provides 3600 computing nodes equipped with an Intel® Xeon Phi 7250 Knights Landing (KNL) CPU (Sodani 2015) each, which means 68 cores per node able to handle up to 272 computing threads, thanks to the Intel's Hyper-Threading technology. The whole cluster is connected via the Intel OmniPath Architecture. Scaling performances on both shared and distributed memory have been investigated on the KNL hardware of Marconi.

2.4.1 Shared memory scaling

These tests investigate the scaling performance on a single computing node, involving only the *intra-node* parallelization managed by OpenMP.

The first test, shown on Fig.2.9 left, involves the algorithm scaling on a single core, among the 68 ones of the Xeon Phi, by measuring the *speedup* obtained thanks to the Hyper-Threading technology. Here *strong-scaling* is considered, which means that the total workload is kept constant and it is distributed among a different amount of threads.

[4]For more information, please visit `www.openmp.org`

[5]For more information, please visit `http://www.fftw.org/`

[6]These directives are not discussed here.
If curious, you can have a look here: `https://www.openmp.org/resources/tutorials-articles/`

[7] For further information visit `www.hpc.cineca.it/hardware/marconi`

[8]Consorzio Interuniversitario del Nord-Est per il Calcolo Automatico (CINECA), Italy

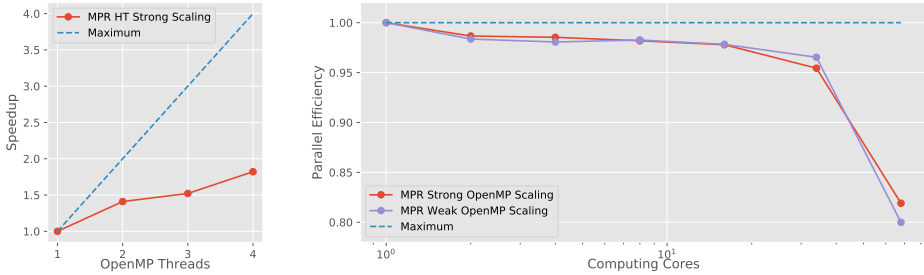


Figure 2.9: On the left: the *speedup* due to the Hyper-Threading technology on a single core, by exploiting up to 4 OpenMP threads. On the right: the *parallel efficiency* on the whole KNL node, as function of the exploited cores, from 1 (4 threads) to 68 (272 threads).

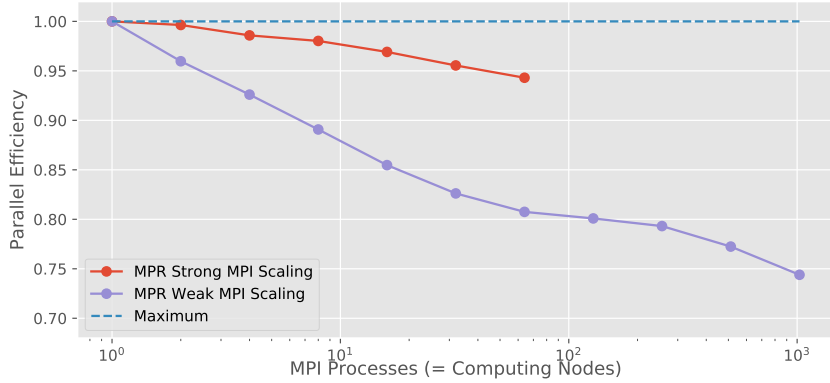


Figure 2.10: Scaling performance on distributed memory. Every computing node is equivalent to one MPI process, because shared-memory parallelization inside every node is entrusted to OpenMP threads.

Given t_T the measured execution time by exploiting T threads, the *speedup* value S has been calculated via the formula $S = \frac{t_1}{t_T}$. Every computing core of Xeon Phi is able to handle up to 4 threads.

The second test, shown on Fig.2.9 right, concerns the scaling on the whole computing node, from 4 threads, which means 1 core, up to 272 threads, that is 68 cores. Here, also *weak-scaling* is considered, which means that the workload per core is kept constant. Given t_C the execution time measured by exploiting C computing cores, the *parallel efficiency* value for *strong-scaling* E_{strong} has been calculated as $E_{\text{strong}} = \frac{t_1}{C \times t_C}$, while, for *weak-scaling*, $E_{\text{weak}} = \frac{t_1}{t_C}$.

2.4.2 Distributed memory scaling

This test concerns the algorithm scaling on multiple computing nodes, managed by MPI. For N computing nodes, exactly N MPI processes are required, due to the fact that, for every process, *intra-node* parallelization on shared memory is handled by OpenMP. In this test, 272 OpenMP threads are exploited for each computing node. Two different kinds of scaling are shown in Fig.2.10. The *parallel efficiency* for *strong-scaling* is given by $E_{\text{strong}} = \frac{t_1}{N \times t_N}$. This test has been performed from 1 computing node (272 OpenMP

threads) to 64 (for a total of 17408 threads). The *parallel efficiency* for *weak-scaling* is, instead, given by $E_{\text{weak}} = \frac{t_1}{t_N}$. This test has been performed from 1 up to 1000 computing nodes (for a total of 272000 OpenMP threads).

It is worth noting that most of the MPI communications in this implementation are point-to-point, and, in the case of *weak scaling*, their amount per computing node is asymptotic to a constant value, as the number of nodes increases.

The only step requiring MPI collective communications is the **Selection** step (see Alg.20). During this step, collective communications concern only the *reference* array Ref, with the MPI_AllGather function. The weight of MPI_Gather is expected to grow linearly with the total amount of data, which means, in the case of *weak-scaling*, with the number of computing cores. Every element of this array contains, as previously said, only 16 bytes of data (Ref.F, Ref.$Index$ and Ref.$Rank$). Its weight is, so, negligible for a low amount of computing nodes, compared to the amount of data exchanged during the **Crossover** step, while it becomes relevant with the increasing of computing nodes, as shown by the right side of Fig.2.10.

Moreover, the reader may be surprised to see a better performance of the *weak scaling*, if compared to the *strong scaling*. For most applications, the *weak scaling* is much better than strong scaling, due to the fact that the workload per computing unit is kept constant, without fragmenting the problem. On the other side, a good *strong scaling* is often hard to get, due to the fact that it is computed by keeping the total workload constant, which leads to a fragmentation of the problem that increases with the number of computing units. However, for MPR the situation is peculiar, because a non-negligible part of the computation is spent in sending and receiving data among computing nodes. Thus, when *weak scaling* is computed, the total amount of communications per computing unit remains constant, while, for *strong scaling*, the number of communications per computing node decreases, providing a better performance.

Chapter 3

MPR Characterization

This chapter is dedicated to the analysis of the MPR performance as function of its parameters and of the type of the treated data. In order to provide a fully controlled characterization of MPR behavior, in this chapter only simulated data will be treated.

Setting up a test on simulated data is not so simple as it may appear. Given a synthetic solution ρ^{sol}, a simulation of the scattering is, in principle, simply provided by $M = \sqrt{I}$ where $I = |\mathsf{DFT}[\rho^{\text{sol}}]|^2$. The truth is that some issues deviate the experimental diffraction pattern from the ideal one:

- **Noise** - Experimental noise not only may derive from the thermal noise of the detector, but it is intrinsic of the measurement process. If the first can be assumed to be a white noise, equally distributed on the experimental data I, the latter derives from the fact the output values provided by the detector are actually counts, i.e. the number of photons fallen on the single pixels. An usual way to express the amount of noise of a dataset is the Signal to Noise Ratio (SNR).

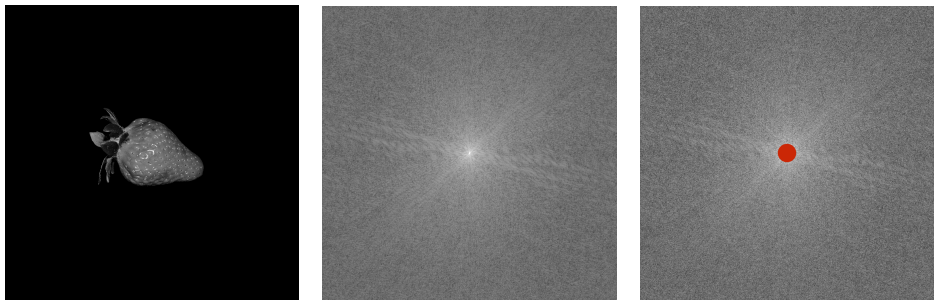
- **Saturation** - Detectors usually have a fixed *dynamic range*. This means that each detector pixel is able to count up to a certain number of photons. When the radiation hitting a pixel exceeds this saturation threshold, the recorded data on that pixel becomes meaningless, and thus it can't be used.

- **Beam stopper** - When high energy radiation is exploited (mainly X-rays), the amount of radiation that arrives on the central part of the detector is so much that it could cause severe damages to the (really expensive) detectors. For this reason, a so-called *beam stopper* is placed in the central part of the detector, in order to block the radiation "dangerous" for the detector.

A *realistic* simulated data, thus, should provide a reasonable level of noise and the lack of experimental data in the central part of the pattern, as depicted in Fig.3.1.

It's clear now that a *definitive* test on simulated data cannot be designed, because the issue concerning experimental data depends a lot not only on the experimental setup (in particular the detector features and the exposure time), but also on the sample of interest. The latter not only influences, with its size, the oversampling degree of the acquired data, but different shapes and features of a sample make the phase retrieval process more or less difficult. It is possible, however, to group datasets basing on their issues:

- **Data-limited datasets**: phases are tricky to retrieve due to lack of data. This lack has to be intended in a general meaning. The lack of data could be provided by a low oversampling degree O_d or/and by missing values in the central part of the pattern due to saturation or due to the presence of the beam-stopper. In these



(a) The solution **(b)** Ideal diffraction pattern **(c)** Realistic pattern

Figure 3.1: Given a sample (a), that represents the solution to the phase problem, (b) is its ideal Fourier module, while experimental diffraction data often looks like (c), which is characterized by noise and the presence of the beamstopper that hides low resolution information.

cases, phasing algorithms struggle to find the solution due to loose constraints: this features make the the error functional E particularly smooth and full of local optima. Thus, iterative projection algorithms, that depend on the gradient, lack what we have defined as *ergodicity*: in fact, the low value of the gradient, along with a space full of local minima, prevent them to well explore the space of configurations.

- **Noise-limited datasets**: in this case, instead, the issue is at the opposite side. Noise in the diffraction pattern moves away the two sets, $\mathcal{M}$ and $\mathcal{S}$, by a modification of the shape of the set $\mathcal{M}$. In this way, making the two constraints more incompatible, all of the algorithms based on *feedbacks* and *reflections* (at the end of the story, all but the Error Reduction) become particularly unstable, i.e. they are no longer able to converge: they lack of *stability*.

An exhaustive set of synthetic datasets should cover all the possible issues that an experimental measurement may have, that is various combination of noise and data lack, but also different sample shapes and oversampling degree. Moreover, an exhaustive test should be performed by considering also all the combinations of *iterative projection algorithms*, like RAAR, ASR, HIO, HPR etc. Testing all of these approaches in all the possible situations would be too computationally expensive, too time wasting and really too boring.

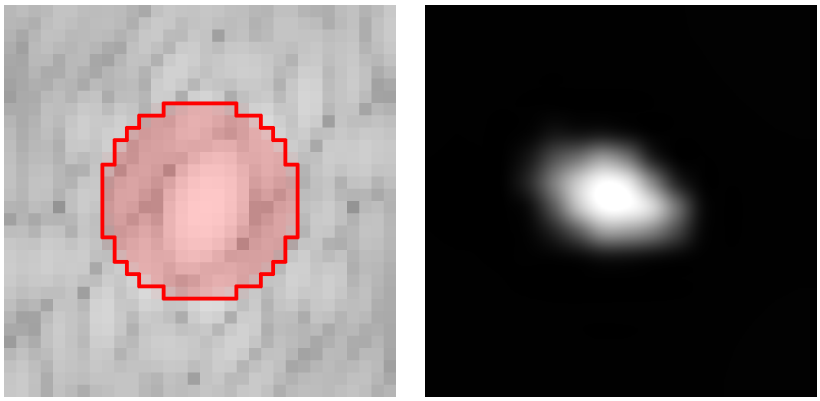
Thus, only some examples will be provided. These simulated datasets will be characterized by three different kinds of issues: **noise**, **lack of diffraction data** and **loose support function**.

As previously said, it's hard to give a generic shape to the **noise**, because it depends also on the features of the acquisition equipment. Here we follow the method suggested by Miao et al. (1998). For a given diffraction pattern I_ij, a noise N_ij has been added by following this equation:

$$N_{i,j} = \frac{I_{i,j}}{SNR} \times \text{Rand}(-0.5, 0.5) \tag{3.1}$$

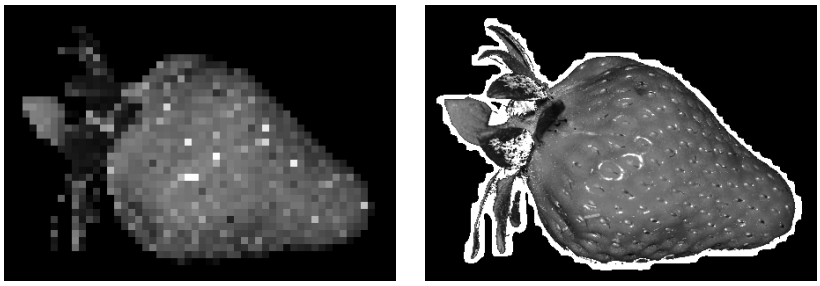
where SNR is the Signal-to-Noise ratio, while $\text{Rand}(-0.5, 0.5)$ represents a random number with a flat distribution in the interval $[-0.5, 0.5)$.

The **lack of diffraction data** concerns, instead, missing values in the central part of the diffraction pattern (Fig.3.1b), simulating the presence of a beam stopper. Those



(a) **(b)**

Figure 3.2: A detail of the simulated diffraction pattern (a): red pixels have been removed in order to simulate the presence of the beam stopper. In (b), the Fourier Transform of the red pixels of (a), that provides an idea of the low resolution information lost due to the beam stopper.



(a) **(b)**

Figure 3.3: A the low resolution version of the solution, from which the support function has been computed (right image).

pixels are displayed in Fig.3.2: the lost pixels contain a lot of information about the sample shape, as far as the whole central peak of the Fourier Transform is lost. Fig.3.2b is an inverse FT of the excluded pixels, and it clearly shows the low-resolution distribution of the sample.

Concerning the **loose support issue**, it derives from the fact that the support function is usually extracted from a low-resolution image of the sample of interest. In order to simulate this problem, the support function is computed by applying a threshold to a low resolution image of the solution (in our test it will be eight times worse), displayed in Fig.3.3a. Fig.3.3b shows instead the solution superimposed to the support function extracted from Fig.3.3a, just to show that this support in lager than the actual extension of the sample. To solve this loose support issue it is convenient to use the *Shrink-wrap* algorithm, which is able to adjust the constraint $\mathcal{S}$. Due to this great feature, it is difficult to compare different phase retrieval algorithms combined with the *Shrink-wrap*, thus it won't be exploited for all the following tests, while a section will be fully dedicated to the *Shrink-wrap* approach.

Before venturing into MPR performance analysis on simulated datasets with different

amounts of noise, lack of data and more or less loose supports, a digression on the meaning of MPR parameters will be presented. As seen in the previous chapter, a MPR phasing procedure depends on many parameters, which mainly involve the *genetic* operations of *Selection*, *Crossover* and *Mutation*.

3.0.1 Evaluation of the performance

It is worth defining in which way the quality of a reconstruction is evaluated, in order to provide the most objective and quantitative description possible. Along a qualitative interpretation of the result by the scientist, which is always needed, a quantitative measurement of the reconstruction reliability is given by the error value. Where shown, the error of a reconstruction ρ will be computed via Eq.(1.20), but its value is normalized in a different way. Thus, the error value $\mathcal{E}[\rho]$ is computed in the following way:

$$\mathcal{E}[\rho] = \sqrt{\frac{E[\rho]}{\sum_{i,j} I_{i,j}}} = \sqrt{\frac{\sum_{i,j}(M_{i,j} - |\tilde{\rho}_{i,j}|)^2}{\sum_{i,j} M_{i,j}^2}} \tag{3.2}$$

where I is the acquired diffraction pattern and $M_{i,j} = \sqrt{I_{i,j}}$. The advantage of this definition resides in the fact that its value is almost always in the range $0 < \mathcal{E} < 1$. Its worth noting that $E[\rho^1] < E[\rho^2]$ implies that $\mathcal{E}[\rho^1] < \mathcal{E}[\rho^2]$.

Even if the above mentioned error value is the tool par excellence of the evaluation of a reconstruction quality, it actually represents just an evaluation of the global quality of a retrieved density, without giving any information about the the achieved resolution.

A very common way to perform this more punctual evaluation is represented by the Phase Retrieval Transfer Function (PRTF) (Chapman et al. 2006). For this evaluation, an average reconstruction ρ^{aver} of N independent reconstructions (i.e., started from random phases) is considered. Thus, the PRTF function is provided by:

$$\text{PRTF}(k_0) = \frac{\sum_{|\vec{k}|=k_0} \frac{|\tilde{\rho}^{\text{aver}}(\vec{k})|}{M(\vec{k})}}{\sum_{|\vec{k}|=k_0} 1} \tag{3.3}$$

where $M = \sqrt{I}$ are the experimentally measured modules, and the points where $M = 0$, or modules are unknown, are excluded from the computation.

The PRTF provides an estimation of the reached resolution. In fact, when all the reconstructions have similar phases, their average tends to have a module near to the experimental one, such that $\text{PRTF}(k_0)$ will tend to 1. When, instead, those reconstructions differ in the phase value, $\text{PRTF}(k_0)$ will tend to 0. A reconstruction is usually considered reliable as long as $\text{PRTF}(k_0) > 0.5$. The point where the PRTF goes below 0.5 is considered as the maximum achieved resolution.

The use of the PRTF to evaluate a MPR result is not straightforward. A first possibility could be to evaluate the PRTF using a ρ^{aver} computed on the elements inside the population $\mathcal{P}$. The problem is that those reconstructions are not independent because they are strictly correlated by the genetic *Crossover* operator: thus, computing of the PRTF on the MPR average density is unfair and misleading.

A second way, which is surely the most appropriate, is to perform N MPR reconstructions, where, for each of them, a population of densities is randomly initialized. The average value to compute the PRTF is no more the "inner" average, but the average of the N ρ^{best} obtained by each MPR reconstruction. In this way, the total independence of the starting point is ensured and the PRTF analysis makes sense.

Performing in the latter way the PRTF analysis on a MPR reconstruction is, however, too computationally demanding, due to the fact that a single MPR phase retrieval already involves a set of R different reconstructions. Moreover, we can state that, actually, the PRTF analysis does not directly says the reliability of a reconstruction, but it measures how much the final result of a given algorithm is dependent on the starting guess. In fact, a PRTF that never goes below the threshold value of 0.5 doesn't actually ensure that the resolution of a reconstruction has been maximized, but only that the reconstructions, from which the average for the PRTF has been computed, have converged near to a common optimum of the error functional.

In order to provide a more punctual description of the reconstruction quality basing only on a single phase retrieval result, a new estimator can be introduced, which has a formulation similar to the PRTF. Thus, the Error Profile Function (EPF) is defined:

$$\mathrm{EPF}(k_0) = \frac{\sum_{|\vec{k}|=k_0} \left| \left| \mathcal{F}\left[P_{\mathcal{S}}\rho^{\mathrm{best}}\right](\vec{k}) \right| - M(\vec{k}) \right|}{\sum_{|\vec{k}|=k_0} M(\vec{k})} \tag{3.4}$$

The meaning of $\mathrm{EPF}(k_0)$ is easy to interpret: it represents the average discrepancy between the retrieved amplitudes, computed on the reconstruction ρ^{best} on which the support projector has been applied, and the experimental ones. This EPF function, unlike the PRTF, cannot be interpreted in an absolute sense. In fact, the EPF value highly depends on:

- Noise in diffraction data: if the SNR of diffraction data is low, the EPF value will be high
- Support function: if the support function is larger than the correct one, also a wrong result can provide a good EPF.

Thus, the EPF is much less powerful than the PRTF and cannot be exploited to determine the maximum resolution reached by a phase retrieval run. The EPF turns out to be useful only to compare different reconstructions with the same, or similar, support functions, in order to determine which of them has gained the best resolution.

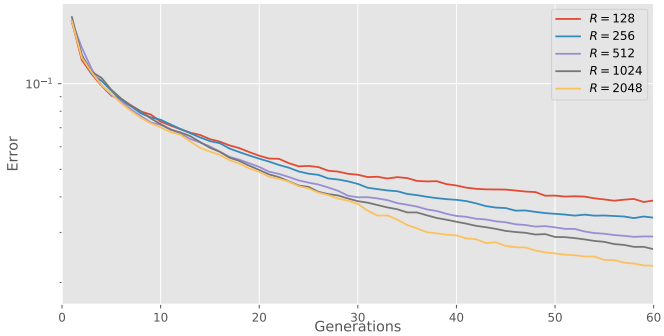
The next section will be dedicated to the characterization of MPR behavior with respect to different values for its parameters.

3.1 Algorithm tuning

The tricky feature of MPR is surely its dependence on many parameters, which may discourage the user. On the other side, this feature makes MPR particularly versatile, i.e. able to adapt to different situations.

A first differentiation of MPR parameters, listed in Table 2.6, can be done basing on their computational cost. In fact, it's easy to see that the only parameters that influence the computation time are R, that is the population size, and N_{alg}, that are the number of iterations performed during the Self-improvement step. Other parameters are practically cost-free, such that the user can freely tune them without any worries about computing hardware limitations.

The following subsections will investigate, case by case, different values for those parameters. The provided results are based on a simulated diffraction pattern, which present noise (Signal-to-Noise Ratio equal to 10), lack of data (as depicted by Fig.3.2) and a support function extracted from a low-resolution image of the simulated sample



(a)

Name	Value
R	–
C_{RF}	1
C_R	1.8
C_{GB}	0.5
C_D	1
F_{FC}	*true*
C_{MP}	0.2
C_{MF}	0.66
C_{ML}	0.66
N_{HIO}	40
N_{ER}	20

(b)

(c) R=128 **(d)** R=256 **(e)** R=512

(f) R=1024 **(g)** R=2048

Figure 3.4: MPR results as function of the population size. In (a), the error behavior (in logarithmic scale) as function of the number of generations for different population sizes. MPR parameters for this test are listed in table (b). Results for increasing values of R are depicted, from (c) to (g).

(Fig.3.3). Data will be considered real-valued, such that the reality of ρ has been added to the constraints.

3.1.1 The population size

The first parameter analyzed in this discussion is the size of the population $\mathcal{P}$, which is ruled by the value R, i.e. the number of individuals inside the population. The main issue regarding this parameter is that different values of R imply different computational costs, concerning both the computing time and the memory occupancy. If the first just implies more time to wait for the end of computation, the second is actually the true limiting factor, because the total memory used by MPR must be less than the total memory installed on the computing hardware.

Fig.3.4 depicts difference performances of MPR as function of R. As expected, the greater R, the better the result. This is underlined not only by the plot in Fig.3.4a, but also by the visual comparison provided by the figures 3.4c, 3.4d, 3.4e, 3.4f and 3.4g.

Just as example, Fig.3.5 depicts the EPF function for the results depicted in Fig.3.4. As said before, the EPF function has only a relative meaning, as like as the error function plotted in Fig.3.4a. This example clearly shows the information that the EPF function

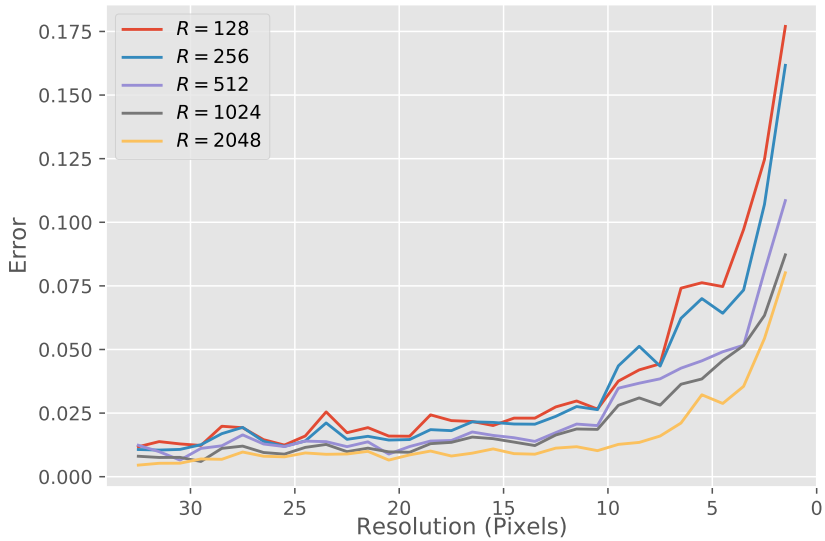
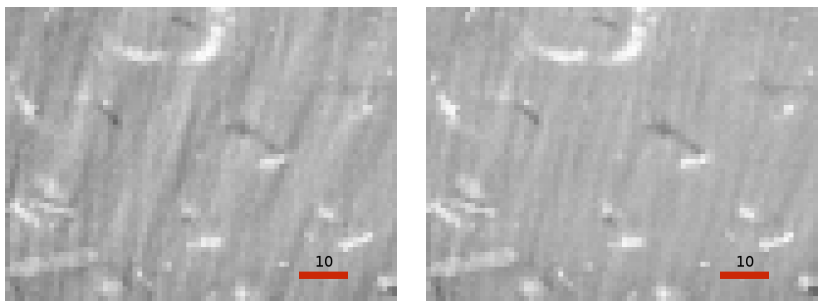


Figure 3.5: The EPF value as function of the resolution in real space, plotted for different values of the population size R.

can provide: in particular, the EPF value well reflects the entity of artifacts in the reconstructions. First of all, all but the $R = 2048$ reconstructions have a strong increasing of the EPF value around a $|\vec{k}|$ equivalent to about 8-10 pixels in direct space, while, in this zone, the $R = 2048$ remains flat. This behavior derives from the presence of vertical stripes with a characteristic width of that size in all the other reconstruction, which are absent in the $R = 2048$ one.

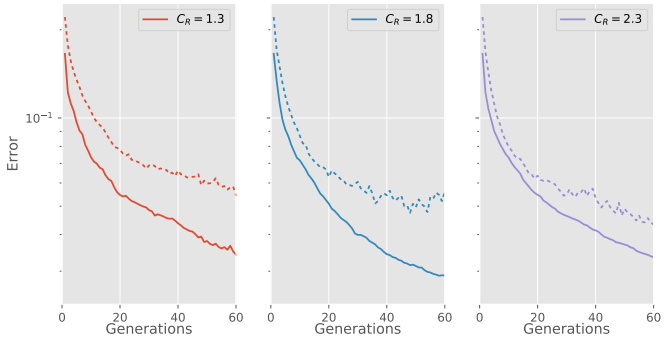
However, looking again at the EPF of $R = 2048$, there is an increase of its value around the resolution of 5 pixels, which corresponds to the emerging of artifacts of that size. These artifacts are well visible in Fig.3.6.

In this example, the dimension of data is 1024×1024, such that each individual, has a size of $1024^2 \times 8$ Bytes. Assuming single precision floating point values, Eq.(2.10) gives thus a memory occupancy of about 2.15GB for $R = 128$ up to nearly 35GB for $R = 2048$.



(a) **(b)**

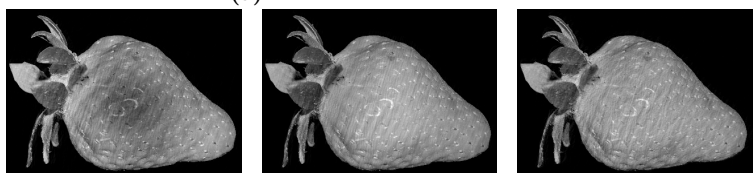
Figure 3.6: Detail of the $R = 512$ (a) and $R = 2048$ (b) reconstructions. Stripes of about 10 pixels size are clearly visible in (a), while artifacts in (b) are at a higher resolution.



(a)

Name	Value
R	512
C_{RF}	1
C_R	–
C_{GB}	0.5
C_D	1
F_{FC}	*true*
C_{MP}	0.2
C_{MF}	0.66
C_{ML}	0.66
N_{HIO}	40
N_{ER}	20

(b)



(c) $C_R = 1.3$ **(d)** $C_R = 1.8$ **(e)** $C_R = 2.3$

Figure 3.7: MPR results as function of the *roulette* coefficient C_R. In (a), the error behavior (in logarithmic scale) as function of the number of generations for different C_R values. MPR parameters for this test are listed in table (b). Results for increasing values of C_R are depicted from (c) to (e). Dashed lines in (a) depicts the error value of the element with index $\frac{R}{2}$ in the sorted array. This dashed line provides an idea about the differences in error values among the individuals. The greater the distance between the continuous line (i.e. the error of the best individual) and the dashed one, the more spread the population.

$R = 512$ needs about 8.6GB of memory, which more or less represents the limit of memory installed on a middle-high end personal computer nowadays.

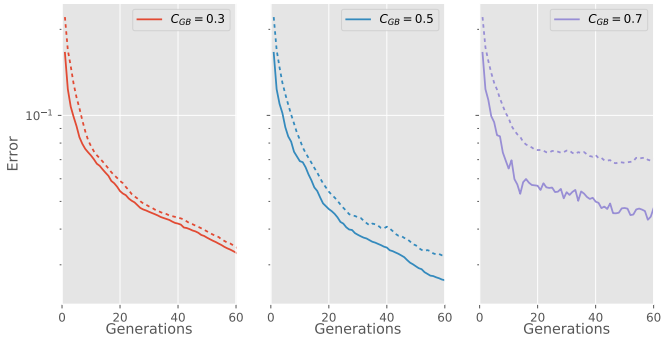
For this reason, the following tests concerning parameters values are performed on a population of $R = 512$ elements, being, however, aware that a higher value of R will provide a better result.

3.1.2 The roulette coefficient

The *roulette coefficient* C_R is the first of the four parameters involved the *Crossover* step. Just to remind, it modifies the probability to choose a "good" parent for the *Crossover* operation (see Alg.7). $C_R = 1$ means that every element in the population $\mathcal{P}$ has the same probability to be selected for reproduction, while the higher C_R, the more likely a parent with lower error will be chosen. Thus, a high value for C_R makes the previous *Selection* step "stronger", while a low value makes it "weaker".

Fig.3.7 shows the behavior of the algorithm for different values of C_R. Among the three chosen values (1.3, 1.8 and 2.3), the one which performs better is the middle one, as the plot in Fig.3.7a shows along with the visual representations in Fig.3.7c, 3.7d and 3.7e. In particular, the dashed lines in Fig.3.7a show the error value of the middle element in the population, i.e. the one with index $\frac{R}{2}$ in the sorted population (in this case, $R = 512$, so the dashed line is the error of the element with index 256). This information provides an idea of how much "spread" the population is.

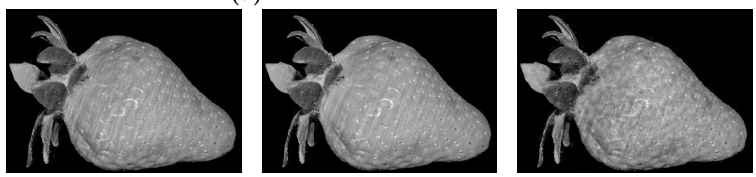
It's worth noting that, clearly, a value of C_R too near to 1 makes the *Selection* too



(a)

Name	Value
R	512
C_{RF}	1
C_R	1.8
C_{GB}	–
C_D	1
F_{FC}	*true*
C_{MP}	0.33
C_{MF}	0.66
C_{ML}	0.66
N_{HIO}	40
N_{ER}	20

(b)



(c) $C_{GB} = 0.3$ **(d)** $C_{GB} = 0.5$ **(e)** $C_{GB} = 0.7$

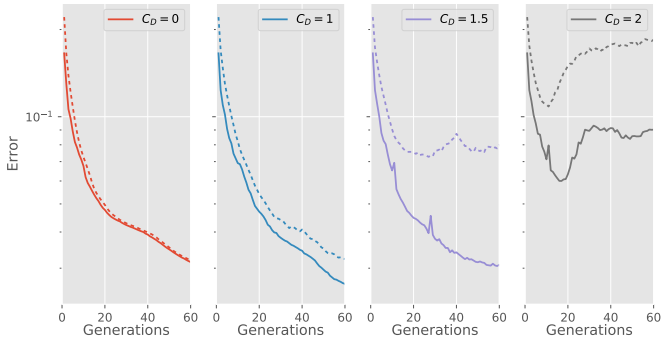
Figure 3.8: MPR results as function of the *genetic balance* coefficient C_{GB}. In (a), the error behavior (in logarithmic scale) as function of the number of generations for different C_{GB} values. MPR parameters for this test are listed in table (b). Results for increasing values of C_{GB} are depicted from (c) to (e). It's worth noting the big differences in the error spread inside the population, by looking at the distance between the dashed and the continuous line. If C_{GB} is too low, individuals will be similar and the algorithm will tend to stagnate: conversely, if C_{GB} is too high, MPR will have an unstable behavior.

weak, such that there is no discrimination between "good" and "bad" individuals. On the other side, a too strong *Selection* shrinks the genetic pool too much, i.e. in few generations the individuals tend to resemble each other, making the algorithm stagnate. Speaking in terms of algorithm convergence properties, C_R near to 1 makes the algorithm *ergodic*, while high values of C_R make the algorithm *stable*. Thus, good values for C_R, in this case, run from 1.5 to 2, which represent a good compromise for preferring the "good" individuals without causing a stagnation of the algorithm.

3.1.3 The genetic balance coefficient

This parameter is the second one involved in the *Crossover* step. As described by Alg.8, it tunes how much of the genetic pool is inherited by the the first parent. In a "standard" *Crossover*, the effects of C_{GB} are symmetric with respect to 0.5, which means that half of the values are randomly taken from the first parent and half from the second. As said in the previous chapter, this implementation exploits instead a *Differential Crossover*, which means that the second parent is actually a combination of three different ones (see Alg.8). For this reason, C_{GB} has symmetric effects with respect to 0.5 anymore: with $C_{GB} < 0.5$ the most part of the genetic pool is inherited by the first parent, while with $C_{GB} > 0.5$ the most values are given by the combination of the second, the third and the fourth ones.

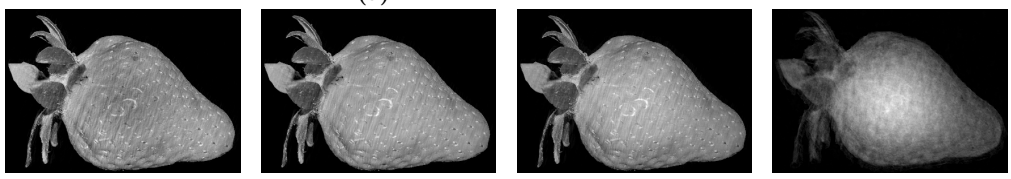
Fig.3.8 depicts the behavior of MPR depending on C_{GB}. Both the plot in Fig.3.8a and the results displayed in Fig.3.8c, 3.8d and 3.8e well show that the best results are



(a)

Name	Value
R	512
C_{RF}	1
C_R	1.8
C_{GB}	0.5
C_D	–
F_{FC}	*true*
C_{MP}	0.33
C_{MF}	0.66
C_{ML}	0.66
N_{HIO}	40
N_{ER}	20

(b)



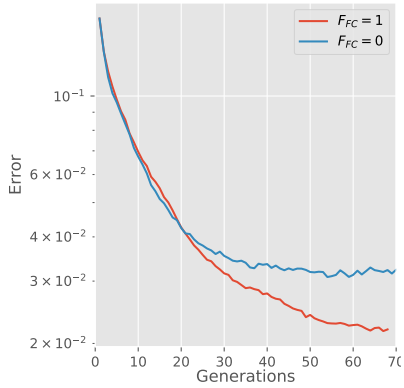
(c) $C_D = 0$ (d) $C_D = 1$ (e) $C_D = 1.5$ (f) $C_D = 2$

Figure 3.9: MPR results as function of the *differential* coefficient C_D. In (a), the error behavior (in logarithmic scale) as function of the number of generations for different C_D values. MPR parameters for this test are listed in table (b). Results for increasing values of C_{GB} are depicted from (c) to (e). As it was shown for the *genetic balance* coefficient, a too low value for C_D causes the algorithm stagnation, while a too high value leads to an unstable reconstruction.

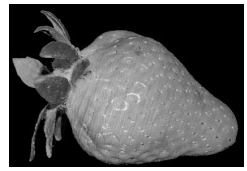
achieved with $C_{GB} = 0.5$ and $C_{GB} = 0.3$. The issue with the latter is that, as the dashed line representing the middle element suggests, the individuals of the population are very close to each other, providing a relatively low differentiation of the genetic pool. In this case, the issue could be solved by providing, for example, a stronger mutation step. The latter result with $C_{GB} = 0.7$ shows instead a too high differentiation of the genetic pool, deriving from the fact that about the 70% of the values are inherited from the combination of three different parents following a differential combination. Following these results and basing on experience, "good" values for this parameters are $0.3 \leq C_{GB} \leq 0.5$. Values near to (or less than) 0.3 privilege *stability*, while values close to (or greater than) 0.5 make the algorithm *ergodic*.

3.1.4 The differential coefficient

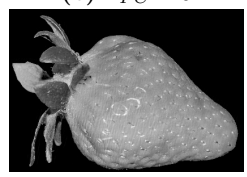
The differential coefficient C_D strongly influences the differential Crossover step. Its aim is to amplify the differences among the individuals, focusing on those coordinates whose value differs from an element to another. As stated in the previous chapter, C_D influences the way in which the second parent ρ is created, via the relation $\rho_{i,j} = \rho^{p2}_{i,j} + C_D(\rho^{p3}_{i,j} - \rho^{p4}_{i,j})$. If $\rho^{p3}_{i,j}$ and $\rho^{p3}_{i,j}$ have similar values at coordinate (i,j), $(\rho^{p3}_{i,j} - \rho^{p4}_{i,j})$ is small. If, otherwise, they differ a lot, the value of the second parent $\rho_{i,j}$ differs a lot from both $\rho^{p2}_{i,j}$, $\rho^{p3}_{i,j}$ and $\rho^{p4}_{i,j}$. This operation is designed following the idea that if the value at given coordinate (i,j) is almost the same among different individuals, it is likely to be well retrieved, while, otherwise, new values at that coordinate must be explored. C_D, thus, rules the weight of this operation. When $C_D = 0$, the *Crossover* assumes its standard form, loosing its



(a)



(b) $F_{FC} = 0$



(c) $F_{FC} = 1$

Name	Value
R	512
C_{RF}	1
C_R	1.8
C_{GB}	0.5
C_D	1
F_{FC}	–
C_{MP}	0.33
C_{MF}	0.66
C_{ML}	0.66
N_{HIO}	40
N_{ER}	20

(d)

Figure 3.10: MPR results as function of the *fourier crossover* flag F_{FC}. In (a), the error behavior as function of the number of generations for the two cases $F_{FC} = 0$ and $F_{FC} = 1$. In (b) and (c) the two results are depicted. MPR parameters for this test are listed in table (d).

differential feature.

Fig.3.9 depicts the results for different values of C_D. The first point is that too high values for C_D carry to a lack of convergence of the algorithm, as clearly shown by Fig.3.9a and by the result in Fig.3.9f. The best result is achieved by $C_D = 1$ (Fig.3.9d). It's worth noting that $C_D = 0$ and $C_D = 1.5$ reached similar results, equally worse than $C_D = 1$ but for very different reasons. As the dashed lines show, $C_D = 0$ is too *stable*, i.e. the population is too similar. The case $C_D = 1.5$, instead, provides too much *ergodicity*, as the distance between the error of the best and the error of the middle element (dashed line) suggests.

Thus, "good" values for the differential coefficient are $0 \leq C_D \leq 1.5$. Values near to 0 privileges *stability*, while *ergodicity* is enforced by values close to (or greater than) 1.

3.1.5 The Fourier crossover flag

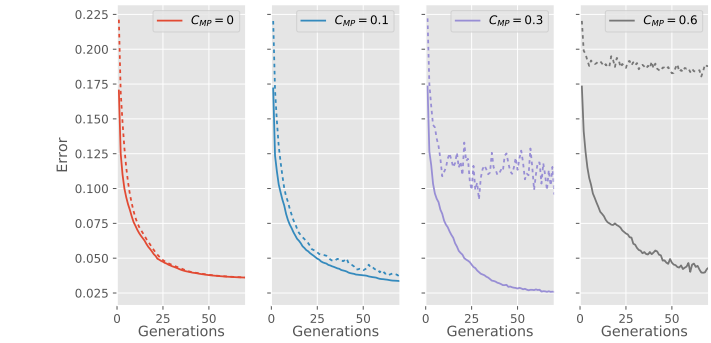
This parameter is the last one involved in the *Crossover* step. F_{FC} is, actually, a *flag*, which means that it can assume only two values, *true* or *false* (1 or 0). When $F_{FC} = 1$ the crossover operation is performed in the frequency domain, which means that the genetic pool is composed by the values of the Fourier Transform of the individuals. If, instead, $F_{FC} = 0$, the genes are considered to be the values of the density in the direct space.

Fig.3.10 describes the result for this two different cases. As well depicted both by the plot in Fig.3.10a and the Fig.3.10b and 3.10c, in this case a crossover operation performed in the frequency domain performs better. Even if this behavior is often observed, this is not a genera rule, and sometimes, depending on the treated dataset and on the other parameters, a real space crossover may appear more convenient. It's however worth noting that different values for F_{FC} carry different meanings for the *Crossover* step, mainly due to its differential implementation (the differential Crossover treated before).

3.1.6 The mutation probability

After having characterized the MPR behavior as function of the parameters involved in the *Crossover* step, it's now time to have a look at the ones involved in the *Mutation*. The

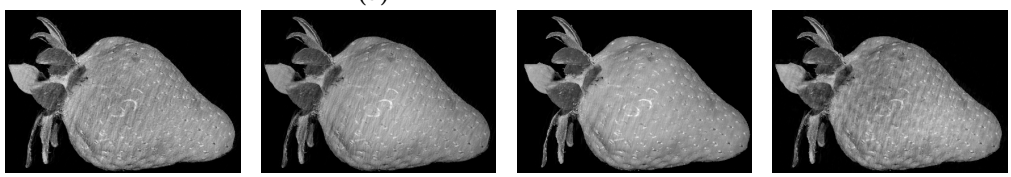
first mutation parameter discussed here is the mutation probability coefficient C_{MP}. As declared by Alg.10, C_{MP} is the probability for each individual to undergo mutation. This means, for example, that if $C_{MP} = 0.3$, each individual has a 30% probability to mutate, which implies that, among the R individuals, about the 30% will undergo mutation.



(a)

Name	Value
R	512
C_{RF}	1
C_R	1.8
C_{GB}	0.5
C_D	0.3
F_{FC}	*true*
C_{MP}	–
C_{MF}	1
C_{ML}	1
N_{HIO}	40
N_{ER}	20

(b)



(c) $C_{MP} = 0$ **(d)** $C_{MP} = 0.1$ **(e)** $C_{MP} = 0.3$ **(f)** $C_{MP} = 0.6$

Figure 3.11: MPR results as function of the *mutation probability* coefficient C_{MP}. In (a), the error behavior as function of the number of generations for different C_{MP} values. MPR parameters for this test are listed in table (b). Results for increasing values of C_{GB} are depicted from (c) to (f).

Fig.3.11 depicts the result for different values of C_{MP}. Error values as function of generations are plotted in Fig.3.11a, while visual representations of the results are provided by Fig.3.11c, 3.11d, 3.11e and 3.11f. In this situation, i.e. with this diffraction data and with the other parameters set to the values listed in Fig.3.11b, the best result is achieved by $C_{MP} = 0.3$. The case $C_{MP} = 0$ (no mutation) well depicts the importance of a mutation step: in fact, as underlined by the dashed line representing the error of the middle element $\mathcal{P}[\frac{R}{2}]$, it appears that the individuals in the population are too much similar, and, after some generations, the algorithm stagnates. $C_{MP} = 0.1$ and $C_{MP} = 0.3$ introduce a well balanced amount of new genetic pool which avoids this issue. The case $C_{MP} = 0.6$ provides instead a too strong mutation, such that MPR is no more able to converge. Low values for C_{MP} privilege *stability*, while high values (> 0.3) make MPR *ergodic*.

These results well represent a common situation with real experimental data: however, the optimal value of C_{MP} highly depends also on the other two parameters involved in the mutation step, C_{ML} and C_{MF}.

3.1.7 The mutation strength

Due to the fact that the genetic pool of each individual is composed by floating point values, a mutation of those genes can be performed in different ways. In particular, the mutation, as said before, modifies the phases of a given individual, and the way in which

it is performed can be tuned. The two parameters involved in the *Mutation* step that tune its strength are the *mutation level coefficient* C_{ML} and the *mutation fraction coefficient* C_{MF}.

The first one rules how much a phase at a given coordinate (i, j) is randomly modified: if $C_{ML} = 1$ the new phase is totally random, while if, for example, $C_{ML} = 0.5$ the new phase will be a random value in an interval of size π centered on the original value $\arg[\tilde{\rho}_{i,j}]$.

The latter parameter, C_{MF}, sets up the probability, for a given coordinate (i, j), to undergo mutation. In this way, if $C_{MF} = 1$ all the phases will be modified: if, for example, $C_{MF} = 0.6$, each phase will have a 60% probability to be mutated, such that about the 60% of the total number of genes will be modified.

The optimal value for these parameters cannot be evaluated in an absolute sense. In fact, the role of the mutation step is to provide *ergodicity* to the algorithm, i.e. let the algorithm better explore the space of possible solutions. If, for example, the diffraction pattern I has very low noise but provides a low oversampling degree, the mutation step must provide a high level of *ergodicity* because the *iterative projection algorithms* in the *Self-improvement* step will suffer from stagnation. On the other side, if the diffraction pattern I is particularly noisy, the issue with the *Self-improvement* step is the lack of *stability*, such that the parameters for the mutation step will be more conservative. These two parameters, C_{ML} and C_{MF}, are, in a sense, complementary: high values of C_{ML} and low values of C_{MF} provides effects similar to low values of the first and high values of the second.

Fig.3.12 provides an overview on different combinations for these two parameters. A direct evaluation of the total mutation entity is given by the dashed lines in Fig.3.12a, which represent the error value of the "middle" individual: the bigger the distance between the dashed and the continuous line, the higher the ergodicity provided by the *Mutation* step. On the other side, the lower the distance between those two lines, the higher the *stability* of the algorithm.

Figures from 3.12c to 3.12k show the results of the phase retrieval procedures whose behavior is plotted in Fig.3.12a. The best result is in this case provided by the combination of $C_{ML} = 0.75$ and $C_{MF} = 0.75$, as underlined by Fig.3.12a and Fig.3.12g.

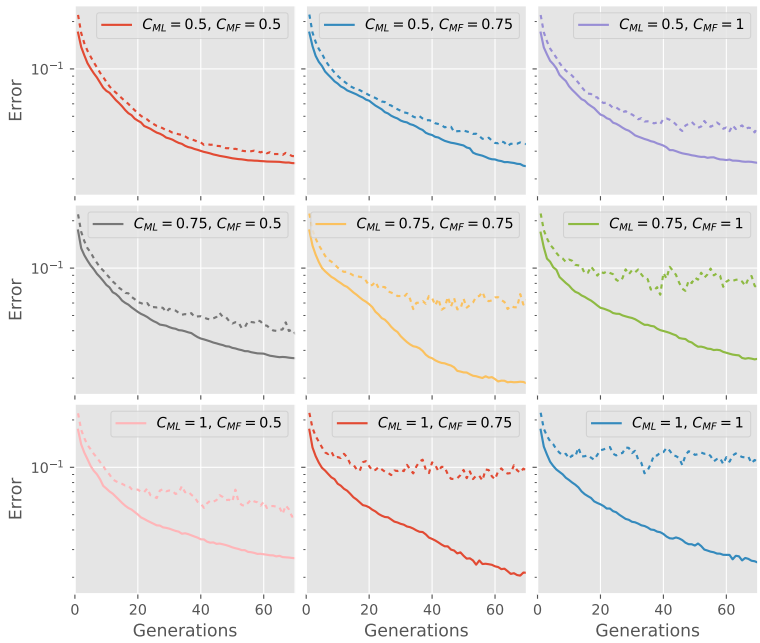
In this case, the mutation probability coefficient C_{MP} has been set to 0.3 (see Fig.3.12b), which means that each individual has a 30% probability to mutate.

Fig.3.13 shows instead the same combinations of C_{ML} and C_{MF} for $C_{MP} = 0.6$, which means that about the 60% of individuals underwent mutation. This choice is more *ergodic* than the previous case, such that high values for C_{ML} and C_{MF} make the algorithm too unstable, as the lower right part of Fig.3.13a depicts. In this case, the best result is achieved by a more conservative choice of $C_{ML} = 0.5$ and $C_{MF} = 0.5$.

It's worth remarking that the optimal values given by these plots (Fig.3.12 and Fig.3.13) cannot be assumed as the optimal combinations for all the possible datasets. However, the general behavior of the population $\mathcal{P}$ as function of these parameters is well depicted by these examples and can be considered a common conduct.

3.1.8 The algorithm sequence for Self-Improvement

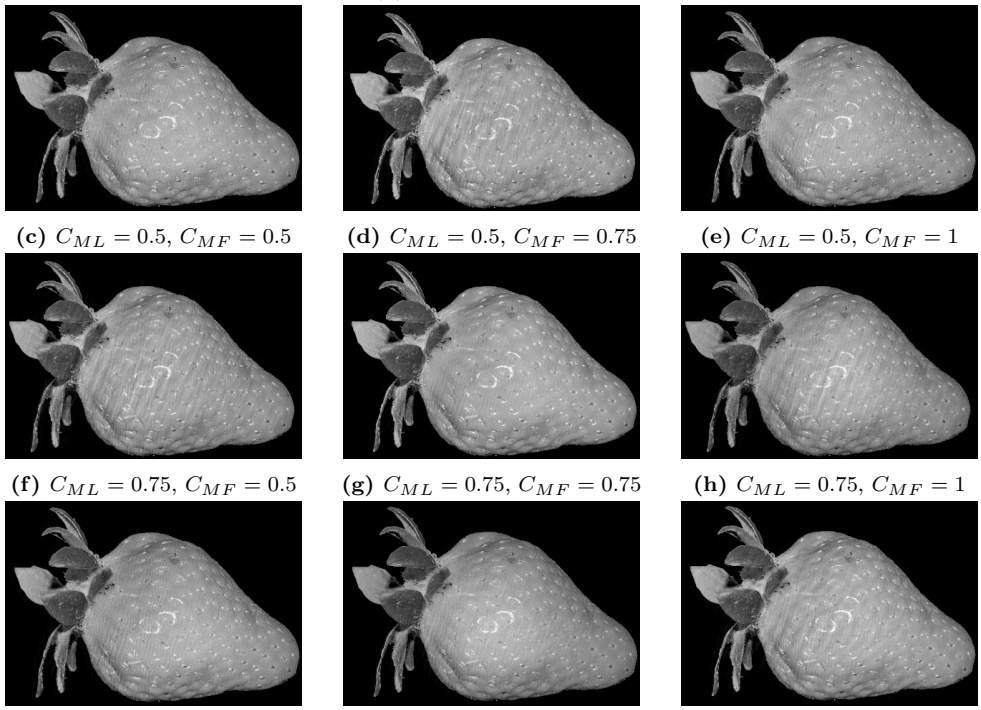
The choice of the suited algorithm (or set of algorithms) for facing the phase retrieval problem is anything but a trivial choice. Actually, the perfect iterative projection algorithm, i.e. the one that always gives the best result, does not exist. Moreover, the majority of these algorithms depend on, at least, a parameter (such as the β for HIO and RAAR), whose best value depends on the treated dataset. The reason is that different



(a)

Name	Value
R	512
C_{RF}	1
C_R	1.8
C_{GB}	0.5
C_D	0.3
F_{FC}	*true*
C_{MP}	0.3
C_{MF}	–
C_{ML}	–
N_{HIO}	40
N_{ER}	20

(b)

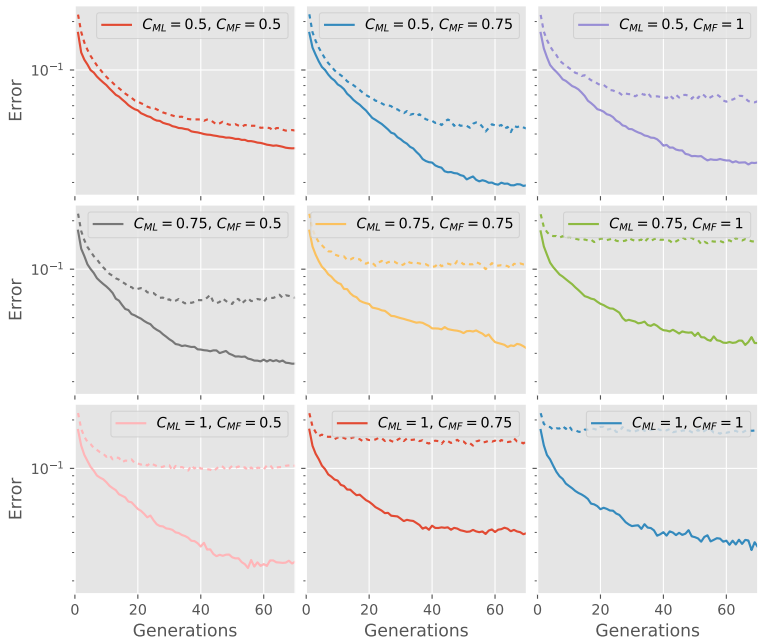


(c) $C_{ML} = 0.5$, $C_{MF} = 0.5$ **(d)** $C_{ML} = 0.5$, $C_{MF} = 0.75$ **(e)** $C_{ML} = 0.5$, $C_{MF} = 1$

(f) $C_{ML} = 0.75$, $C_{MF} = 0.5$ **(g)** $C_{ML} = 0.75$, $C_{MF} = 0.75$ **(h)** $C_{ML} = 0.75$, $C_{MF} = 1$

(i) $C_{ML} = 1$, $C_{MF} = 0.5$ **(j)** $C_{ML} = 1$, $C_{MF} = 0.75$ **(k)** $C_{ML} = 1$, $C_{MF} = 1$

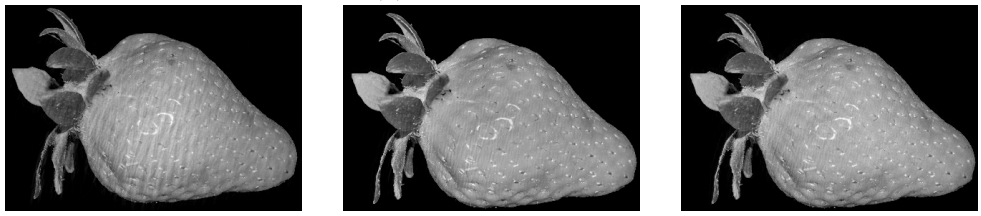
Figure 3.12: MPR results for different configurations of the C_{ML} and C_{MF} coefficients with the mutation probability coefficient set to $C_{MP} = 0.3$. In (a), the error behavior (in logarithmic scale) as function of the number of generations for different coefficient values. MPR parameters for this test are listed in table (b). Results are depicted from (c) to (k).



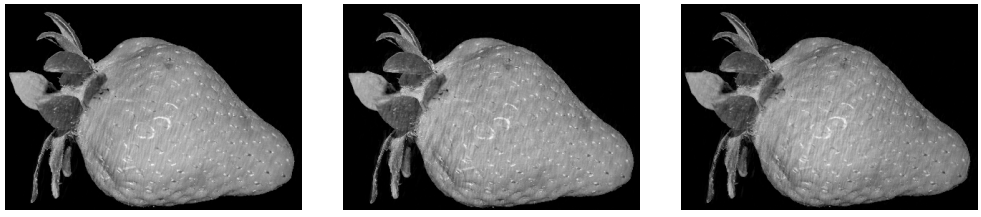
(a)

Name	Value
R	512
C_{RF}	1
C_R	1.8
C_{GB}	0.5
C_D	0.3
F_{FC}	*true*
C_{MP}	0.6
C_{MF}	–
C_{ML}	–
N_{HIO}	40
N_{ER}	20

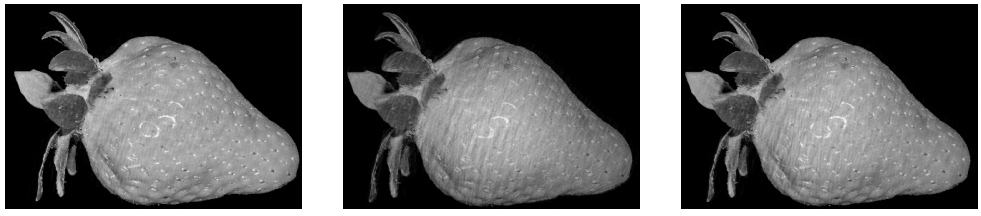
(b)



(c) $C_{ML} = 0.5$, $C_{MF} = 0.5$ **(d)** $C_{ML} = 0.5$, $C_{MF} = 0.75$ **(e)** $C_{ML} = 0.5$, $C_{MF} = 1$

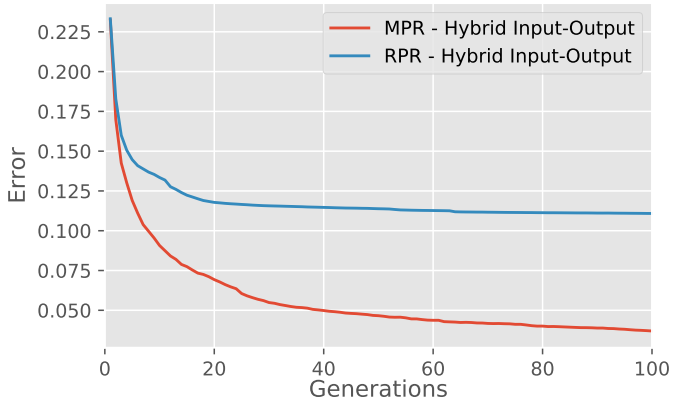


(f) $C_{ML} = 0.75$, $C_{MF} = 0.5$ **(g)** $C_{ML} = 0.75$, $C_{MF} = 0.75$ **(h)** $C_{ML} = 0.75$, $C_{MF} = 1$



(i) $C_{ML} = 1$, $C_{MF} = 0.5$ **(j)** $C_{ML} = 1$, $C_{MF} = 0.75$ **(k)** $C_{ML} = 1$, $C_{MF} = 1$

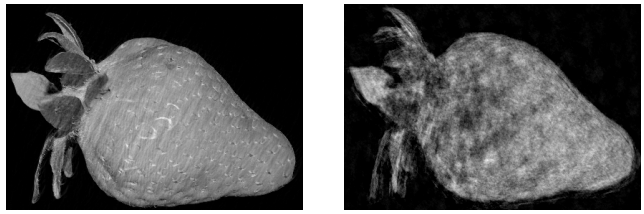
Figure 3.13: MPR results for different configurations of the C_{ML} and C_{MF} coefficients with the mutation probability coefficient set to $C_{MP} = 0.6$. In (a), the error behavior (in logarithmic scale) as function of the number of generations for different coefficient values. MPR parameters for this test are listed in table (b). Results are depicted from (c) to (k).



(a)

Name	Value
R	512
C_{RF}	1
C_R	1.8
C_{GB}	0.4
C_D	1
F_{FC}	*true*
C_{MP}	0.25
C_{MF}	0.5
C_{ML}	0.75
N_{HIO}	40

(b)



(c) MPR (d) RPR

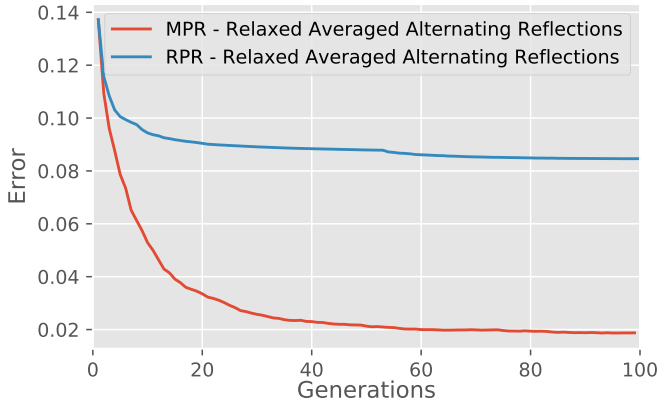
Figure 3.14: Comparison between MPR and the standard RPR approach with the same amount of Hybrid Input-Output algorithm iterations. In (a), the error behavior as function of the number of generations. Algorithm parameters for this test are listed in table (b). MPR result is displayed in (c), while RPR result is shown in (d).

algorithms provide a different balance between *stability* and *ergodicity*, such that different datasets may prefer a more *stable* or a more *ergodic* algorithm.

In this section different algorithms for the MPR *Self-Improvement* step will be tested on the same simulated diffraction data, and their behavior will be compared to their *standard* use, what has been previously called *Random search Phase Retrieval* (RPR) approach. It's worth remarking that the simulated dataset is affected by noise, lack of data in the central zone and a low knowledge on the support function, which make phase retrieval a non trivial task.

The first test concerns the use of the Hybrid Input-Output (HIO) algorithm as *Self-Improvement* operator, which was the first proposed algorithm with good convergence properties (Fienup 1978), described in the previous chapter. Fig.3.14 provide an example about the performances of HIO inside MPR. In this case, for each generation, each of the $R = 512$ individuals undergo $N_{\text{HIO}} = 40$ iterations of HIO ($\beta = 0.9$), as described by Fig.3.14b. As Fig.3.14c depicts, MPR correctly retrieved the shape of the sample and many high-resolution details, even if some stripes at middle resolution arises. It's however worth comparing this result with the standard RPR one, both by looking at the plot in Fig.3.14a and at the result in Fig.3.14d: even if the shape and some details are somehow retrieved, the results is far away from the solution. This difficulties in finding the correct solution mainly arise from noise and missing low-resolution information, because of both the lack of data in the central zone of the pattern and the inaccurate support function.

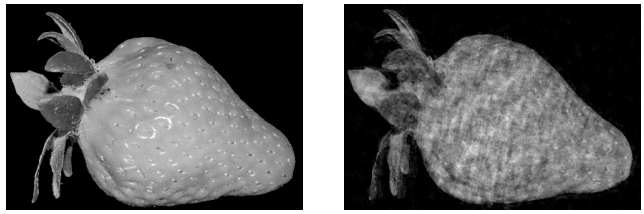
A second example is about the performance of the Relaxed Averaged Alternating Re-



(a)

Name	Value
R	512
C_{RF}	1
C_R	1.8
C_{GB}	0.4
C_D	1
F_{FC}	*true*
C_{MP}	0.25
C_{MF}	0.5
C_{ML}	0.75
N_{RAAR}	40

(b)



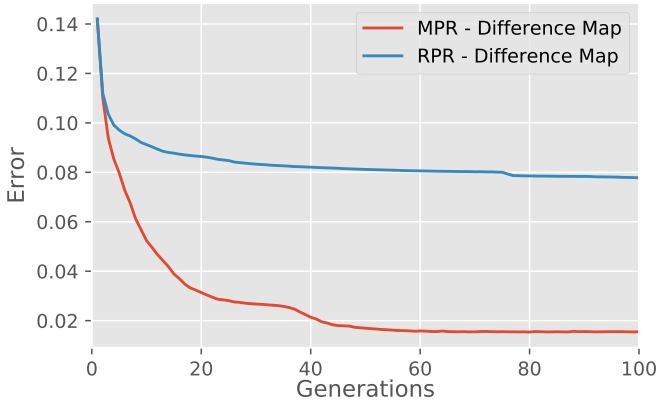
(c) MPR **(d)** RPR

Figure 3.15: Comparison between MPR and the standard RPR approach with the same amount of Relaxed Averaged Alternating Reflections algorithm iterations. In (a), the error behavior as function of the number of generations. Algorithms parameters for this test are listed in table (b). MPR result is displayed in (c), while RPR result is shown in (d).

flections (RAAR) algorithm (Luke 2004), whose results are provided by Fig.3.15. Unlike the previous case, MPR here can be considered to have reached the solution (Fig.3.15c), with no evidence of artifacts at any resolution. Again, the RPR approach failed to reach the solution, even if some details are better retrieved with respect to the HIO case, which is underlined by a lower error (Fig.3.15a).

The third example concerns the use of the Difference Map (DM) algorithm (Elser 2003) as Self-Improvement. Results provided by Fig.3.16 are similar to what we have previously seen with the RAAR algorithm: MPR reaches the solution, shown in Fig.3.16c, while RPR stays far away (Fig.3.16d). This difference is again well underlined by the plot in Fig.3.16a.

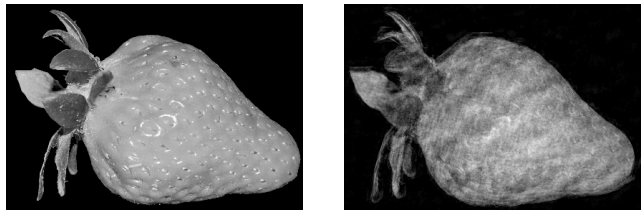
After this three examples, a general comment is needed. These results showed a total lack of convergence of the RPR approach with HIO, RAAR and DM algorithms. This fact, actually, does not mean that these algorithms are not able to treat this kind of data: different values for their parameters, such as β for HIO, or a greater number of iterations may lead to convergence. Moreover, the use of the Shrink-wrap to refine the support function would help a lot the quality of their result. However, the clear fact that arises from these few tests is that the use of those algorithms in the MPR framework makes their result much less dependent on their parameters. In fact, as stated in the previous sections, the exploration of the space of configuration is largely entrusted to the genetic operators, letting the Self-Improvement step, based on *iterative projection algorithms*, focus on the search for near optima.



(a)

Name	Value
R	512
C_{RF}	1
C_R	1.8
C_{GB}	0.4
C_D	1
F_{FC}	*true*
C_{MP}	0.25
C_{MF}	0.5
C_{ML}	0.75
N_{DM}	40

(b)



(c) MPR **(d)** RPR

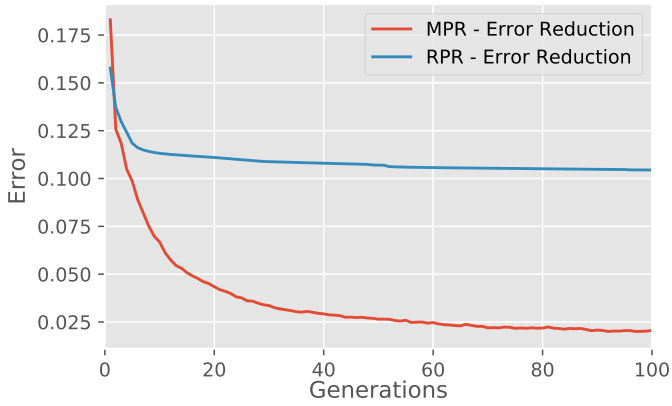
Figure 3.16: Comparison between MPR and the standard RPR approach with the same amount of Difference Map algorithm iterations. In (a), the error behavior as function of the number of generations. Algorithms parameters for this test are listed in table (b). MPR result is displayed in (c), while RPR result is shown in (d).

To enforce this statement, a last test on the Error Reduction (ER) algorithm (Fienup 1978) is provided in a dedicated subsection.

Error Reduction only phasing

The ER algorithm, as said before, ensures that the error value in Eq.(1.20) always decreases from a step to the next one. Thanks to this peculiarity, ER is the only phase retrieval algorithm for which a convergence to an optimum is mathematically ensured. As we saw, this is at the same time the strength and the weakness of ER approach: from one side, it always converges to the nearest optimum, thus it is a really *stable* algorithm. From the other side, this stability is paid with a total lack of *ergodicity*, that we have defined as the ability to well explore the space of configurations. These peculiarities make the ER algorithm impossible to be used alone for phase retrieval. In fact, Fienup, in 1976, stated that "*In practice, when the Error Reduction approach is used for the present application for large two-dimensional images, the mean-squared error decreases rapidly for the first few iterations, but then decreases extremely slowly for later iterations, requiring an impractically large number of iterations for convergence*" (Fienup 1975). ER is commonly combined with other more *ergodic* approaches (like HIO) to enhance their convergence, e.g. some iterations of HIO and some of ER are alternated.

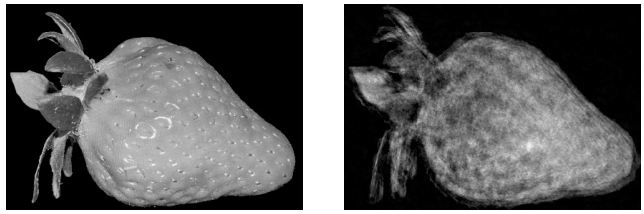
Here we try, instead, to exploit the Error Reduction algorithm alone to perform a full phasing from totally random starting guesses. The diffraction data and the support



(a)

Name	Value
R	512
C_{RF}	1
C_R	1.8
C_{GB}	0.4
C_D	1
F_{FC}	$true$
C_{MP}	0.25
C_{MF}	0.75
C_{ML}	0.75
N_{ER}	40

(b)



(c) MPR **(d)** RPR

Figure 3.17: Comparison between MPR and the standard RPR approach with the same amount of Error Reduction algorithm iterations. In (a), the error behavior as function of the number of generations. Algorithms parameters for this test are listed in table (b). MPR result is displayed in (c), while RPR result is shown in (d).

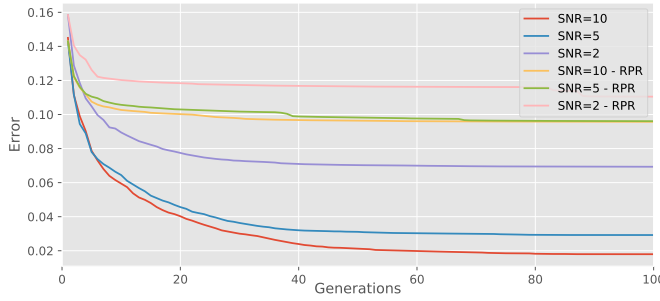
function are as like as the previous tests.

Results are shown by Fig.3.17. Fig.3.17c and Fig.3.17d depict the results after 100 generations of the MPR and RPR algorithms respectively. For each generation, 40 iterations of Error Reduction algorithm are performed on each individual in the population $\mathcal{P}$, whose size is $R = 512$ (as stated by the table in Fig.3.17b). The error value, plotted as function of the generation, clearly reflects the huge difference between the standard RPR approach and the MPR one: MPR has practically reached the goal, while RPR stagnates far from the solution as expected. The importance of this result is remarked by reminding that the support function, shown in Fig3.3, has been kept constant, i.e. it has not been refined by the use of the *Shrinkwrap* algorithm.

This simple test clearly points out one of the strengths of MPR: the ability to be *ergodic* without giving up *stability*.

3.2 Results on simulated data

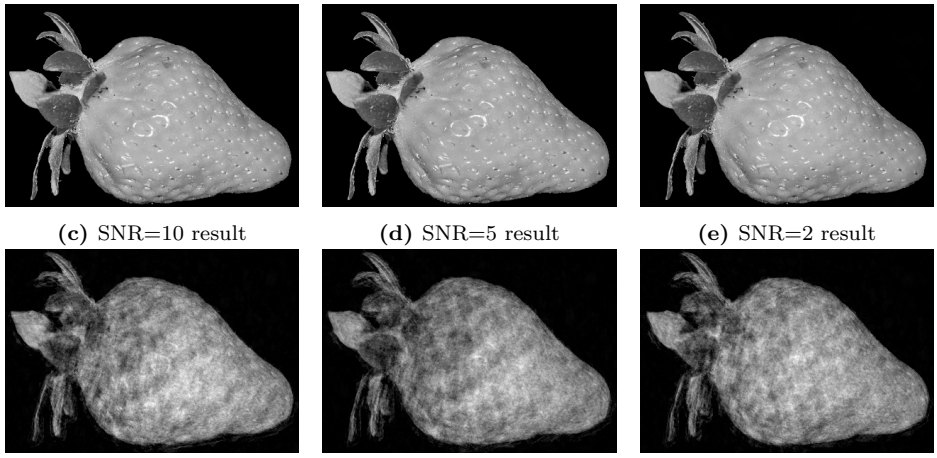
One test will concern real-valued data, i.e. the density ρ has a null imaginary part, such that this constraint can be added to the phasing process. A second test will concern instead complex-valued data, well known to be much more tricky to treat. For what concerns the algorithms, only HIO and ER will be used, by alternating N_{HIO} iterations of HIO algorithm and N_{ER} iterations of ER algorithm.



(a)

Name	Value
R	8192
C_{RF}	1
C_R	1.8
C_{GB}	0.5
C_D	1
F_{FC}	*true*
C_{MP}	0.2
C_{MF}	0.66
C_{ML}	0.66
N_{HIO}	60
N_{ER}	40

(b)



(c) SNR=10 result **(d)** SNR=5 result **(e)** SNR=2 result

(f) RPR with SNR=10 **(g)** RPR with SNR=5 **(h)** RPR with SNR=2

Figure 3.18: MPR and RPR results as function of the amount of noise in diffraction data. In (a), the error behavior as function of the number of generations for different SNR values, both for MPR and RPR. MPR parameters for this test are listed in table (b). Results for increasing values of noise are depicted from (c) to (e) for MPR and from (f) to (h) for the standard RPR approach.

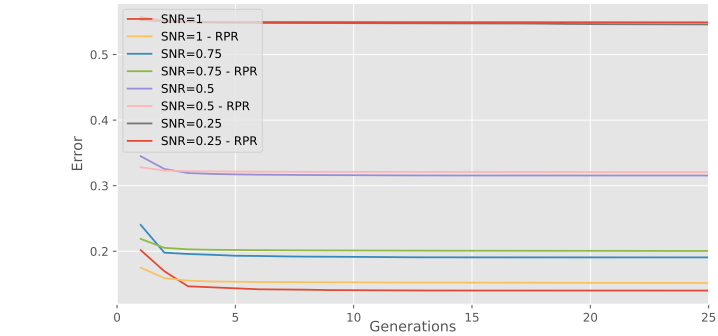
3.2.1 2D Real-valued tests

Data-limited dataset

This first test on real-valued data has been performed using a loose support constraint and an non-complete diffraction pattern, affected also by some noise.

Fig.3.18 shows MPR results for different amounts of noise, introduced in the diffraction pattern I via Eq.(3.1). The first row of figures (Fig.3.18c, 3.18d and 3.18e) provide the MPR results for SNR=10, SNR=5 and SNR=2 respectively. The row below (Fig.3.18f, 3.18g and 3.18h) provide, instead, RPR results for the same amount of noise.

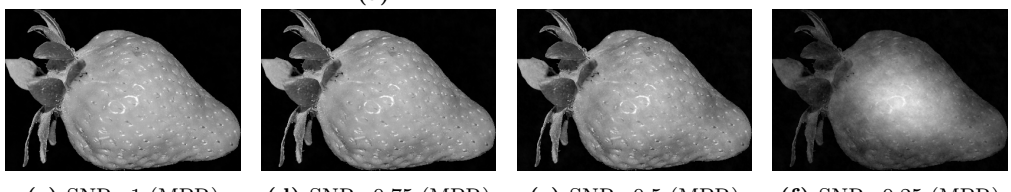
For this test, the hard task was to retrieve the correct density distribution by retrieving also the missing values at low resolution, without precise information about the support function. Thus, the algorithm must retrieve these low resolution values basing on the high resolution ones, that are affected by noise. Both the plot in Fig.3.18a and the images below show a nearly perfect result of MPR, while RPR fails in all of the three cases. It's worth reminding that this dataset belongs to the category of data-limited datasets, as it is clear that, when the solution is found, it appears almost equal to the correct noise-free one.



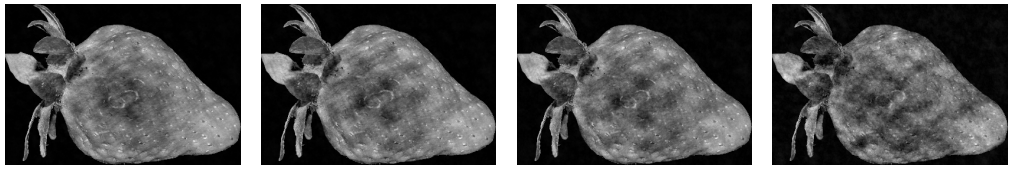
(a)

Name	Value
R	8192
C_{RF}	1
C_R	1.8
C_{GB}	0.5
C_D	1
F_{FC}	*true*
C_{MP}	0.2
C_{MF}	0.66
C_{ML}	0.66
N_{HIO}	$20, 10, 5, 2$
N_{ER}	40

(b)



(c) SNR=1 (MPR) **(d)** SNR=0.75 (MPR) **(e)** SNR=0.5 (MPR) **(f)** SNR=0.25 (MPR)



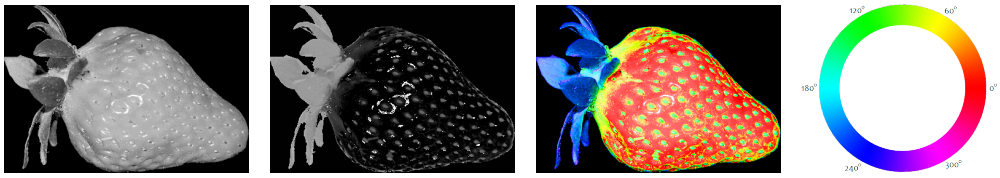
(g) SNR=1 (RPR) **(h)** SNR=0.75 (RPR) **(i)** SNR=0.5 (RPR) **(j)** SNR=0.25 (RPR)

Figure 3.19: MPR and RPR results as function of the amount of noise in diffraction data. In (a), the error behavior as function of the number of generations for different high amounts of noise, both for MPR and RPR. MPR parameters for this test are listed in table (b). Results for increasing values of noise are depicted from (c) to (f) for MPR and from (g) to (j) for the standard RPR approach. Note that the amount of HIO iterations is different, depending on the SNR value, due to the instability of the HIO algorithm when facing noisy data.

Noise-limited dataset

This second test on real valued data is performed by assuming a perfect knowledge on the support function. Even it is far from reality, where an uncertainty on the support shape is always present, this test aims to totally focus on how the approaches handle the noise. Thus, the support function has been set to perfectly fit the simulated sample (Fig.3.1a). Again, the presence of a beam stopper has been simulated, as depicted in the previous test. Different levels of noise, computed via Eq.(3.1), has been inserted, i.e. SNR has been set to 1, 0.75, 0.5 and 0.25 respectively. This really high amount of noise introduces many issues when using iterative projection algorithms. In fact, this noise strongly reshapes the set $\mathcal{M}$, causing a detachment from the set $\mathcal{S}$ and making the landscape of the error $E[\rho]$ much more riddled. In particular, the higher the noise in the diffraction pattern, the more likely to find a ρ far from the solution that has a similar error value.

Moreover, due to this corrugated shape, algorithms based on *feedback* like HIO are too unstable to identify a solution, and they continue to travel in the space of configurations. For this reason, the results shown in Fig.3.19 have been obtained by using different amounts of HIO iterations, depending on the noise amount. In particular, 20 iterations



(a) $|\rho|$ **(b)** $\arg[\rho]$ **(c)** ρ **(d)** Phase colormap

Figure 3.20: Solution of the complex valued dataset exploited for the tests. In (a), the module of the solution, while the phases of the solution are depicted in grayscale in figure (b). Modules and phases can be combined for an easy visualization (c) by using the colormap in (d) for the phases and assigning a lightness proportional to (a).

of HIO have been performed for the case SNR=1, 10 for SNR=0.75, 5 for SNR=0.5 and 2 for SNR=0.25. These iterations are then followed by 40 iterations of ER algorithm for all of the four cases.

The first row of images (from Fig.3.19c to Fig.3.19f) represents MPR results for decreasing values of the SNR. The second row provides, instead, RPR the results (from Fig.3.19g to Fig.3.19j). Visual differences are striking, and MPR fails to retrieve the low-resolution details (the ones hidden by the beam stopper) only when SNR=0.25. However, the most interesting aspect of this example concerns the error values of the reconstructions, plotted in Fig.3.19a. It's, first of all, clear that the higher amount of noise, the higher the minimum reachable error. Moreover, even if the MPR result looks much better than the RPR one, the error difference between the two approaches decreases with the increasing of noise. In particular, the RPR error is about 8% greater than the MPR one for SNR=1, 4.6% for SNR=0.75, 1.6% for SNR=0.5 and about 0.4% for SNR=0.25. This means that, in presence of low SNR, it's hard both to find the solution and to identify it.

As said before, for this test an increasingly lower amount of HIO iterations has been performed, because, otherwise, the HIO algorithm would diverge from the solution. On the other side ER, the most stable algorithm, tends to stuck in local optima, and this is one of the reasons of why the RPR approach failed. The advantage of MPR with respect to the standard RPR derives from its ability to explore the space of configurations regardless of the exploited algorithm.

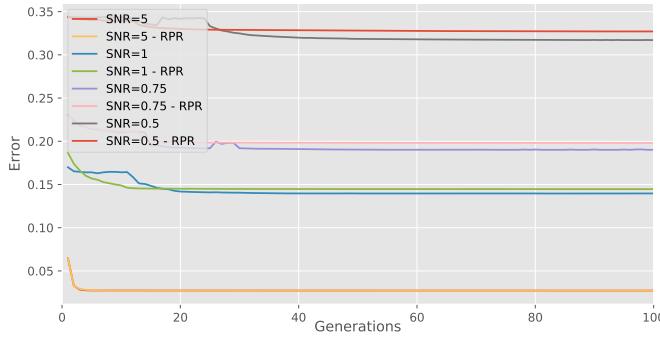
3.2.2 Complex-valued tests

A second class of tests concerns complex-valued data. This dataset has been created by giving to ρ^{sol} both a real and an imaginary part. This has been done by assigning a value to $\arg[\rho]$, as depicted by Fig.3.20b, and keeping $|\rho|$ (Fig.3.20a) equal to the one of the real test dataset (Fig.3.1a). As shown in Fig.3.20c, a convenient way to display this complex data is to assign the lightness to $\arg[\rho]$ and the hue to $\arg[\rho]$, following the scheme of Fig.3.20d.

As Fig.3.20 depicts, the assigned phases range from 0 to 2π: this kind of data is known to be particularly hard to retrieve (Miao et al. 1998).

The simulated diffraction pattern has been deprived of the central pixels, as like as what was done for the real case (Fig.3.2a). In this test, the support function has been assumed to be known exactly, focusing instead to the behavior of the algorithm in presence of high noise.

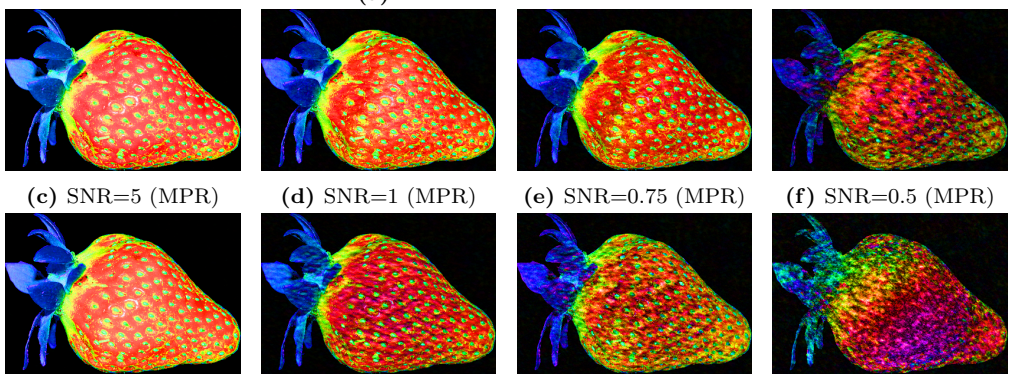
Fig.3.21 shows results for different amounts of noise, comparing the reconstructions of MPR with respect to RPR. As done before with real-valued data, a different number



(a)

Name	Value
R	8192
C_{RF}	1
C_R	1.8
C_{GB}	0.5
C_D	1
F_{FC}	*true*
C_{MP}	0.2
C_{MF}	0.66
C_{ML}	0.66
N_{HIO}	$20, 10, 5, 2$
N_{ER}	40

(b)



(c) SNR=5 (MPR) **(d)** SNR=1 (MPR) **(e)** SNR=0.75 (MPR) **(f)** SNR=0.5 (MPR)

(g) SNR=5 (RPR) **(h)** SNR=1 (RPR) **(i)** SNR=0.75 (RPR) **(j)** SNR=0.5 (RPR)

Figure 3.21: MPR and RPR results as function of the amount of noise in diffraction data for a complex-valued density distribution. In (a), the error behavior as function of the number of generations for different high amounts of noise, both for MPR and RPR. MPR parameters for this test are listed in table (b). Results for increasing values of noise are depicted from (c) to (f) for MPR and from (g) to (j) for the standard RPR approach. Note that the amount of HIO iterations is different, depending on the SNR value, due to the instability of the HIO algorithm when facing noisy data.

of HIO iterations has been used: 20 iterations of HIO have been performed for the case SNR=5, 10 for SNR=1, 5 for SNR=0.75 and 2 for SNR=0.5.

As displayed before in the real case, again MPR shows to provide a better estimation of the solution when the amount of noise becomes critical.

3.2.3 The Shrinkwrap algorithm

Results shown so far have been obtained by exploiting an a-priori information about the support constraint. In some cases, as in the real-valued tests, it was assumed to be known at a resolution lower than the one given by the diffraction pattern, while for the complex-valued test the support was assumed to be perfectly known. The a-priori knowledge of the support function can be considered a realistic situation only for those samples that can be imaged also via other techniques. These situations are peculiar, and, in general, the whole information about the sample extension must be extracted directly from the diffraction pattern.

The main information about the sample extension is provided by performing a Fourier

Transform of the diffraction pattern $I(\vec{k})$. In fact

$$\mathcal{F}[I](\vec{x}) = R_\rho(\vec{x}) = \int \rho(\vec{y})\bar{\rho}(\vec{y}-\vec{x})d\vec{y} \tag{3.5}$$

where R_ρ represents the autocorrelation function of ρ. It is known that the autocorrelation of a function $\rho(\vec{x})$ has an extension which is twice the one of the function itself. This mean, first of all, that a first estimation of the support function can be provided by applying a threshold to the Fourier Transoform of the diffraction pattern (i.e. the autocorrelation), due to the fact that it surely contains the whole sample.

An other improvement for the initial estimation of the support function is to exploit its double extension with respect to the sample one. In fact, given an autocorrelation function with a maximum extension D_x on the x axes and D_y on the y axes, the maximum extension of the support $S(\vec{x})$ will be $\frac{D_x}{2}$ and $\frac{D_y}{2}$ respectively. In this way, the maximum extension of the support function is known pretty well.

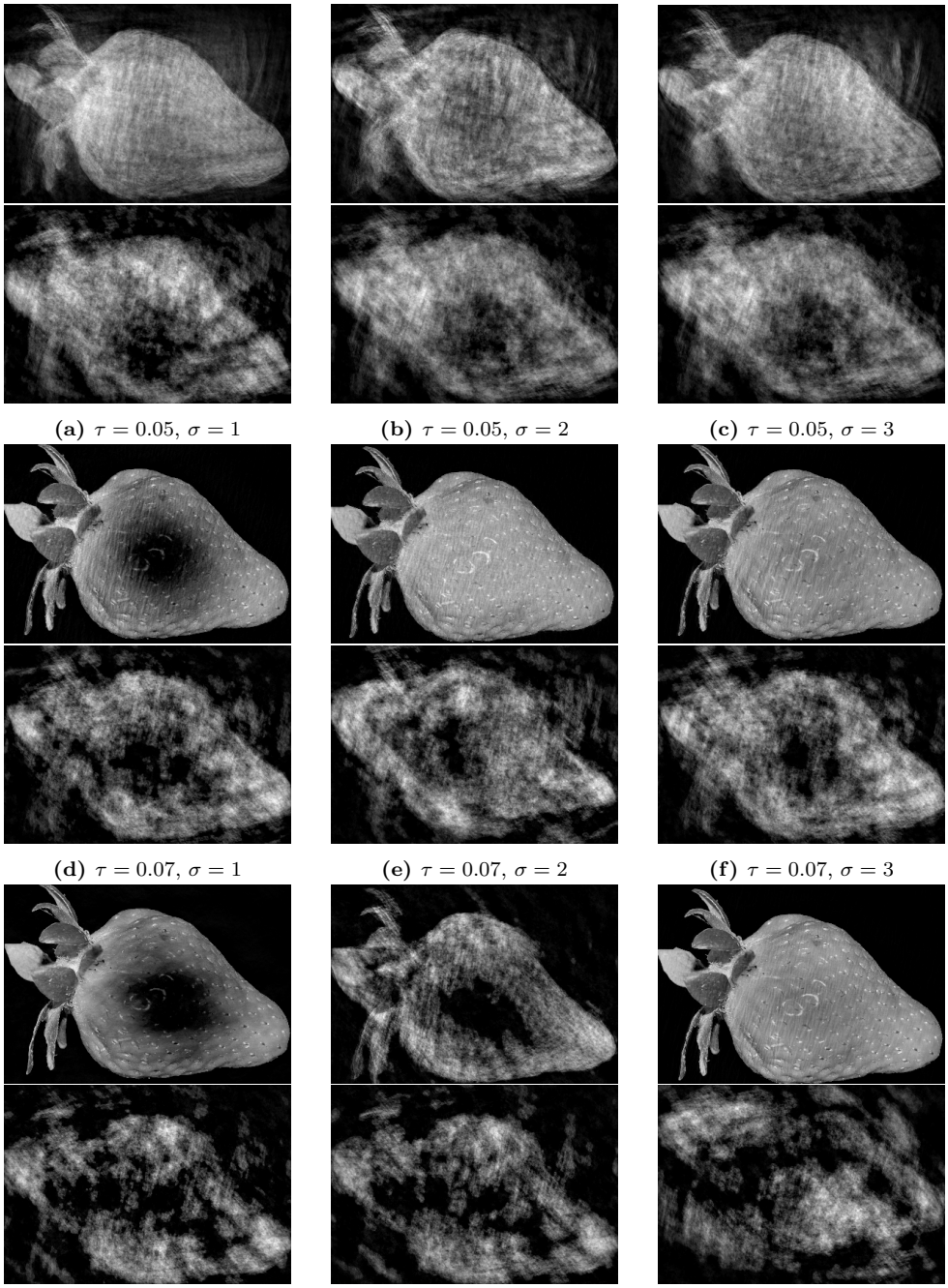
Once that a first support guess is given to the algorithm, the support function can be refined through the Shrinkwrap algorithm (Marchesini et al. 2003), as described in the second chapter. Just to remind, MPR provides a peculiar implementation of this approach, due to the fact that the support function represents a constraint to the problem, and it is impossible to compare two reconstructions *evolved* with supports of different extensions. Thus, the *standard Shrinkwarp* is exploited only on the best individual of the MPR population, while the others adapt the threshold value σ to get a support with the same number of pixels. In this way, each element of the population will have its own support function, but all of those supports will have the same extension (i.e. $\sum_{i,j} S_{i,j}$ is the same for all the individuals), such that their *fitness* value is comparable.

Here a test of the performances of the MPR version of the *Shrinkwrap* algorithm is presented. In this case, the experimental pattern is the one used for the real-valued test, shown in Fig.3.1. The pattern is affected by a SNR=10 (Eq.3.1), while the support function is a rectangle of dimension $S_x \times S_y$, where S_x and S_y are the maximum extensions of the solution (Fig.3.1a) in the two dimensions.

First of all, it's worth reminding that the *Shrinkwrap* algorithm depends on two parameters, that are σ (which rules how much the reconstruction is smoothed) and τ (which sets the threshold at which the reconstruction is cut to get the new support). The ability of the algorithm to identify the correct support function highly depends on the right choice for these two parameters (in particular the threshold value τ).

Fig.3.22 shows the comparison between MPR (upper images) and RPR (lower images) for different combinations of σ and τ. What clearly emerges from this comparison is that, first of all, MPR is capable of extracting the right sample shape almost for all combinations of the parameters. Results in Fig.3.22e, Fig.3.22f and Fig.3.22i provide not only the correct shape, but also a good guess for the solution, while, in the other cases, a relatively good guess for the shape is given, but the reconstruction is still far from the solution. For what concerns the RPR approach, it fails to identify the support shape in all of the situations provided by Fig.3.22. This result does not mean that RPR always fails to retrieve this data, but simply that the choice of the optimal τ and σ is much more tricky, and usually their values (at least σ) varies during the phase retrieval procedure (Marchesini et al. 2003). Moreover, the perfect symmetry of the provided initial support function makes the algorithms fall into the so-called *twin images* issue (Fienup & Wackerman 1986), where $\rho(\vec{x})$ and $\rho^*(-\vec{x})$ are superimposed.

This comparison is clearly not exhaustive, because a more complete series of tests should be performed, investigating different levels of noise, different values of oversampling



(a) $\tau = 0.05$, $\sigma = 1$ **(b)** $\tau = 0.05$, $\sigma = 2$ **(c)** $\tau = 0.05$, $\sigma = 3$

(d) $\tau = 0.07$, $\sigma = 1$ **(e)** $\tau = 0.07$, $\sigma = 2$ **(f)** $\tau = 0.07$, $\sigma = 3$

(g) $\tau = 0.1$, $\sigma = 1$ **(h)** $\tau = 0.1$, $\sigma = 2$ **(i)** $\tau = 0.1$, $\sigma = 3$

Figure 3.22: Comparison between the result obtained by the *Shrinkwrap* algorithm for MPR (upper images) and RPR (lower images) with the same amount of noise in the diffraction pattern. Results are displayed for different combinations of the *Shrinkwrap* parameters τ and σ. This dataset turns out to be hard to solve due to the presence of the beamstopper and noise in diffraction data. Moreover, the starting guess for the support function is perfectly symmetrical, such that the algorithms tends to retrieve two overlapped versions of the solution rotated by 180 degrees. This fact makes the *Shrinkwrap* algorithm fail when used in combination with the RPR approach.

degree and different sizes of the beam stopper. Fig.3.22 however shows a key feature of MPR, which is its (relative) stability of the performance with respect to the algorithm parameters.

3.3 Some considerations about test results

This chapter could be considered, in a way, a stand-alone work. The motivation of this work was the investigation of MPR performances as function of its parameters and as function of the issues of diffraction data.

In the first part we played with MPR parameters, and that section may result of interest to a newbie MPR user who may need an overview on the effects of parameters changes. In this part the reader may have been frightened of the amount of parameters on which MPR depends. This is a common feature of stochastic optimization methods: the best convergence performances are often reached after a meticulous tuning of the algorithm parameters. However, this is not entirely true for MPR. In fact, MPR can be considered a hybrid approach, where iterative projection algorithms play an important role. For this reason, MPR is much less sensitive with respect to a variation of its parameters. This statement can be verified by looking at the parameters tables provided for almost every example in this chapter: MPR parameters are almost similar from a case to an other, even if the dataset changes. Thus, it can be said that:

1. MPR doesn't need an excessively fine tuning of its parameters, and, as consequence,
2. MPR can provide good performances even with non-optimal parameters.

This fact facilitates a lot a beginner user when he approaches the MPR framework.

A second (small) part of the chapter was dedicated to the use of different *iterative projection algorithms* inside the Self-improvement operation of MPR. The possibility to insert inside MPR any desired phase retrieval algorithm is one of the main strengths of the approach. In particular, different algorithms may suits better for different kinds of data, such that the user may decide to use an algorithm instead of an other. Moreover, it means that MPR doesn't compete with iterative approaches and it will benefit from any improvement of them in the future.

An example of this interpenetration between MPR and standard approaches is well represented by the integration of the *Shrinkwrap* algorithm (Marchesini et al. 2003) in the Self-improvement step. The performances of the *Shrinkwrap* algorithm inside MPR are well beyond its standard use in the (hard) example provided in the chapter.

The third part of the chapter may be interesting to those who are already familiar with phase retrievals algorithms and are well aware of the difficulties that some kind of data could present. Without listing all the results, it's worth saying that MPR always showed better performances than the standard RPR approach. This fact doesn't mean that MPR is always more convenient than RPR. In fact, the tests of this chapter concerns (simulated) datasets that are often hard to solve, due to noise, complexity of the density to retrieve, loose support function, low oversampling degree, etc.. Thus, we can state that it is worth using the MPR approach instead of the RPR one only when the latter fails to converge or to maximize the resolution.

As said at the starting point of the chapter, the examples provided here do not aim to be exhaustive, nor to prove that MPR is the best approach to face the phase retrieval problem. However, the relatively long experience of the author in the use of MPR suggests that MPR usually has, at least, performances not worse than the standard RPR approach.

Chapter 4

Results on Experimental Data

As mentioned in the second chapter, quite a lot of phase retrieval algorithms have been developed, and none of them can be assumed to be the best one. Moreover, tests of those approaches on simulated data, even if as close as possible to a realistic diffraction pattern, are only indicative, and the real proof of an algorithm reliability must concern experimental data.

The main issue when tests are performed on experimental data is that the "true" solution is not known, or at least it is known up to a certain resolution and/or precision. Thus, a direct comparison with the solution to the phase problem for a given diffraction data is not possible, and the only way to evaluate a reconstruction is to interpret the reconstruction error in Eq.(3.2). In many cases, the discrimination between a good and a bad reconstruction is entrusted to a visual interpretation of the result, mainly basing on what the scientist expects.

It can be easily stated, however, that the main issue that often prevents one to find the correct image is the identification of the correct support function, in particular for those experimental patterns that provide a low oversampling degree. In fact, with low oversampling degrees, the support function is expected to be wide. Under a mathematical point of view, this means that the support constraint is particularly loose, and the set $\mathcal{S}$ tends to approach the set $\mathcal{M}$ (see Fig.1.3b as reminder). The closer the two constraints, the lower the difference in error value between a correct result and an incorrect one. Moreover, if we add also the need to retrieve the correct support function via the Shrinkwrap algorithm, things will likely go wrong.

The oversampling degree σ of a diffraction pattern can be decided a-priori, by regulating the pixel numerical aperture of the detection apparatus. However, σ cannot be freely chosen, due to the fact that increasing of the oversampling degree means reducing the sample size when imaged. Thus, considering that the number of pixels is fixed by the detector, if from one side a low oversampling degree will make the phase retrieval a hard task, a too high oversampling degree will degrade too much the spatial resolution of the reconstruction.

An other issue concerning experimental data is the beam stopper size, that is the size of the central area of missing pixels in the diffraction pattern. The issue introduced by this area cannot be judged in an absolute sense, but it is strictly related to the oversampling degree. In fact, the size of the central spot of the diffraction pattern, which contains the low resolution information about the sample, has a size which is inversely proportional to the sample size. The higher the oversampling degree, the smaller the object size, the larger the central spot. On the other hand, the lower the oversampling degree, the bigger the object size, the thinner the central peak of the diffraction pattern. When a beam stopper hides those central pixels of the detector, the phase retrieval algorithm must extrapolate that information from the given pixels at higher resolution. Given a beam

stopper with a fixed size, if the oversampling degree is high, some part of the large central peak will survive, providing many useful information to identify the correct sample shape. If, instead, the oversampling degree is low, the whole central peak, and probably other secondary peaks, are totally missing, such that for the algorithm is much more difficult (and sometimes impossible) to retrieve that low resolution information.

If the mentioned features can be considered as general issues to face when attempting both 2D and 3D CDI, the latter is affected by a further obstacle which is strictly bounded to the way 3D data is prepared.

As long as 3D CDI exploits a set of bi-dimensional diffraction patterns, which are then assembled in the three dimensional Fourier space, a 3D diffraction data is much more incomplete than the 2D one. In fact, the acquired 2D diffraction patterns represent only slices on the 3D reciprocal space, such that only some slices of the 3D diffraction pattern are known, while the other values have to be retrieved along with the phases.

This fact makes 3D CDI more tricky, because usually this missing information resides at high resolution and phase retrieval algorithms, even if near to the correct solution, tend to badly estimate those values, which results in high-resolution artifacts in the reconstruction. This issue is clearly added to the greater computational weight of the three dimensional case, which is usually up to three orders of magnitude bigger than the 2D one.

In this chapter some results on experimental CDI data are presented, and each of them has some peculiar issues and points of interest that will be discussed case by case.

4.1 2D imaging

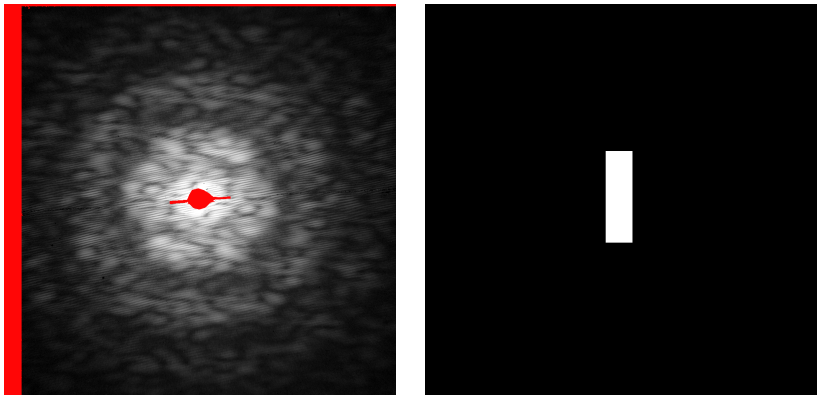
4.1.1 Golden clusters

It can be stated without doubts that the possibility to perform full CDI, i.e. to retrieve an unknown spatial density distribution for the diffraction pattern, was practically viable only thanks to the birth of the *Shrinkwrap* algorithm (Marchesini et al. 2003). The role of this approach it to retrieve the correct support function during the phase retrieval process. It results to be indispensable when an a-priori information on the support function is unavailable, and in particular in those cases where there are missing experimental values at low frequencies, as it happens in most datasets due to the presence of the beamstopper. The description of the Shrinkwrap algorithm by Marchesini et al. (2003) is presented along with a practical demonstration of the method on soft X-rays diffraction data. The dataset concerned two clusters of gold spheres of about 50 nm diameter.

As told in the previous chapters, the Shrinkwrap algorithm depends on two parameters. The support function is, in fact, updated after having applied a threshold τ to a reconstruction smoothed with a gaussian filter characterized by a width σ.

The vaule of τ refers to a fraction of the maximum pixel value of the reconstruction, such that if $\tau = 0.1$ the support function is defined by those pixels whose value is greater than the 10% of the maximum value of the smoothed reconstruction. Moreover, $\sigma = 2$ means, for example, that the reconstruction is convolved with a Gaussian function with a width of 2 pixels.

The Shrinkwrap algorithm is applied to the current estimation of the density every N iterations of iterative algorithm, and in particular Marchesini et al. (2003) updated the support function every 20 iterations of HIO algorithm. While the τ value has been kept fixed to $\tau = 0.2$, σ has been set to 3 at the first iteration, and then reduced of 1% each 20 HIO iterations down to a minimum value of 1.5 . The great success of the



(a) **(b)**

Figure 4.1: Diffraction pattern (a) and starting support function (b). Red pixels in (a) highlight areas where pixels are unknown.

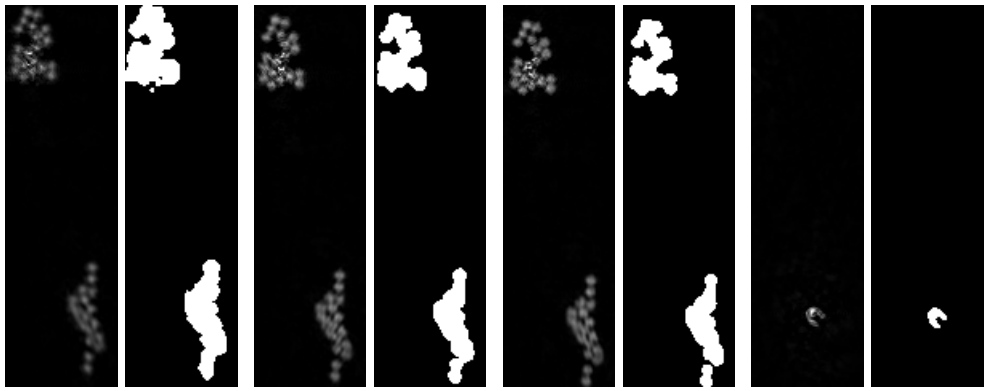
Shrinkwrap algorithm is proved by the fact that almost all CDI reconstructions make use of this approach.

However, the main issue of the Shrinkwrap algorithm resides in the use of the threshold applied to the reconstruction, ruled by the τ value. It is trivial to note that, when applying a threshold value to the density ρ, let's say $\tau = 0.2$, all the features of the reconstruction with a value $|\rho_{i,j}| < \tau \cdot \max[\rho]$ after the smoothing are excluded from the support function. If it doesn't represent an issue when the sample density distribution has well defined high-valued contours, it turns out to be a problem when the sample has smoothed contours and/or isolated features with low intensity. This consideration does not mean that the Shrinkwrap algorithm is sometimes useless, but simply that it may exclude interesting features of the sample from the support function. Thus, it appears evident that the lower the threshold value τ that gives the correct reconstruction, the more low-valued details of the reconstruction emerge, the better the reconstruction quality. A second, less crucial, issue of the Shrinkwrap algorithm is that it is usually sensible to τ and σ values, such that a correct reconstruction is given only if their value is well tuned.

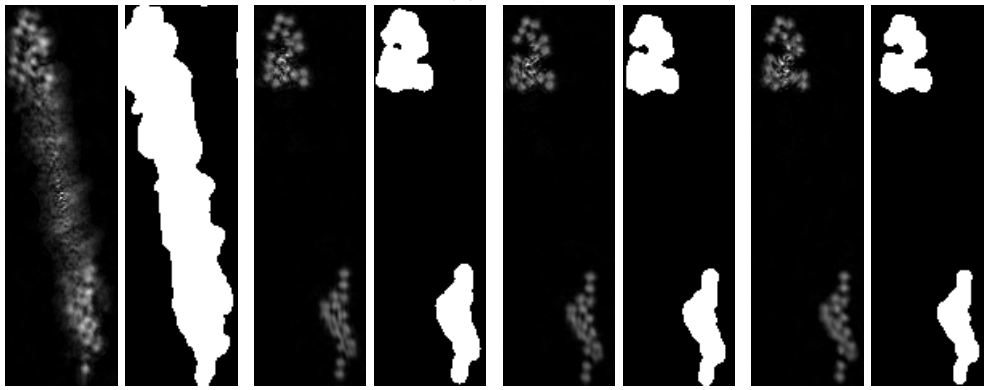
The capability to find the solution with a low, or non-optimal, τ value is all up to the phase retrieval algorithms performance, and in particular to their ability to identify the solution even with a loose or inaccurate support function.

For this reason, MPR has been tested on the same dataset of Marchesini et al. (2003) concerning the golden balls, by studying the result of the reconstruction as function of τ and σ values.

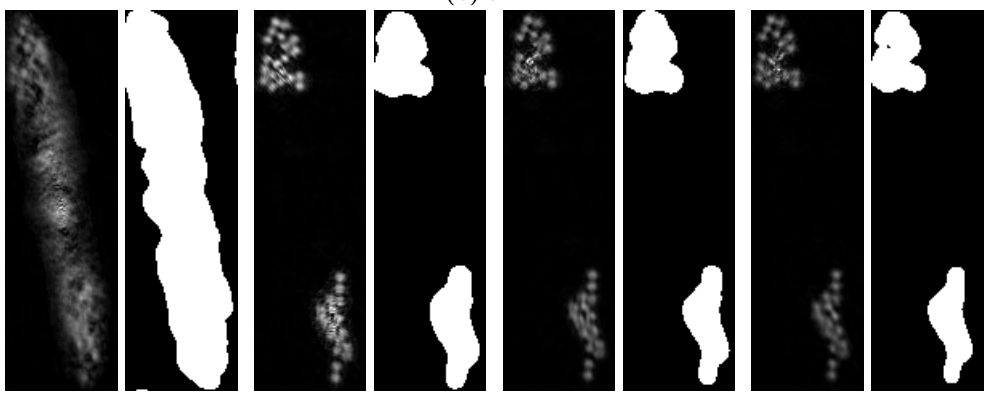
While in the original paper the initial support has been extracted by thresholding the autocorrelation function given by the fourier transform of the diffraction pattern, here the choice is to start from a support function of rectangular shape, with dimensions equal to the half of the autocorrelation extension in the two directions. Thus, Fig.4.1 describes the inputs given to the algorithm. In particular, Fig.4.1a represents the diffraction data in logarithmic scale, where the unknown pixels, i.e. those pixels that must be retrieved along with the density distribution, have been highlighted in red. Fig.4.1b shows instead the initial guess for the support function, which will be refined during the phase retrieval procedure thanks to the Shrinkwrap algorithm. During the reconstruction, ρ was let complex, i.e. its imaginary part wasn't constrained to 0.



(a) $\sigma = 1$



(b) $\sigma = 2$



(c) $\sigma = 3$

Figure 4.2: Results of the *Shrinkwrap* algorithm inside MPR for different τ values, from 5% (on the left) up to 20% (on the right). The test has been performed for different values of σ, from $\sigma = 1$ in (a) to $\sigma = 3$ in (c).

MPR has been used with different combinations of σ and τ, leaving their value unchanged up to the end of the phase retrieval procedure (which differs from the decreasing value for σ in the original paper).

Fig.4.2 shows the results for all the combinations of $\sigma \in \{1, 2, 3\}$ and $\tau \in \{0.05, 0.1, 0.15, 0.2\}$. As it can be well understood from a visual evaluation, there are three cases in which the algorithm was not able to converge towards the correct sample shape. The first two, one in Fig.4.2b left and Fig.4.2c left, didn't reach the solution due to a too much conservative choice of σ and τ. The high value of σ (which smooths a lot the current density estimation) and the low threshold τ (in these cases equal to 0.05) let the support function too inaccurate to guarantee the convergence of the algorithm. On the other side, a too aggressive choice for those parameters, i.e. high τ and low σ, carries to the opposite situation, depicted in Fig.4.2a right, where the support function has been shrank too much.

In all of the other cases, the algorithm finds out the correct sample shape. It's worth giving a comment on the fact that the τ values are particularly different, and run from 0.05 (left column of Fig.4.2) up to 0.2 (right column). This result shows two important features of the Shrinkwrap algorithm inside MPR. The first one is that MPR is particularly ductile with respect to τ and σ values, such that the reconstruction is able to converge also if these parameters assume non-optimal values. The second point is that, thanks to this ductility, a lower threshold value can be used, letting low intensity features emerge, if any.

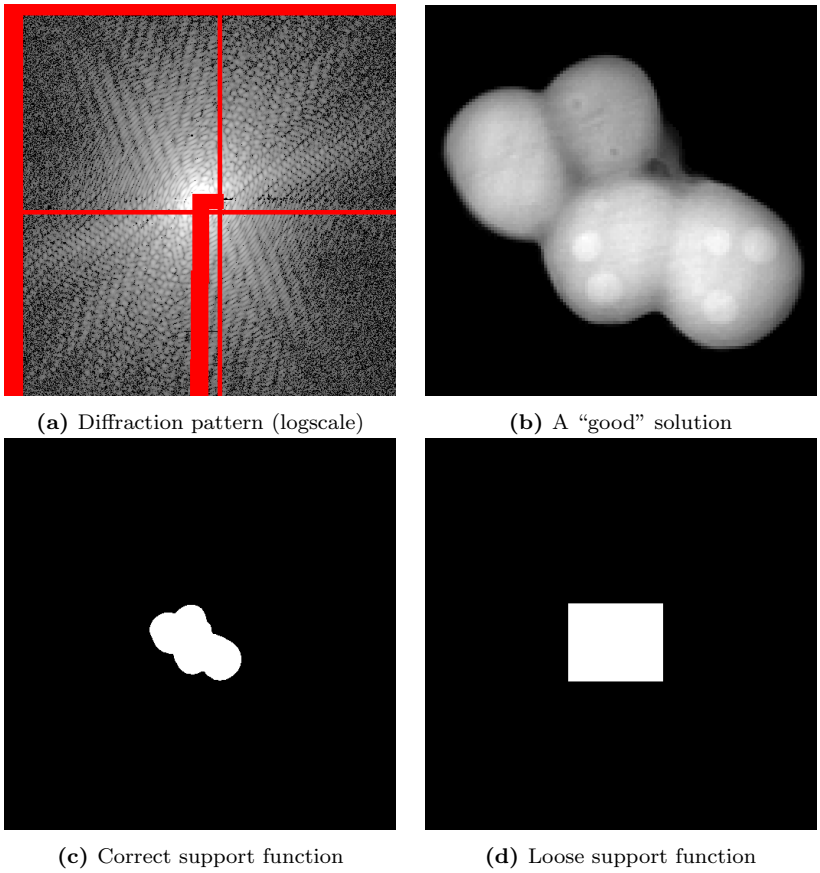
4.1.2 Error Reduction phasing on experimental data

In this section we investigate the performance of the Error Reduction algorithm inside the MPR framework. As it was said in the previous chapter, the Error Reduction algorithm represents, from one side, the only algorithm whose convergence is mathematically ensured, while on the other hand it cannot be used alone, due to its stagnation in local optima. We have seen that, on simulated data, this statement isn't actually true with MPR, due to the fact that its evolutionary dynamic helps the Error Reduction algorithm to escape from local optima, avoiding part of the stagnation problem. In this section we investigate again this situation, now regarding a real X-rays diffraction pattern. The experimental data concerns a biological sample, and in particular a *Deinococcus Radiodurans* bacteria[1].

The diffraction pattern is depicted in Fig.4.3a: as usual, unknown pixels are highlighted in red. Fig.4.3b shows, instead, the solution to the diffraction data in Fig.4.3a. Three different tests will be performed here, exploiting two different support functions, shown in Fig.4.3c and Fig.4.3d. For all of these tests, MPR performance is compared to the one of the standard RPR approach. For each generation, $N_{ER} = 40$ iterations of Error Reduction algorithm are performed on each individual of the population $\mathcal{P}$, whose size is $R = 1024$ for the first two tests and $R = 4096$ for the last one. The first test concerns the phasing of this dataset exploiting the first support function in Fig.4.3c, which has been extracted by applying a 10% threshold to the solution in Fig.4.3b after a gaussian smoothing with a width of 2 pixels. It can be considered as the correct support function, even if, due to the smoothing, it has a resolution two times worse than the one provided by the diffraction data.

Fig.4.4 represents the comparison between MPR and RPR. Fig.4.4a depicts the behavior of the error value (Eq.(3.2)) as function of the generations. Fig.4.4b compares the MPR result (top) and the RPR one (bottom). The difference seen in the error value well reflects the discrepancy between the two approaches concerning the reconstruction quality. This first test clearly shows two facts. First of all, it evidences that the Error

[1]This diffraction data has not yet been published. I acknowledge Yuriy Chushkin, Benoit Maillot, Petra Pernot and Federico Zontone for the availability of this diffraction dataset.



(a) Diffraction pattern (logscale) **(b)** A "good" solution

(c) Correct support function **(d)** Loose support function

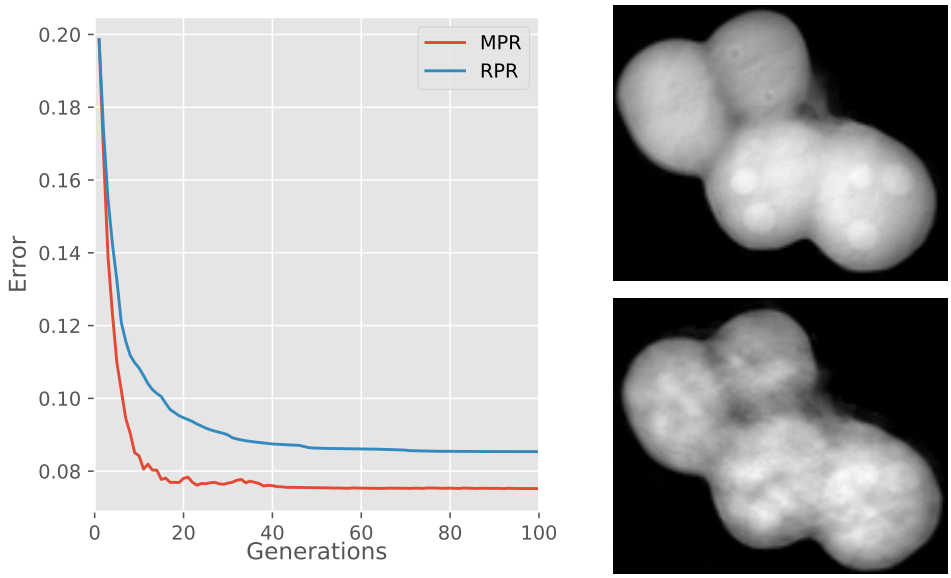
Figure 4.3: Input data for this test (a) and the solution (b). The two support functions in (c) and (d) will be exploited for different tests in this section. In particular, (d) was computed by halving the autocorrelation extension computed from (a).

Reduction algorithm is useless if exploited alone in the standard RPR way: it fails to find the solution also in a relatively simple case with a nearly perfect support function. Second, the stochastic dynamic of MPR is able to make up for the lack of *ergodicity* that tends to stuck the Error Reduction algorithm in local optima of the error functional.

Even if this result may appear suggestive, because it proves that an Error Reduction based phase retrieval is possible, on the other side it may results useless. In fact, there are only few cases where a support function is known a-priori with a decent precision. For this reason, the second test of this section concerns the phasing of the same dataset by the use of the Shrinkwrap algorithm.

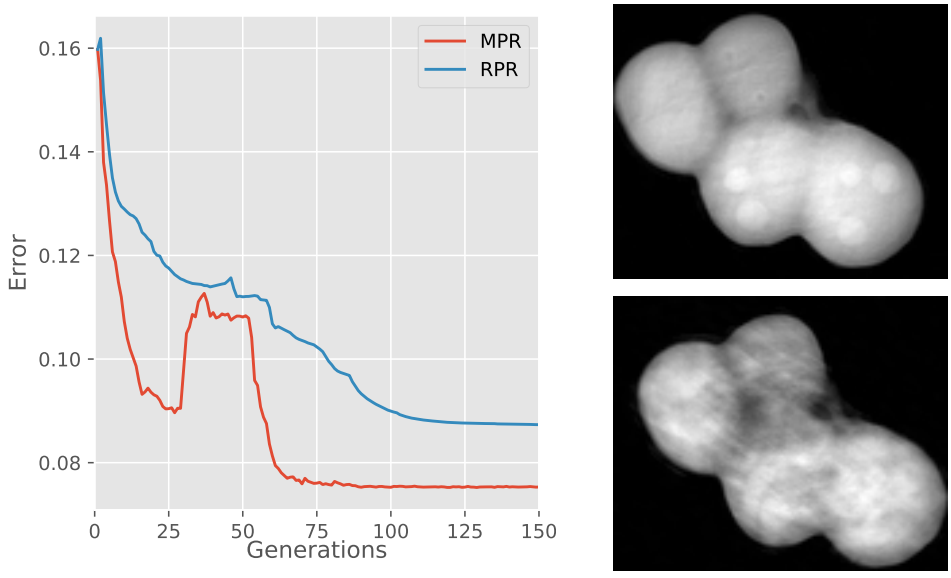
For this test, a loose support function has been given as initial support estimate, depicted in Fig.4.3d. It has been extracted from the autocorrelation function, i.e. the FT of the diffraction pattern, by halving its extension in the two directions. The Shrinkwrap algorithm was tuned with $\sigma = 2$ and $\tau = 0.05$.

The results of this configuration are presented in Fig.4.5. Again, the difference in the final error value, whose behavior is shown in Fig.4.5a, well reflects the difference in the reconstruction quality, as it can be easily observed in Fig.4.5b. Even if the RPR approach



(a) **(b)**

Figure 4.4: Phase retrieval results of MPR and RPR exploiting the Error Reduction algorithm alone, using the exact support function in Fig.4.3c. In (a), the error behavior as function of the generations. In (b), the upper image refers to the MPR result, while the lower one depicts the RPR result.



(a) **(b)**

Figure 4.5: Phase retrieval results of MPR and RPR exploiting the Error Reduction algorithm alone, using the loose support function in Fig.4.3d. The support was updated during the reconstruction via the *Shrinkwrap* algorithm. In (a), the error behavior as function of the generations. In (b), the upper image refers to the MPR result, while the lower one depicts the RPR result.

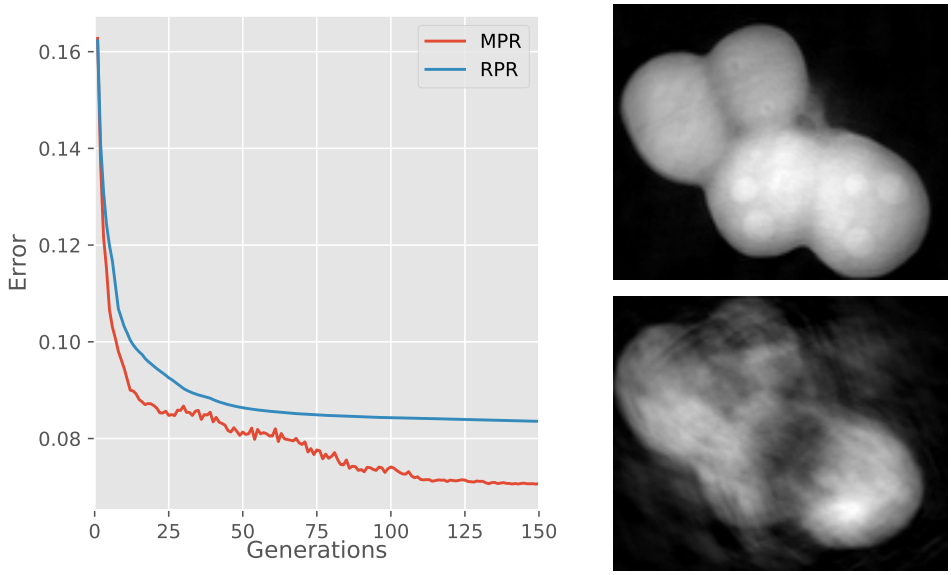


Figure 4.6: Phase retrieval results of MPR and RPR exploiting the Error Reduction algorithm alone, using the loose support function in Fig.4.3d and without updating it. In (a), the error behavior as function of the generations. In (b), the upper image refers to the MPR result, while the lower one depicts the RPR result.

was able to retrieve a nearly correct sample shape, it totally fails again to converge to the solution. The peculiar behavior of the error value in Fig.4.5a, in particular the MPR one, is usual when the Shrinkwrap algorithm is exploited. In fact, when the support function is shrank, the error value tends to grow up as long as the algorithm isn't able to find a good optimum. This result provides the same information as the previous test, adding the fact that the Shrinkwrap algorithm can be exploited, inside the MPR framework, also with the Error Reduction algorithm alone.

A third test on this dataset is performed, exploiting the same square support function as the previous one. The huge difference between this test and the previous one is that now the hard task is to converge to the correct solution starting from the loose support function, but without exploiting the Shrinkwrap algorithm. This mean the the algorithm has to retrieve both the sample density distribution and the zero scattering pixels inside the square support function.

Results of this test are presented is Fig.4.6. As usual, Fig.4.6a represents the error value, while the visualization of the results is provided by Fig.4.6b. As expected, RPR fails to find the solution, and in this case also to correctly retrieve the sample shape, even if some features of the correct shape emerge. The striking result is, instead, the MPR one: MPR succeeded in finding the correct solution, retrieving also the zero scattering values inside the loose squared support function (i.e. the black pixels of Fig.4.6b top), by using the Error Reduction algorithm alone.

4.1.3 Electron Diffraction Imaging data

The very first use of MPR on original experimental data was made on electron diffraction data. The possibility to use electrons for Coherent Diffraction Imaging, which in this case is named Electron Diffraction Imaging (EDI), was experimentally demonstrated by Zuo et al. (2003). While photons scattering function is related to the electron density, electrons diffract basing on the atomic potential. A particular type of EDI is called Keyhole Electron Diffraction Imaging. The peculiarity of this approach is that the zero-scattering region, whose reciprocal is the support function, is obtained with a zero-illumination region by using a confined probe. This method was firstly demonstrated or X-rays by Abbey et al. (2008), while the first demonstration with electrons, i.e. the KEDI approach, was performed by De Caro et al. (2012).

The KEDI approach requires a High Resolution Transmission Electron Microscope (HRTEM) image of the sample of interest and a diffraction pattern of the same sample acquired with the same optical conditions. The real-space HRTEM image of the sample is then exploited in two ways. First of all, it gives an estimation of the support function. Second, it provides low resolution information that can be inserted in the central part of the acquired diffraction pattern, which is affected by saturation and/or the presence of the beam stopper.

The sample of this KEDI experiment is a nanocrystal of $SrTiO_3$, whose crystalline structure is depicted in Fig.4.7a. The sample has been prepared in the [100] zone axis as this configuration makes the atomic columns of different species visible. The HRTEM image in this configuration is provided by Fig.4.7b, while a detail is provided on the right, in Fig.4.7c. In this picture, the atomic columns composed by Sr and Ti are clearly visible, while the weak signal provided by the oxygen columns, which should arise between the Sr and Ti atomic ones, it totally invisible.

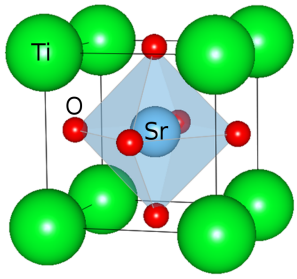
The final KEDI diffraction pattern shown in Fig.4.7d has been obtained, as previously said, by the combination of the measured diffraction pattern with the FT amplitudes of the HRTEM image in Fig.4.7b, after a suitable rotation and scaling. This diffraction pattern represents the experimental constraint for the phase retrieval procedure. The real-space constraint, i.e. the support function, has been extracted by the HRTEM image by applying a threshold to Fig.4.7b, giving the result provided by Fig.4.7e.

The phasing procedure with MPR started from random phases, even if the HRTEM image could provide a first estimation of the phases. The retrieved density is shown in Fig.4.8: the lightness of the image is proportional to $|\rho(\vec{x})|$ while, due to the fact that the density is complex valued, $\arg[\rho(\vec{x})]$ is defined by the hue of the image.

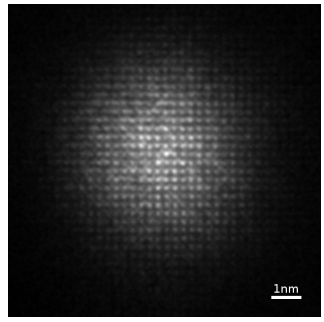
A first comment should be done on the KEDI method: by comparing Fig.4.8 with Fig.4.7b, it is visually evident the striking gain in resolution that KEDI provides with respect to the standard HRTEM image.

The key point of this reconstruction is that it provides, along with a qualitative description of the sample, also important quantitative information. In fact, if the profile of the illumination function is subtracted to the result of Fig.4.8, the result is the projected potential of the specimen, displayed in Fig.4.9a. Fig.4.9b shows, instead, the theoretical projected potential which results to be very close to the MPR result, meaning that Fig.4.8 contains quantitative information.

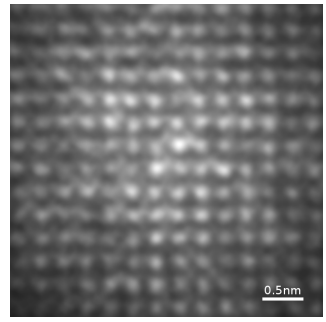
The complexity of this dataset, under the phase retrieval point of view, resides in two points. The first one is that the data is complex, such that both the real and the imaginary part of ρ have to be retrieved. The second issue is the strong symmetry of the support function. As it was said before, for a given diffraction pattern, there are some ambiguities in the identification of the solution, and, in particular, the functions $\rho(\vec{x})$ and $\rho^*(-\vec{x})$



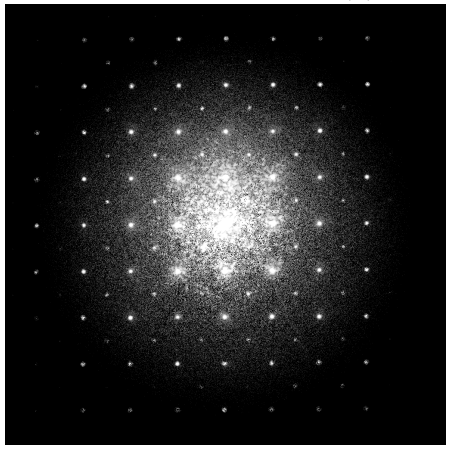
(a) The $SrTiO_3$ cell



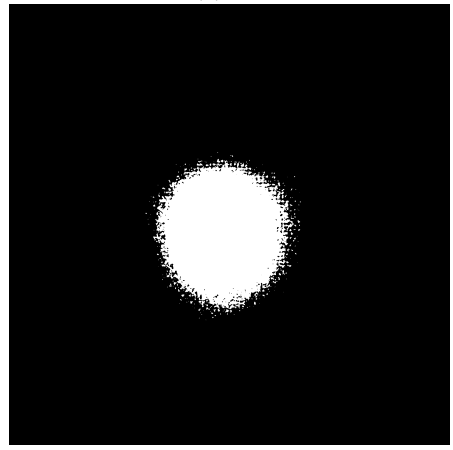
(b) The HRTEM image



(c) A detail of the HRTEM



(d) The KEDI diffraction pattern



(e) The support function

Figure 4.7: $SrTiO_3$ dataset. In (a), the crystalline structure of the sample. In (b), a TEM image of the sample of interest and a detail (c). In (d), the diffraction pattern obtained by combining the electron diffraction pattern and the FT of the TEM image in (b). In (e), the support function provided as constraint to the algorithm, extracted by thresholding the TEM image in (b).

have the same FT modulus, i.e. they are both solutions to the phase problem. If from one side this doesn't represent a problem, because we know that both can be accepted as solutions, on the other hand, if the support function has a nearly centrosymmetric shape, the phase retrieval algorithms tend to stagnate in a superposition of $\rho(\vec{x})$ and $\rho^*(-\vec{x})$ (the so-called twin images problem). Thus, a sample with nearly centrosymmetric shape requires much more effort to be retrieved with respect to the non-centrosymmetric case. If these situations are usually pathological, here is the case, due to the fact that the support function, which is the complementary of the zero-scattering region, is defined by the confined illumination function, which is intrinsically characterized by a quasi-centrosymmetric shape.

4.2 3D imaging

Three dimensional CDI is both conceptually and practically identical to 2D CDI, such that all of the 2D phase retrieval approaches can be trivially adapted to treat 3D diffrac-

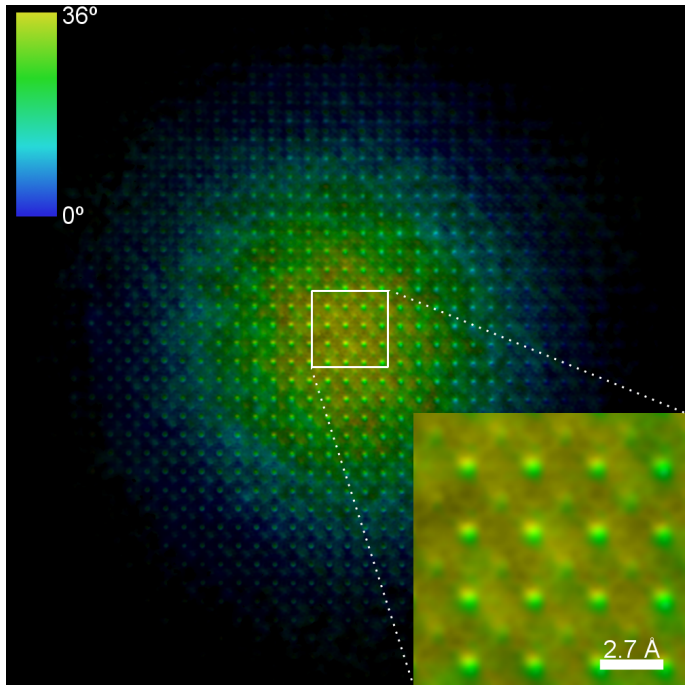
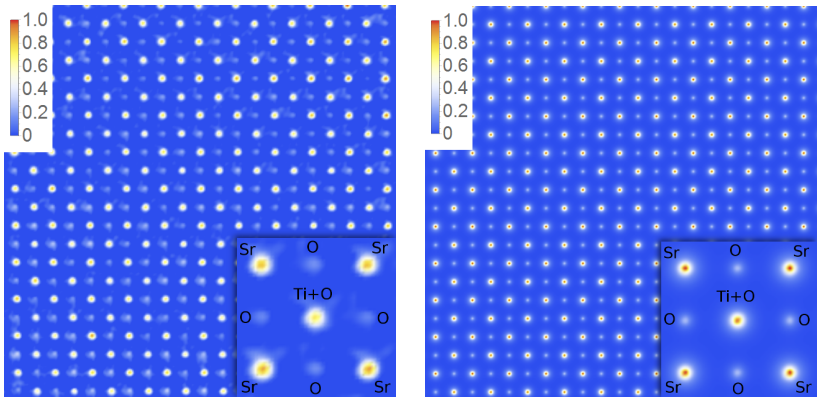


Figure 4.8: The retrieved density distribution of the $SrTiO_3$ nanocrystal. Lightness refers to the density module, while the phase values are highlighted by the hue.

tion data. The difference between 2D and 3D CDI mainly resides in the properties of experimental data. In fact, 3D diffraction patterns aren't practically recorded, but are a combination of many 2D diffraction patterns acquired with different sample orientations, as described, for example, by Miao et al. (2002). Other kinds of 3D acquisition are possible, for example with random sample orientations (for example as done by Ekeberg et al. 2015) recombined by a suitable algorithm (Loh et al. 2010), but they won't be treated in this work.

A typical 3D diffraction pattern that will be faced in this work looks like the one in Fig.4.10.

In this situation two issues arise. The first one concerns the oversampling ratio, because the number of equations is less than the total number of coordinates in the space: this problem is overcome by setting up an experiment with a high oversampling ratio (i.e. with a small object extension). The second issue is, instead, much more tricky to face: 2D diffraction data, as said before, is immersed in the 3D space, representing just a slice. The sample is rotated on an axis, let's say z, such that the acquired patterns are combined in the 3D space with different angles on the xy plane. The maximum sample rotation is less than 180° due to experimental constraints, such that there are two regions, on the xy plane, where diffraction data is totally unknown for every $|\vec{k}|$. This lack of data represents an issue for *iterative projection* algorithms, which fails to correctly estimate those unknown diffracted intensities. Their wrong estimation carries to a reconstruction in the direct space highly affected by artifacts. A common way to avoid artifacts is to perform an average of many reconstructions, but this way tends to suppress also high resolution features of the retrieved sample.



(a) **(b)**

Figure 4.9: A comparison between the retrieved atomic projected potential (a) and the theoretical simulated one (b).

In the following sections some results on this kind of 3D data will be presented. The first dataset will be used to describe these artifacts and how they can be mitigated thanks to the MPR approach.

4.2.1 Vaterite microsphere: facing artifacts

Vaterite is one of the three stable crystalline forms of $CaCO_3$ (calcium carbonate). Vaterite can be obtained from amorphous calcium carbonate, and usually presents spherical morphology with high porosity. The treated dataset, along with a detailed description of the sample, are described by Cherkas et al. (2017).

Issues arising from this diffraction data derives, as said before, from the unknown amplitudes in some directions. The issue can be easily understood if we look at Fig.4.11. This figure is a slice of the 3D diffraction pattern and it well depicts the lack of data which concerns two opposite angles of about 20° degrees on the xy plane, along with an area in all directions with $\sqrt{k_x^2 + k_y^2} > K_{xy}^{\max}$.

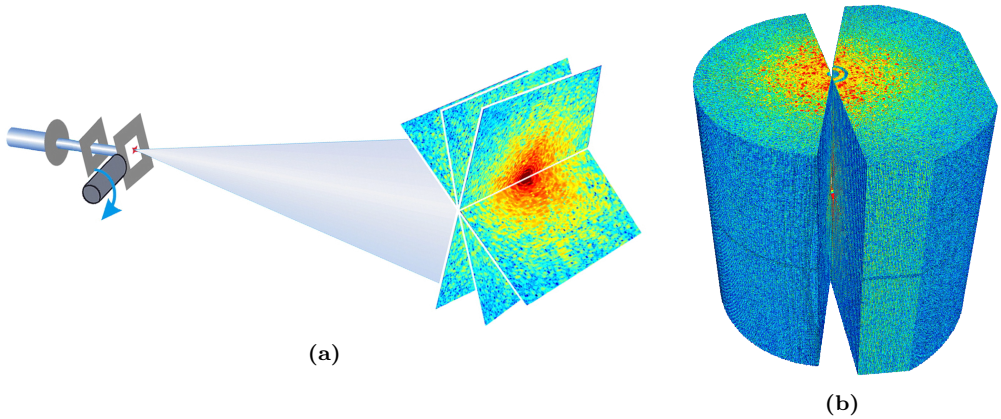
If a "brute force" phasing approach is tempted, i.e. a reconstruction is performed without caring about those missing values, the result is highly affected by artifacts.

Fig.4.12 shows this result. In particular, Fig.4.12a represents the projection (top) and a cut (bottom) on the z axis. The same is done for the y axis in Fig.4.12b and for the x axis in Fig.4.12c. In these reconstruction two kinds of noise are clearly visible:

1. **High resolution noise.** It derives from the unknown amplitudes greater than $K_{xy}^{\max}$ on the xy plane.

2. **Middle resolution noise.** It derives from the unknown diffraction data at middle-low resolutions on the xz plane.

These two kinds of noise can be well distinguished in the three slices of Fig.4.12. It is worth noting that the central xz plane of the diffraction data, i.e. the central cut along the y axes, contains the integral value in the y direction in the direct space. This means that a wrong estimation of those pixels carries to a bad estimation of the pixels value in



(a) **(b)**

Figure 4.10: A pictoric representation of the 3D data acquisition procedure (a): the sample is rotated, and the acquired bi-dimensional diffraction data is then combined to provide a three dimensional diffraction pattern (b). Due to limitations in sample rotation, which is less than 180 degrees, there are two opposite wedges of missing data clearly visible in (b).

Fig.4.12b (top) which directly reflects into striped artifacts along the y axes well visible in Fig.4.12a and Fig.4.12c.

The bad estimation of these missing pixels directly derives from the fact that they are not constrained during the phase retrieval process. For this reason, iterative algorithms, in order to lower the error value based on known diffraction values, tends to modify the unknown amplitudes to better adapt to the known ones. The effect of this adaptation is well shown by Fig.4.13a, which is the FT of Fig.4.12a (top) and represents the retrieved version of the diffraction slice in Fig.4.11. If we look at the two radial profiles in Fig.4.13b, computed in the directions highlighted in Fig.4.13a, it appears evident that, over about 60 pixels of distance from the center, the two radial profiles, which should theoretically appear similar, assume values that differs by up to 2 orders of magnitude (please note the logscale on the y axis).

At this point it is clear that a solution to this amplitudes overestimation must be found. The common way is to perform many reconstructions and then average the results, but this way leads to a degradation in the achieved resolution, as discussed in Section 1.6, due to a general underestimation of $|\tilde{\rho}^{\text{aver}}(\vec{k})|$ for high $|\vec{k}|$.

A good strategy to overcome this overestimation issue is to set an upper bound for the values of those unknown amplitudes. A common strategy, exploited in crystallography, is to derive a radial profile by interpolating the known amplitudes, an then use it as a constraint for the unknown ones. If this method may result reliable for crystallographic imaging, the case of CDI is much more complex due to the fact that the radial profile shape usually differs from a direction to an other (it derives from the non-periodic nature of CDI samples). Thus, this upper bound must be extracted in a different way.

The key idea for facing artifacts inside the MPR approach is to exploit the information given by the population of candidate solutions $\mathcal{P}$. In particular, the average density ρ^{aver} of the population can be computed by simply averaging the values for each pixel, as described in section 1.6. As said before, ρ^{aver} will tend to provide an underestimation of the amplitudes. Another way to perform the average is to compute the average amplitudes

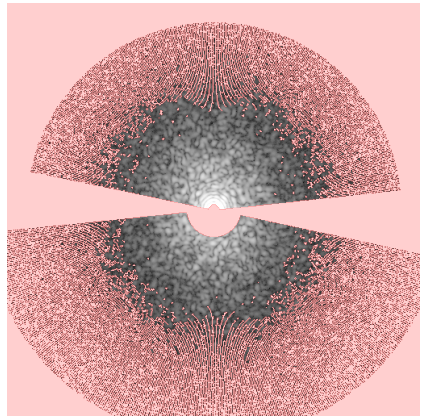


Figure 4.11: A cut of 3D diffraction data on the xy plane. The two missing wedges are clearly visible, along with some lacks of data at high resolution and totally unknown pixels near the image edges.

M^{aver} in the following way:

$$M_{i,j}^{\text{aver}} = \frac{1}{R} \sum_{r=0}^{R-1} \left| \tilde{\rho}_{i,j}^{r} \right| . \tag{4.1}$$

From one side, $|\tilde{\rho}_{i,j}^{\text{aver}}|$ usually represents an underestimation also when $|\tilde{\rho}_{i,j}^{R}|$ is overestimated of each R (i.e. for each individual of the population $\mathcal{P}$. On the other side, in the same situation, the value given by Eq.(4.1) is an overestimation of that unknown amplitudes. After these considerations, the following relation is likely to be true:

$$|\tilde{\rho}_{i,j}^{\text{aver}}| < M_{i,j}^{\text{true}} < M_{i,j}^{\text{aver}} \tag{4.2}$$

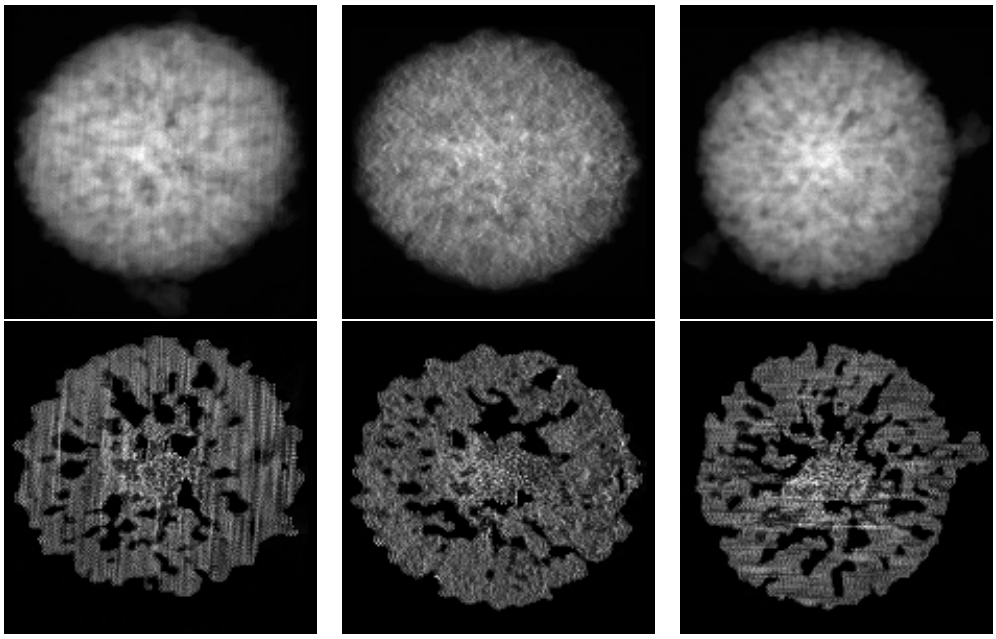
where M^{true} represents the true values of the unknown amplitudes that have to be retrieved.

Even if a reasonable bound for the unknown amplitudes is given, the gap between $|\tilde{\rho}_{i,j}^{\text{aver}}|$ and $M_{i,j}^{\text{aver}}$ tends to be of some orders of magnitude, and a stronger constraint is needed. Looking at the features of $|\tilde{\rho}_{i,j}|$, it's worth noting that:

- the value of $|\tilde{\rho}_{i,j}|$ can vary of some orders of magnitude from an individual of the population to an other when $M_{i,j}$ is unknown and must be retrieved
- the intensity profiles, as the one depicted in Fig.4.13, have a linear or nearly linear behavior in logarithmic scale
- intensity values are strictly positive.

These considerations suggest that a clever way to compute the mean value for the amplitudes could be to exploit the geometric average, instead of the arithmetic one as done in Eq.4.1. The geometric mean of the amplitudes can be computed as follows:

$$M_{i,j}^{\text{aver}_G} = \left(\prod_{r=0}^{R-1} \left| \tilde{\rho}_{i,j}^{r} \right| \right)^{\frac{1}{R}} = \exp \left[\frac{1}{R} \sum_{r=0}^{R-1} \log \left(\left| \tilde{\rho}_{i,j}^{r} \right| \right) \right] . \tag{4.3}$$



(a) xy plane **(b)** xz plane **(c)** yz plane

Figure 4.12: MPR reconstruction of the vaterite sample. A projection (above) and a slice (below) are shown for each plane. Two kinds of artifacts are visible: the high resolution noise derives from the unknown pixels at high $|\vec{k}|$ in the diffraction data, while stripes are caused by the two wedges of missing data.

The geometric mean is commonly exploited with phenomena that present proportional growth or exponential behaviors, and expresses the typical value of a set of numbers that differs from each others by orders of magnitude, i.e. a situation very similar to our " missing amplitudes" issue. A strong relationship between arithmetic and geometric mean exists, that is $M_{i,j}^{\text{aver}_G} \leq M_{i,j}^{\text{aver}}$, where the equality holds only when the averaged items have all the same value (which is not our case).

Again, if we expect that $|\tilde{\rho}_{i,j}^{r}|$ represents an overestimation for the unknown amplitudes for every, or almost all, the individuals inside $\mathcal{P}$, thus this new relationship tends to be true:

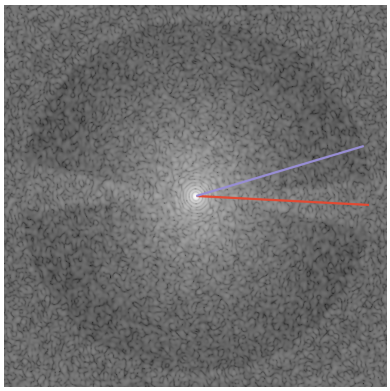
$$|\tilde{\rho}_{i,j}^{\text{aver}}| < M_{i,j}^{\text{true}} < M_{i,j}^{\text{aver}_G} \tag{4.4}$$

It worth noting that, thanks to the geometric mean, the upper bound can be some orders of magnitude less than the upper bound of Eq.(4.2). At this point, Eq.(4.4) implies that it must exist a constant C_B such that:

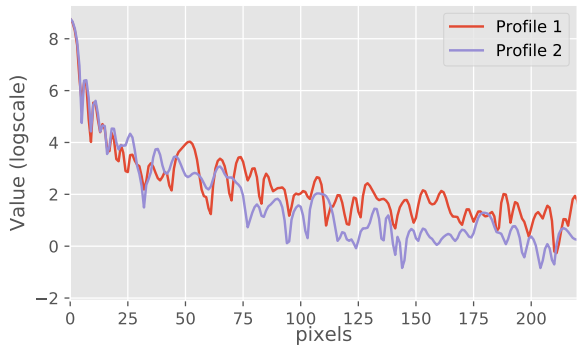
$$M_{i,j}^{\text{true}} < C_B \cdot |\tilde{\rho}_{i,j}^{\text{aver}}| + (1 - C_B) \cdot M_{i,j}^{\text{aver}_G} = M_{i,j}^{\max} \ \forall\, i, j \tag{4.5}$$

Eq.(4.5) doesn't have to be intended as a strong mathematical relationship, but as something which is semi-quantitative, semi-empiric and it is likely to be true for all the coordinates where the experimental amplitudes are unknown.

Under the implementation point of view, two more steps are required with respect to what has been described in Chapter 2. First of all, a function that computes the matrix $M_{i,j}^{\max}$ is needed. Its implementation is trivial and won't be discussed. The function



(a) The FT of Fig.4.12a (top)



(b)

Figure 4.13: A slice of the FT amplitudes of the reconstruction in Fig.4.12 (a) and the two intensity profiles in logarithmic scale (b) extracted from two adjacent radii. The overestimation of the amplitudes in the unknown areas are the cause of the reconstruction artifacts, well visible in Fig.4.12.

that computes $M^{\max}_{i,j}$ must be inserted between the *Self-Improvement* and the *Crossover* steps. A second point concerns the use of this matrix $M^{\max}_{i,j}$. This bound may be applied to all the unknown amplitudes. However, if it is done, given $M^{\max(n)}_{i,j}$ the upper bound computed at the n-th generation, the relation $M^{\max(n+1)}_{i,j} < M^{\max(n)}_{i,j}$ holds, pushing down to zero the unknown values.

For this reason, it is convenient that only a fraction of this upper bound matrix is used, such that each individual ρ^r has its own one, $M^{\max r}$, computed at each generation as follows:

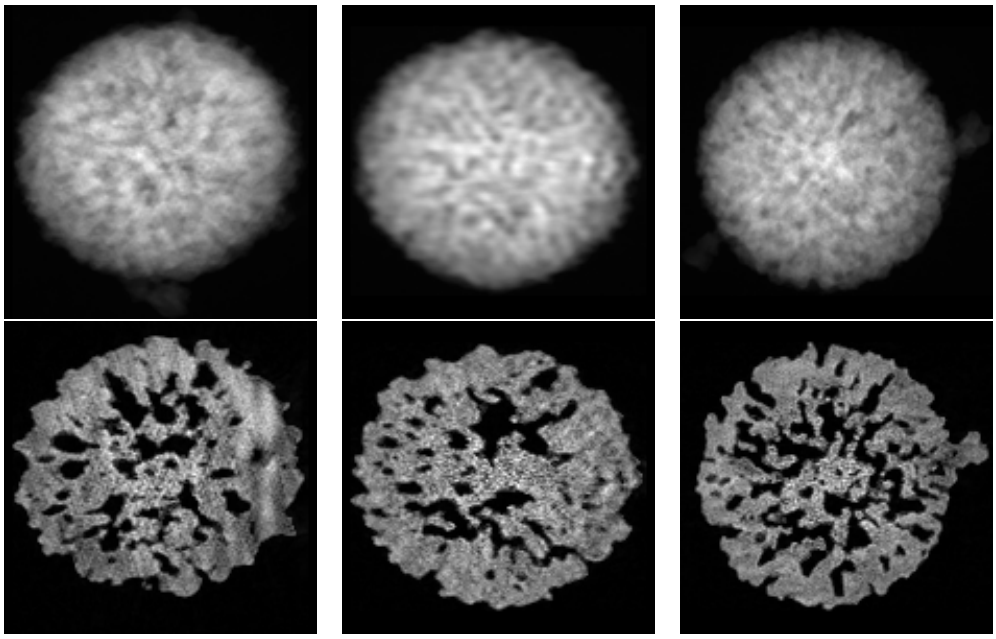
$$M^{\max r}_{i,j} = \begin{cases} M^{\max}_{i,j}, & \text{if rand}[0,1) < C_{BP} \\ -1, & \text{otherwise} \end{cases} \tag{4.6}$$

where $M^{\max r}_{i,j} = -1$ means that those amplitudes, if unknown, will be left untouched.

For example, $C_{BP} = 0.2$ means that only one fifth of the unknown amplitudes will be bounded to a maximum value. The choice of the bounded coordinates is totally stochastic, and differs from an individual ρ^r to an other. In this way, even if an unknown amplitude at a given coordinate is not directly constrained to a maximum value, it will likely have

Algorithm 22 Implementation of the projector $P_{\mathcal{M}}$ with the upper bound for unknown amplitudes

```
1:  function PM(ρ, M^max[i, j])
2:      ρ̃ ← DFT(ρ)
3:      for i, j ← 0 to N do
4:          if M[i, j] is known then
5:              ρ̃[i, j] ← M[i, j] ∗ exp(√−1 ∗ arg(ρ̃[i, j]))
6:          else if M[i, j] is unknown and M^max[i, j] ≥ 0 and M[i, j] > M^max[i, j] then
7:              ρ̃[i, j] ← M^max[i, j] ∗ exp(√−1 ∗ arg(ρ̃[i, j]))
8:          end if
9:      end for
10:     ρ ← DFT^−1(ρ̃)
11:     return ρ
12: end function
```



(a) xy plane **(b)** xz plane **(c)** yz plane

Figure 4.14: Result of the MPR reconstruction using the described bounds for the unknown diffracted intensity. A projection (above) and a slice (below) are shown for each plane. Both stripes and high resolution noise are now only slightly visible.

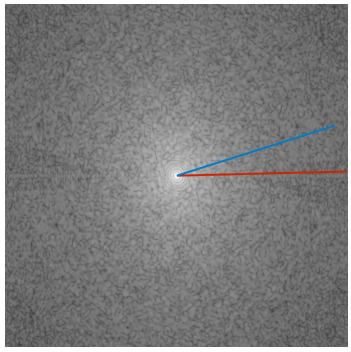
a bounded neighbor. This situation prevents its value from rising too much during the iterative projection procedure.

The third necessary step is the modification of the iterative projection algorithms, such as ER and HIO, concerning in particular the projection operation described by Alg.1. The new version of this projector is proposed in Alg.22.

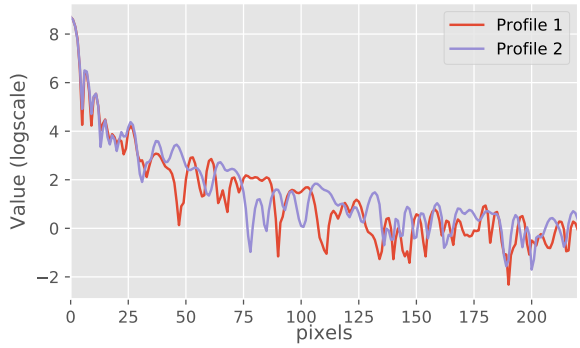
Just to sum up, we have seen that a good way to find an upper bound for the unknown amplitudes is to perform a linear combination of the amplitude of the average density, $|\rho_{i,j}^{\text{aver}}|$ and the average amplitude, $|M_{i,j}^{\text{aver}_G}|$ computed by performing a geometric mean. This upper bound, $M_{i,j}^{\max}$, is exploited by iterative projection algorithms with a probability C_{BP} for each pixel.

An important role in this approach is played by the *Mutation* operator inside MPR. In fact, without a mutation operation, the evolutionary dynamics of the approach tends to collapse to a unique genetic pool, i.e. all the individuals inside the population will be very close in the space of configurations and tends to be very similar. In such a situation, the described approach to reduce artifacts is doomed to fail, due to the fact that, if individuals are very similar and tends to overestimate unknown amplitudes, also $|\tilde{\rho}_{i,j}^{\text{aver}}|$ will provide an overestimation of them, making Eq.(4.5) no more valid also under a qualitative point of view. The role of mutation is, thus, to keep the population reasonably sparse in the configuration space, such as the bounds of Eq.4.5 assume reasonable values.

After this description, it is now possible to show the results obtained by exploiting this method. In this case, $C_B = 1$ and $C_{BP} = 0.5$. Parameters concerning mutation have been set to $C_{MP} = 0.33$, $C_{MP} = 0.8$ and $C_{MP} = 1$. A population $\mathcal{P}$ of $R = 512$ individuals has been used, and, during the *Self-Improvement* step, each individual underwent 30

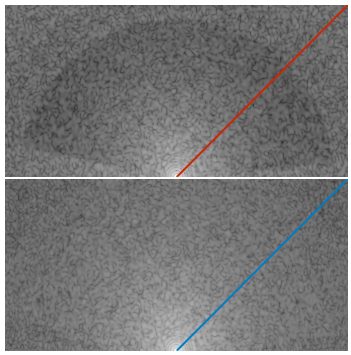


(a) The FT of Fig.4.14a (top)

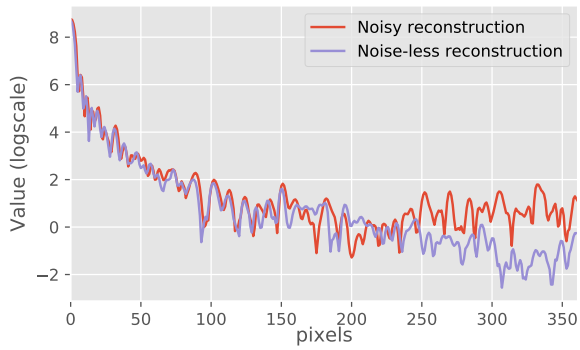


(b)

Figure 4.15: A slice of the FT amplitudes of the reconstruction in Fig.4.14 (a) and the two intensity profiles in logarithmic scale (b) extracted from two adjacent radii in (a). The two radial profiles are now much more compatible.



(a)



(b)

Figure 4.16: A visual comparison between the same slice for the two different reconstructions (a) in Fig.4.12 and Fig.4.14. The two radial profiles, computed in the same direction, are plotted in (b), underlining that the bounds for the amplitudes extracted during the MPR reconstruction works well not only for the missing wedges but also for the missing high resolution amplitudes.

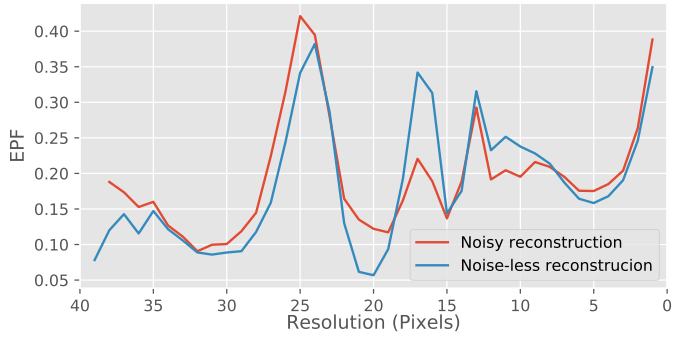


Figure 4.17: Comparison of the Error Profile Function (EPF) between the noisy unbounded reconstruction of Fig.4.12 and the noise-less one in Fig.4.14. The EPFs are pretty similar, which means that the amplitude bounding didn't worsen the achieved resolution.

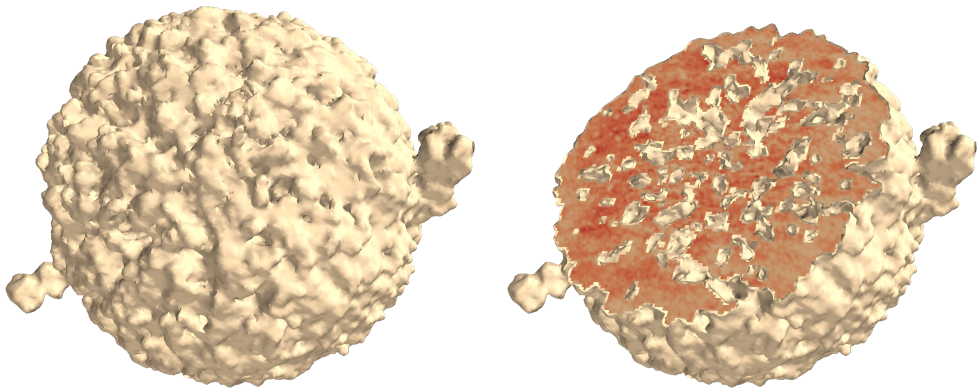
iterations of HIO algorithm.

The retrieved 3D density is depicted in Fig.4.14. First of all, vertical stripes that dominate Fig.4.12a are now almost absent, as Fig.4.14a clearly shows. The same happens for Fig.4.14c with respect to Fig.4.12c, where horizontal stripes dominates the reconstruction. Another big gap in quality between the two reconstructions concerns the xz plane. As said before, most of the artifacts derive from the lack of diffraction data in the central slices of the y axis. This fact reflects into a bad estimation of pixel values in Fig.4.12b. Fig.4.14b, instead, clearly shows a reconstruction much less affected by artifacts.

The reason of this reconstruction quality is well described by Fig.4.15. While Fig.4.13b clearly showed an overestimation of amplitudes where diffraction data lacks, here only a slightly visible difference between the areas of known and unknown amplitudes is present. Under a visual point of view, Fig.4.15a shows a slight underestimation of unknown amplitudes. Fig.4.15b depicts two adjacent radial profiles of diffraction data in logarithmic scale, the blue one computed in a known amplitudes region and the red one in the unknown region, as previously done in Fig.4.13. While Fig.4.13 showed that unknown amplitudes were overestimated of more than an order of magnitude, here the two radial profiles are much more compatible.

The same behavior happens for unknown values at high resolution, which were responsible of high resolution artifacts in Fig.4.12, due to a clearly visible overestimation in Fig.4.13b at very high $|\vec{k}|$. Fig.4.16a shows this difference: as well depicted in the plot in Fig.4.16b, starting from a distance of about 240 pixels from the center, the amplitudes along the radial profile of the reconstruction in Fig.4.12 start to be overestimated, assuming values also greater than the ones between 100 and 240 pixels (red line). The blue line, which represents the radial profile of the reconstruction in Fig.4.14, present a much better behavior, with a tendency to decrease with $|\vec{k}|$, assuming values also two orders of magnitude lower than the red profile.

A last point about this dataset concerns some considerations about the achieved effective resolution. As said before, the only way to have its quantitative evaluation it to perform a PRTF function, but its evaluation for a MPR result requires a huge amount of



(a) **(b)**

Figure 4.18: A 3D rendering of the result shown in Fig.4.14. In (a), the whole reconstruction is shown, while (b) depicts a cutted version of (a), where the internal density distribution is revealed.

computational resources. What can be done is, instead to evaluate an EPF comparison between the result in Fig.4.12 and the artifacts-reduced one in Fig.4.14, in order to evaluate if and how the artifacts reduction method affects the reconstruction quality. In fact, high resolution artifacts could be mitigated for example by using a gaussian filter, which has the drawback to negatively affect the resolution of the image.

Fig.4.17 shows the EPF comparison between the "noisy" and "noise-less" reconstructions. As the plot clearly shows, artifacts reduction does not affect the resolution, i.e. the two EPF reconstructions has a comparable behavior at all resolution scales, and in particular in the range 5-1 pixels that represents the typical scale of artifacts in Fig.4.12.

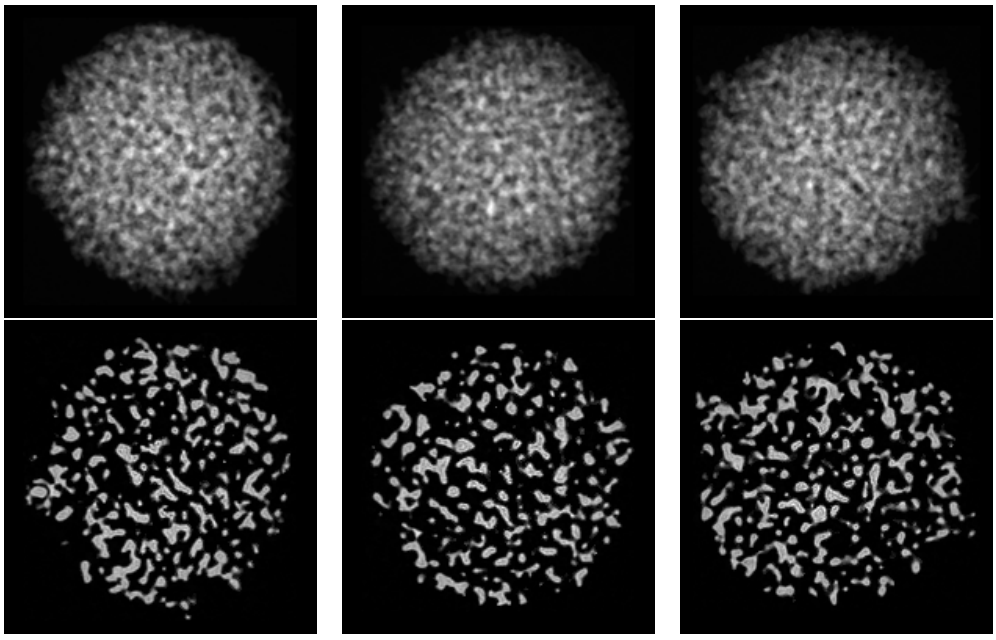
Fig.4.18 is a 3D rendering of the result presented in Fig.4.14. In particular, Fig.4.18a is the rendering of the whole reconstruction, while Fig.4.18b is a cut on the yz plane, revealing the density distribution inside the sample.

The following subsection deals with a further dataset, acquired with the same technique and presenting the same issues.

4.2.2 Silicon dioxide microparticle

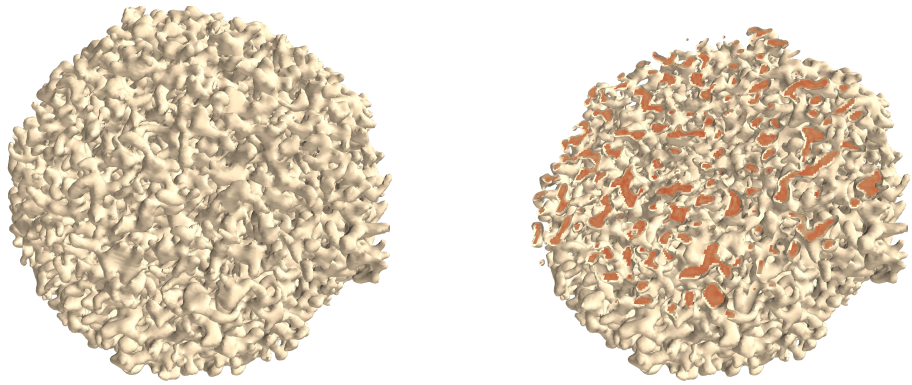
This dataset has features that are somehow similar to the previously treated Vaterite one, concerning both the issues of diffraction data and the sample shape. The dataset is a 3D diffraction data of a SiO_2 microparticle.

Diffraction data has almost the same features of the previous dataset, and a slice of this diffraction data is very similar to the one of the previous dataset in Fig.4.11. In fact, the data acquisition was performed in the same way, and the issues that concerned the previously presented dataset appear again for this diffraction data.



(a) xy plane **(b)** xz plane **(c)** yz plane

Figure 4.19: Result of the MPR reconstruction using the described bounds for the unknown diffracted intensity. A projection (above) and a slice (below) are shown for each plane.



(a) **(b)**

Figure 4.20: A 3D rendering of the result shown in Fig.4.19. In (a), the whole reconstruction is shown, while (b) depicts a cutted version of (a), where the internal density distribution is revealed.

Fig.4.19 reports the projections on the three axes (in the upper part) and slices (in the lower part) of the result obtained by MPR. The method to treat unknown amplitudes, described in the previous example, has been used. As consequence, there are no visible artifacts at high resolution (which may affect Fig.4.19b) nor vertical stripes in Fig.4.19a or horizontal stripes in Fig.4.19c.

A 3D rendering of the reconstruction is, instead, shown in Fig.4.20.

A further comment can be provided about the comparison between MPR and the standard RPR approach. As seen before, performing a reconstruction on this kind of datasets without caring about the overestimation of unknown amplitudes leads to a reconstruction plenty of artifacts, as shown for the previous data in Fig.4.12. The usual way to face these artifacts with the RPR approach is to consider as solution the average of many independent reconstructions, such that, thanks to the averaging, the artifacts are smoothed. The drawback of this approach is that a resolution degradation usually occurs. Fig.4.21 shows a comparison between the MPR result (Fig.4.21a) and an averaged RPR result (Fig.4.21b) concerning a reconstruction detail on the yz plane. Even if, at a first look, the two reconstructions look almost identical, a deeper inspection of data reveals that 4.21b is smoother than Fig.4.21a. This fact is quantitatively remarked by the profile shapes plotted in Fig.4.21c. Looking at the edges of the profile, it can be noted that the density profile extracted by MPR (red line) drops down faster than the RPR result (blue line). This *edge analysis* suggests that the resolution achieved by MPR could be slightly higher than the RPR one.

A further clue about the better performance of MPR in this case is provided by the computation of the retrieved average density value. In fact, This sample is totally composed by amorphous SiO_2, whose tabulated density is 2.196 $\mathrm{g\,cm^{-3}}$. The density value can be extracted by the reconstructions by computing the histogram of the pixels, i.e. the number of occurrences as function of the density interval. Then, the histogram is interpolated with a normal distribution and the mean provided by the interpolation is assumed to be the sample density. This method carries to an estimation of the density value of 2.259 $\mathrm{g\,cm^{-3}}$ for the RPR approach, which represents an overestimation of the real value of about 2.7%. MPR provides instead a value of 2.172 $\mathrm{g\,cm^{-3}}$, which represents an underestimation of 1.1%. The information about average sample density is mainly

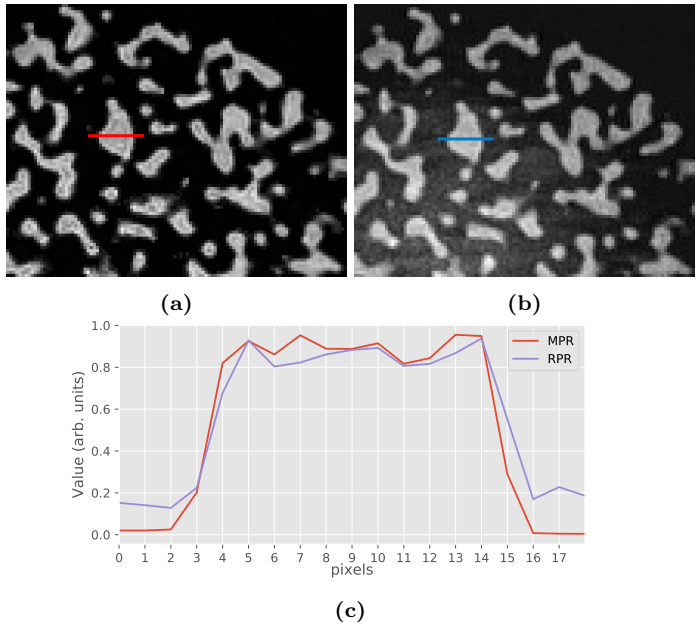


Figure 4.21: A detail of the MPR reconstruction (a) and the *averaged* RPR one (b). The lower definition of (b) is remarked by the density profile analysis reported in (c).

contained in the very central pixels of the diffraction pattern, which are hidden by the beam-stopper and must be retrieved by the algorithm. Thus, we can state that MPR provides a better performance not only on high resolutions, but also in retrieving very low frequencies missing from diffraction data.

Chapter 5

Data Analysis Software

Even if this dissertation is, in the most part, dedicated to the main topic of the PhD research, some other works have been made in the field of Coherent Diffraction Imaging, all concerning CDI data analysis. This chapter is fully dedicated to a description of data analysis and visualization softwares. All diffraction patterns shown in the chapter, exploited only as benchmark for the softwares, were acquired at the Free Electron Laser FERMI[1] by Langbehn et al. (2018).

5.1 Online data analysis tool

The data analysis software has been designed for pump-and-probe experiments. A typical pump-and-probe experiment is used to get information about ultrafast phenomena. The sample of interest is hit by a pump pulse that generates a modification of the sample. After a defined time delay, the probe pulse hit the sample and the diffraction data is acquired. Diffraction data gives information on the sample as function of the time delay between the pump and the probe beams. The main feature of this technique is that it is able to investigate ultrafast phenomena (usually excited states of the sample) without requiring a fast detector, and the resolution in time is only limited by the pulses duration. Samples are contained in a chamber and are delivered to the interaction zone thanks to a pulsed valve. A common experimental configuration is well described by Rupp et al. (2014).

This data analysis tool was developed for a resonant two-color imaging of helium nano-droplets at FERMI FEL. The possibility to have an online analysis of incoming data is a valuable help in tuning the experimental parameters, because the effects of their modification can be directly visualized on the acquired diffraction data. For example, gas phase clusters are produced thanks to a supersonic expansion in the experimental chamber after the valve opening. Their size directly depends on the temperature and pressure of the He inside the chamber (Hagena & Obert 1972) and on the time delay between the He cluster source and the FEL X-ray pulse (Rupp et al. 2014). Thus the success of such an experiment highly depends on the correct tuning of these (and many others) experimental parameters.

The online data analysis tool has been developed to have a live view on experimental results, allowing an easier and faster tuning of sensible experimental parameters. The online analysis software is divided into two parts:

- Analysis software: data acquired by the detector is sent to the online storage. As soon as data is saved, the analysis software reads the files and assigns to each diffraction shot a set of values which describes its features.

[1] `https://www.elettra.trieste.it/lightsources/fermi.html`

- Visualization software: it plots the values assigned by the analysis software and allows to directly access each diffraction pattern.

The analysis software is, technically speaking, a *daemon* software, i.e. a software that runs in the background. On the other side, the visualization software runs in foreground and it can be interactively controlled via an user interface. The following subsections will describe in detail these two tools.

5.1.1 Analysis software

Data is saved on the storage in the HDF5[2] file format. Each file contains, in this case, 100 diffraction patterns with a dimension of 1035x1035 pixels. The acquisition rate of the camera, which is synchronized with the FEL repetition rate, is 50Hz, such that a file with 100 diffraction patterns is saved to the storage every 2 seconds[3]. The role of this tool is to assign to each diffraction pattern, which is uniquely identified by the *bunch id*, a set of values which describes its main features.

The main loop of the software is provided by the following pseudo-code.

Algorithm 23 The main loop of the analysis software

```
1:  function AS
2:      P ← ReadParameters                    ▷ A set of parameters is read from a configuration file and saved to the object P
3:      loop
4:          FileName ← WaitForFile(P.work_dir)   ▷ Wait for a new file. When a new file appears in the work directory, its file name is returned and saved
5:          Data ← ReadFile(FileName)         ▷ Diffraction data is read from the file and stored into Data
6:          Hits ← HitOrMiss(Data, P)         ▷ From the Data array, only diffraction data containing meaningful measurements are selected and stored into the array Hits
7:          Analyze(Hits, P)                  ▷ Each pattern is analyzed
8:          SaveAnalysis(Hits, P)             ▷ Values extracted form the analysis are saved into a separate file
9:      end loop
10: end function
```

The implementation makes use of two different data structures. The first one is the object P, that contains the parameters for the analysis tool. Without listing all of its data members, it is worth noting that, for example, P.work_dir is the directory where the software search new data to analyze, while P.analysis_file_name is the file name where the results of the analysis are saved.

The second data structure is, instead, the one concerning diffraction data. The array Data, declared at line 5 of Alg.23 is an array of objects. Each element in this array has two members: the first, Data[p].pattern, is a $N \times N$ array containing the diffraction data acquired by the detector, such that Data[p].pattern[i][j] is the pixel value at coordinates (i, j) of the p^{th} pattern among the 100 ones inside the loaded file. The second member of this object is Data[p].bunch_id, which contains the identification number of that pattern (it is actually the identification number of the FEL flash that produced the diffraction).

The first operation is to read the needed parameters from a configuration file. This configuration file contains the reference to the working directory, the file name where the analysis results are saved, and some other parameters that will be subsequently described.

Once that parameters have been read, the algorithm enters the main loop (line 3 of Alg.23). The first called routine at line 4 has the role to scan the working directory in

[2]For more information, visit `https://www.hdfgroup.org/`

[3]These parameters refer to the LDM beamline at FERMI

search for a new file to analyze. This function query the file system with a given frequency: when a new file appears in the directory, its name is returned and it is added to the list of analyzed data files.

The function at line 5 reads the data file. Its implementation is trivial, apart for the check of file integrity. In fact, when the file appears of the filesystem, the data transfer has just started. This function has to wait for the end of data transfer. This is achieved by iteratively calling the `H5Fopen` function of the HDF5 library as long as its return value reports an error. When the file transfer is complete, it passes the integrity check and data is read by the software.

The first step in data analysis is performed by the HitOrMiss function, called at line 6 of Alg.23. The aim of this routine is to select, among the diffraction patterns loaded from the file, the meaningful ones, i.e. the ones containing a diffraction data due to the presence of a sample in the interaction zone. The discrimination between a hit and a miss is based on the total diffracted radiation recorded by the detector, i.e. the sum over all pixel values. The patterns whose intensity is above a given threshold can be considered as hits, while the ones below the threshold are discarded. The a-priori knowledge of this threshold value is all but trivial, because it depends on the experimental parameters. Thus, the threshold value is found by performing a statistical analysis of diffraction patterns, by following the steps below:

1. The total diffracted radiation is computed and saved for each diffraction pattern, that is $\sum_{i,j} \text{Data}[p].\text{pattern}[i,j]$ is saved to $\text{Data}[p].\text{sum}$.
2. Data is sorted basing on the sum.
3. The mean S and standard deviation σ_S of the sum value are computed on the 50% less intense patterns
4. Only diffraction patterns whose sum value is greater than $S + \text{P.nsigma} \cdot \sigma_S$ are considered as hits and will be further analyzed. Common values for P.nsigma run from $\text{P.nsigma} = 2$ to $\text{P.nsigma} = 5$.

The advantage of this approach is that it is able to provide a reasonable threshold value independently from the total amount of radiation in the illumination area, the gain of the detector, the presence of global offset in the acquired data etc.. This HitOrMiss function is able to provide a fast and reliable estimation of the hit-rate, giving a first important feedback about the progress of the experiment.

After having selected the meaningful diffraction data, the function Analyze is called and represents the core of the analysis software. The aim of this function is to assign to each diffraction pattern a set of values that describes some salient features.

Some of these characteristic values are directly computed on the diffraction data, while some others are computed in the direct space. The ones directly computed on diffraction data are:

- **Amount of diffracted radiation**. It is the sum of the pixel values. The higher the value, the more the brightness of the pattern.
- **Angular variance**. It is the variance of pixel values at a fixed radius. The lower the value, the higher the spherical symmetry of the pattern.
- **Angular correlation**. It is the correlation length of pixel values, computed in the angular direction and averaged over different radii. The higher the value, the more coherent is the variation of the pattern on the angular direction.

- **Radial correlation**. It is the correlation length of pixel values, computed in the radial direction and averaged over different angles. The higher the value, the more coherent is the variation of the pattern on the radial direction.

- **Width**. It is the mean width of Gaussian functions centered at radius 0 obtained by interpolation with the diffraction pattern, computed on different angles. The higher the value, the thicker the diffraction pattern, the lower the dimension of the sample.

- **Width variance**. It is the width variance of Gaussian functions centered at radius 0 obtained by interpolation with the diffraction pattern, computed on different angles. The higher the value, the less spherical is the sample shape.

- **Fringes distance**. The distance in pixels between diffraction fringes is computed on different radial directions. The average on different angles is given. The lower the value, the bigger the sample size.

- **Fringes angular variance**. The distance in pixels between diffraction fringes is computed on different radial directions. The variance on different angles is given. The higher the value, the less spherical the sample shape.

As said before, it is possible to make some analysis also on the Fourier Transform of diffraction data, which corresponds to the auto-correlation function of the sample density distribution. Due to the presence of the beam-stopper, that makes the diffraction data unavailable in the central part of the pattern corresponding to low-frequencies, the FT of the diffraction pattern is actually a sort of high-pass filtered auto-correlation of the density distribution, which is however able to provide some information about the sample shape. Thus, after having performed a FT of the diffraction pattern, some other values can be extracted, which are similar to the previously listed ones but computed in the real space:

- **Angular variance**. It is the variance of pixel values at a fixed radius. The lower the value, the higher the spherical symmetry of the sample.

- **Width**. It is the mean width of Gaussian functions centered at radius 0 obtained by interpolation with the radial profiles of the autocorrelation function, computed for different angles. The higher the value, the thicker the sample width.

- **Width variance**. It is the width variance of Gaussian functions centered at radius 0 obtained by interpolation with the radial profiles of the autocorrelation function, computed for different angles. The lower the value, the more spherical the sample density distribution.

It can be easily understood that some of those values are redundant, i.e. provide similar information computed in different ways.

After the just described analysis step, called at line 7 of Alg.23, the last point consists in writing those values in a text file. Each line of this analysis file corresponds to a single diffraction pattern. The first column is the bunch_id, which identifies the shot, while the following columns report the computed values previously described. Depending on the number of files read by this analysis tool, the analysis file contains information about hundreds (or thousands) of diffraction pattern. Thus, a tool is needed to move inside this database and select the data of interest, which will be described in the next subsection.

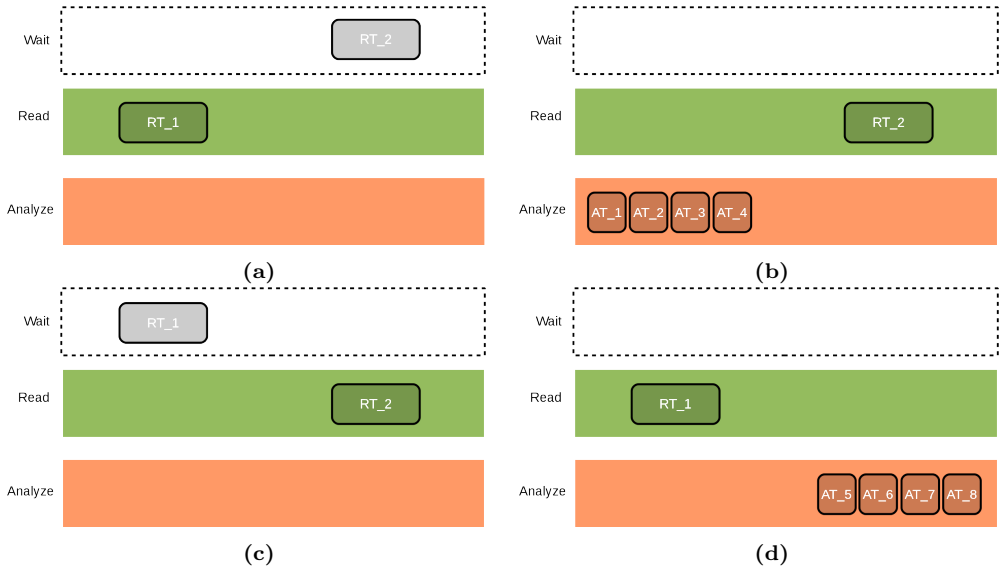


Figure 5.1: The flowchart of the analysis software describing the parallel implementation. RT indicates the *reading threads*, whose role is to read diffraction data from the storage, while AT is the label for the *analysis threads*, whose role is to perform analysis of diffraction data. The organization of computing threads has been designed to optimize the parallel efficiency, such that, at all steps, there is a computing thread that reads data from the storage.

A final note concerns the implementation of this tool. The code has been written in C++, due to the high computational efficiency required by this kind of analysis that must be faster than data acquisition. Thus, the ANALYZE routine has been implemented in parallel, thanks to the `OpenMP` library that allowed a multi-threaded parallelization. In this way, once that the file containing many diffraction pattern has been loaded, many diffraction images can be analyzed at the same time (depending on the computing hardware).

The efficient implementation of the analysis made the I/O operations the real bottleneck of the software. The I/O performances strictly depend on the hardware and cannot be boosted via a different software implementation. The maximum efficiency is, thus, reached when the software is always reading data from the storage. This maximum speedup is again reached thanks to the availability of multi-core CPUs combined with multi-thread programming.

The code has been written such that there are some *Reading Threads* (RT) and some *Analysis Threads* (AT). For example, on a CPU able to manage 8 threads, the flow of data processing is described by Fig.5.1, where the number of RT is 2 and the number of AT is 4.

When new files containing diffraction patterns appear in the working directory, RT_1 (the first Reading Thread) waits for data transfer completion and reads the first file, while RT_2 waits, as depicted in Fig.5.1a.

As soon as RT_1 completes the data loading from the storage, it enters in the analysis step, splitting itself into 4 AT, which means that groups of 4 patterns are analyzed at the same time. In the meanwhile, RT_2 can now start the reading phase on a new file (Fig.5.1b).

As said before, data analysis is faster than data loading, such that the group of ATs ends its work before RT_2 ends data loading. Thus, ATs launched by RT_1 close and RT_1 waits for the completion of the RT2 reading step, as shown in Fig.5.1c.

Fig.5.1d depicts the last step, where RT_2 has ended the reading step and enters in the analysis phase, splitting into 4 threads, while RT_1 is now free to read (or wait for) a new data file. When the second ATs group finishes its work, the configuration restart from Fig.5.1a.

It's worth noting that, for all the described steps, i.e. for all the figures in Fig.5.1, there is always a thread in the green phase, which corresponds to the data loading. In this way, the efficiency of the algorithm is maximum, and a further gain in performance can be achieved only with a hardware update concerning the storage.

5.1.2 Visualization tool

As stated before, as long as the analysis proceeds thanks to the analysis software that runs in background, a lot of information is written in the analysis file. The visualization tool has been developed with the aim to surf the analysis table and to identify the diffraction data of interest. Due to the fact that the visualization software doesn't need high performances and requires a Graphic User Interface (GUI), it has been implemented in Python[4] programming language. The visualization software is based on a GUI composed by two windows, as shown by Fig.5.2.

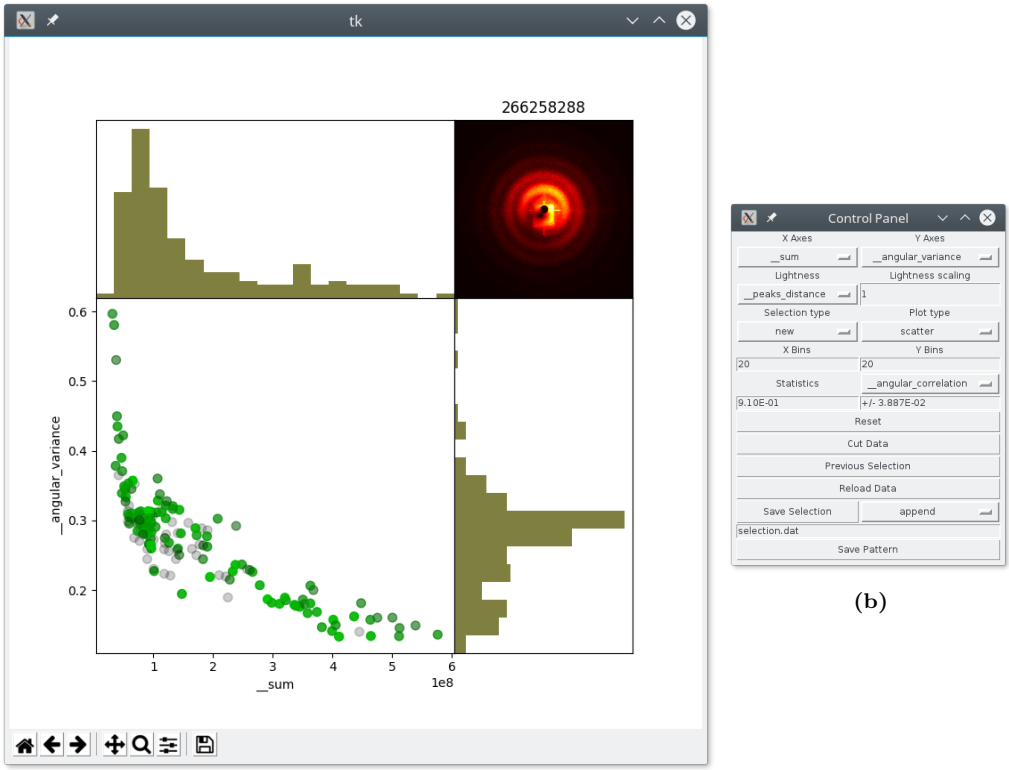
The first window, depicted in Fig.5.2a, is dedicated to the plotting of the values extracted by the analysis tool and written into the analysis file. The main plot is represented by the bi-dimensional scatter plot in the lower left part of Fig.5.2a. Each dot in this plot represents a single diffraction pattern, and its x and y coordinates are two different characteristic values given by the analysis tool (angular variance, angular correlation, fringes distance etc.): also the color of these dots is proportional to one of these values. The rectangle on the upper left is a histogram computed on the x axis, and represents how many diffraction shots are present in the given interval of the x axis. The same is for the lower right rectangle, which is a histogram computed on the y axis. The upper right square area is dedicated to the pattern visualization: when the user clicks on a point of the scatter plot, the pattern corresponding to that point appears.

The features of the plotting window can be controlled and customized thanks to the control panel in Fig.5.2b.

The first controls regard the choice of the parameters on the axes and of the color scale of dots in the plot. When the user clicks on the box entitled X Axes, a drop-down menu opens and enables the user to choose the quantity that will be used as x coordinate in the plot, as shown in Fig.5.3a. Each name corresponds to a column in the analysis file, and column names are declared in its first row. The same can be done for the y axis, in Fig.5.3b and for the color lightness of dots, in Fig.5.3c. The value in the box entitled Lightness scaling changes the relationship between the color lightness of dots and the parameter chosen in the Lightness box on its left. Given λ the lightness, x the chosen parameter and n the value in the Lightness scaling field, the lightness is assigned to dots via the relationship $\lambda = x^n$. If, like in Fig.5.2b, n is set to 1 and the Lightness drop down menu is set to `__peaks_distance`, the dots lightness scales linearly with the detected distance between diffraction fringes. The value of n can be both positive and negative.

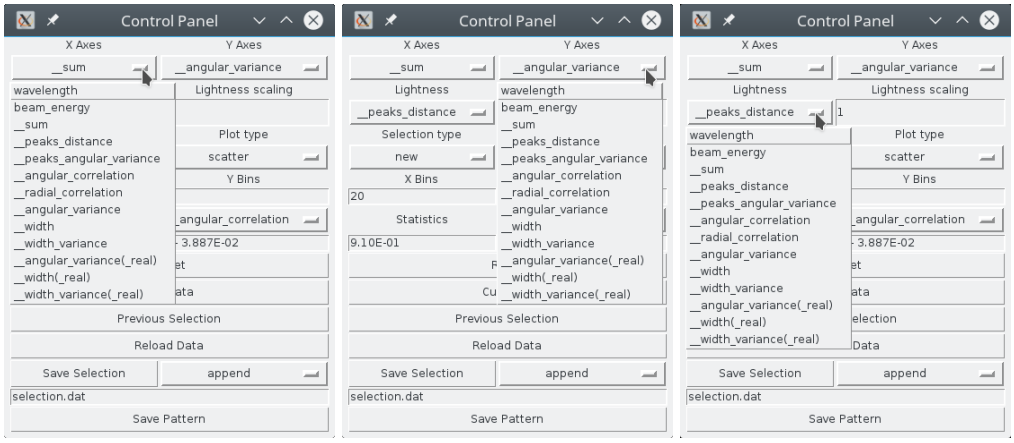
This visualization tool allows the user to visualize data as bi-dimensional histogram, instead of a scatter plot. This utility turns out to be useful when the amount of data is high and it is difficult to get an idea of the statistical distribution of data looking at the

[4] `https://www.python.org/`



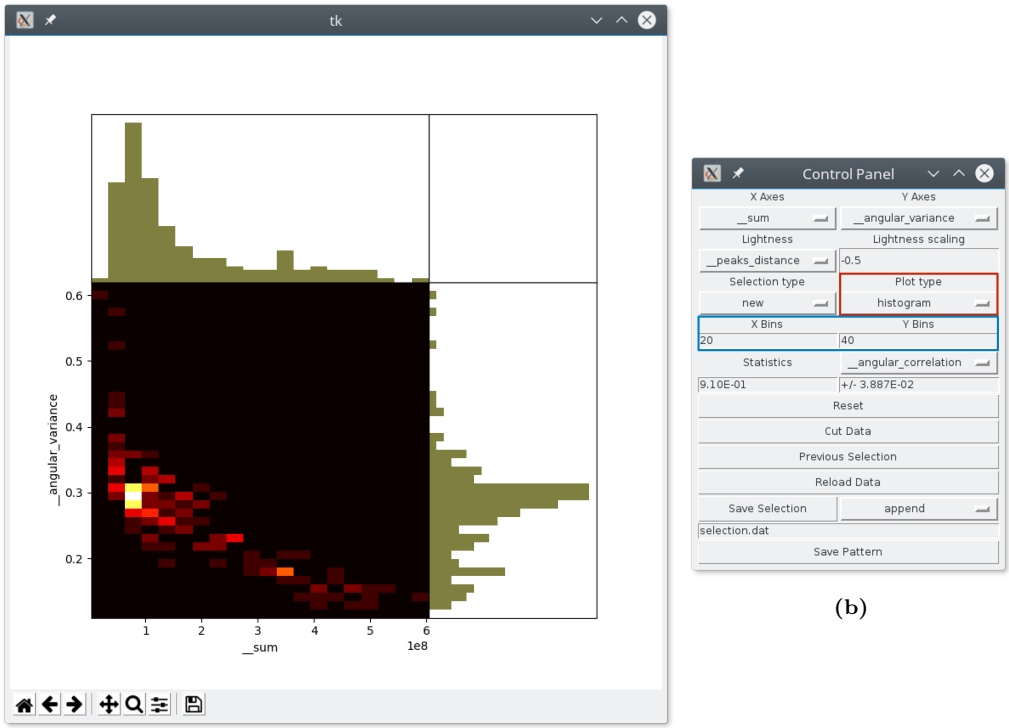
(a) **(b)**

Figure 5.2: A screenshot of the main windows of the visualization software. Panel (a) is dedicated to visualize data as function of the parameters extracted from the analysis, while panel (b) allows the user to control the plot properties and to perform operations on data.



(a) **(b)** **(c)**

Figure 5.3: Screenshots of the control panel that show how it is possible to select different parameters for the plot axes and plot color scale.



(a) (b)

Figure 5.4: Demonstration of the use of the bi-dimensional histogram mode (a) by changing the relative option in the control panel (b), highlighted in red.

scatter plot. The plot type can be changed thanks to the drop-down menu, highlighted in red in Fig.5.4b. The two text boxes highlighted in red change the number of histogram bins on each axis. In the example, the number of bins on the x and y axes have been set to 20 and 40 respectively, and the effect of this choice is clearly visible in Fig.5.4a.

Another feature of the control panel is highlighted in Fig.5.5, which gives the possibility to get the average value of a parameter (selected via the drop down menu on the right) and its standard deviation.

The last worthy of note feature of the visualization tool consists in the possibility to select a subset of the analyzed data. There are many fields in the control panel that allow the user to use the selection tool, as highlighted in Fig.5.6.

First of all, it's worth saying that a selection on data can be performed in a graphical way, by clicking the right button of the mouse and drawing a rectangular region. Selected data remains of green color, while data excluded from the selection turns into red, as depicted by Fig.5.7 on the left. If parameters on the axes are changed, the selection is conserved, as depicted in the right window of Fig.5.7.

Selection can act in different ways, basing on the choice from the drop-down menu inside the red rectangle of Fig.5.6. The meaning of the three options is the following:

new - A new selection is performed, and any previously existing selection is overwritten.

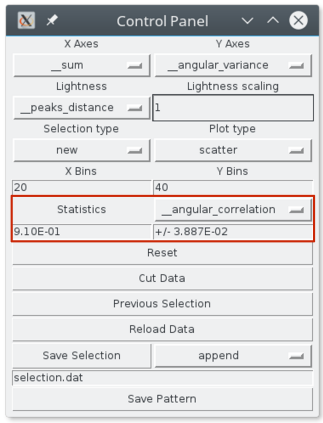


Figure 5.5: The control panel can provide statistics on a given parameter, highlighted in red.

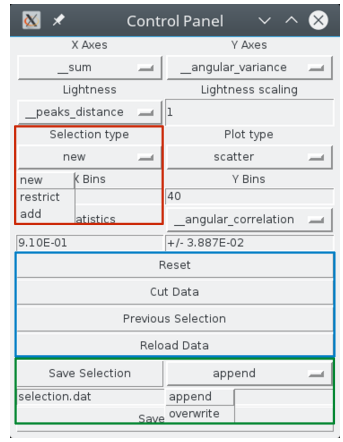


Figure 5.6: The control panel allows to perform some operations on selected diffraction data, by changing the selection type (red), modifying the selection (blue) or saving it in a text file (green).

restrict - The new selection acts only on the previously selected patterns, if any. The resulting selected points are the intersection between the previous selection and the new one.

add - The newly selected points are added to the previous ones, if any.

The blue square in Fig.5.6 contains some buttons that acts over the selection:

Reset - Any present selection is erased and the original data is restored

Cut Data - Data excluded from the current selection is visible no more.

Previous Selection - The selection shape goes back to the previous one, following the selections history, if any.

Reload Data - Data is reloaded from the analysis file. Any current selection will be deleted.

The last operation that the control panel allows to perform on a selection is to save it into a file, which is just a list of bunch ids relative to the selected points. In the green rectangle of Fig.5.6 there is the SAVE SELECTION button that, when clicked, saves the list of selected patterns into the file with a file name written below (in the example, `selection.dat`). The drop-down menu on the right allows to choose between overwriting an existent selection file (if any) or appending the current selection to the one saved in the existing selection file (if any).

Fig.5.8 represents just an example of the potentialities of the analysis tool to inspect data. Fig.5.8a is a screenshot of the visualization tool where the x axis coordinate represents the total scattered intensity, while the y axis refers to the angular variance of the patterns. The pattern in Fig5.8b has been selected from the lower left area, as indicated in Fig5.8a: thus, it provides a low scattered intensity (i.e. the pattern is pretty dark) and a low angular variance (i.e. the patterns has a circular symmetry). Fig.5.8c is, instead,

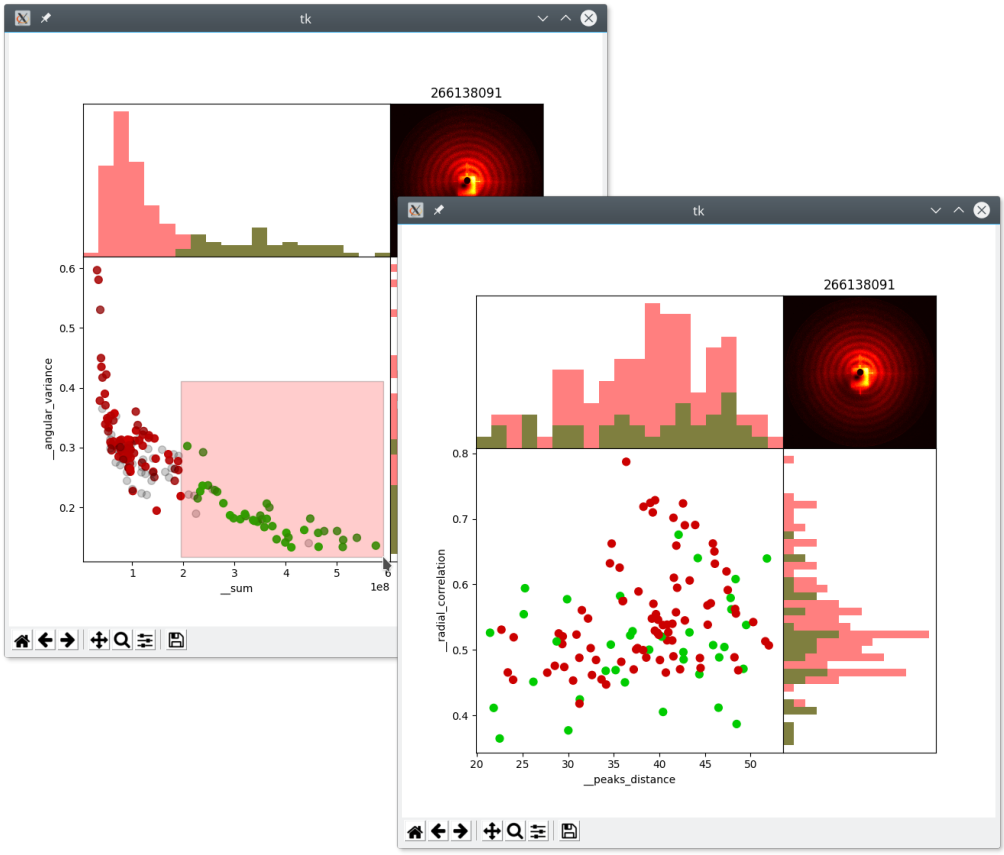


Figure 5.7: Selection can be performed via a right click on the mouse (left). Selection is conserved also when plot properties are changed (right)

extracted from the middle right part of the plot, which means that the scattered intensity is pretty high and it deviates a bit from circular symmetry. The last pattern in Fig.5.8d is, instead, located in the upper part of the plot, which corresponds to high angular variance. In fact, as easily visible in the figure, the pattern has a particularly weird shape, very far from circular symmetry.

As it can be clearly seen in Fig.5.8, diffraction data is affected by some artifacts and defects that directly derive from the features of the acquisition procedure, and a tentative way to mitigate them is undertaken in the next section.

5.2 Microchannel Plate artifacts correction

MicroChannel Plate (MCP) detectors are widely used for detecting X-rays radiation. They are composed, as their name suggests, by many parallel photomultipliers (channels) with the aim to amplify incident X-ray radiation thanks to the photoelectric effect. The use of MCPs in X-ray imaging requires the combination of a phosphor screen, posed after the MCP, to convert MCP output electrons into visible light that can be recorded by a CCD detector. A rough scheme of a MCP structure is depicted by Fig.5.9a, while Fig.5.9b

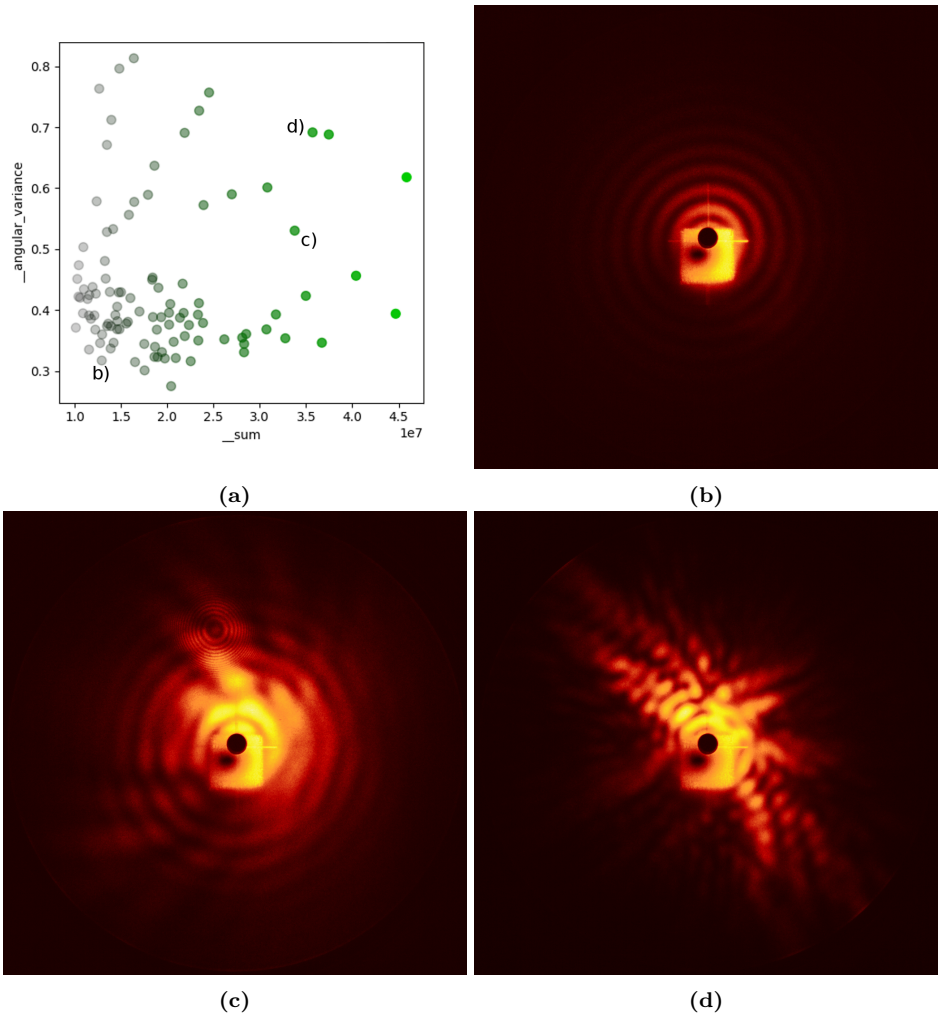


Figure 5.8: Different coordinates in the plot (a) correspond to different properties of diffraction data (b)(c)(d).

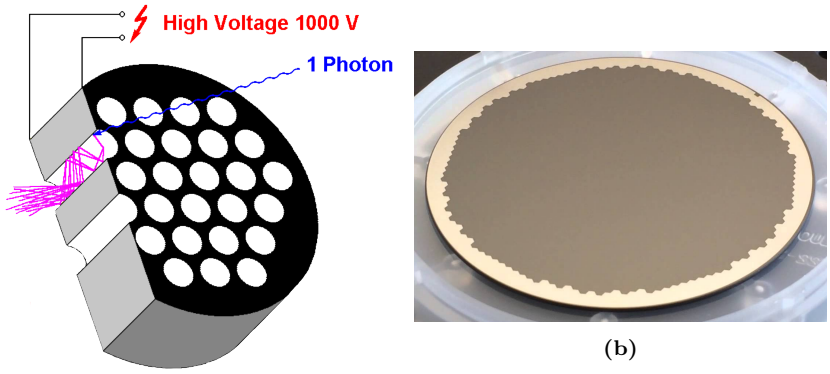
is a picture of a real MCP, which is composed by millions of parallel microchannels.

Different kinds of MCP exist, with different configurations and performances, that won't be discussed here.

There are two big issues concerning MCPs. The first one is that the micro-channel has a saturation value, and when the output signal approaches the saturation value the MCP response is no longer linear with respect to the incoming radiation.

The second problem regards the different response of the single channels depending on their orientation with respect to the incident radiation. In fact, the signal amplification provided by the micro channels depends on how many "rebounds" the radiation has on the channel surface. If the incoming photon perpendicularly enters the channel, there will be few interactions between the photon and the inner surface of the channel, causing a low amplification of the signal.

While the first issue can be, in some sense, bypassed by choosing diffraction patterns



(a) (b)

Figure 5.9: Scheme of a MCP with an exemplification of its operation (a) and picture of a real MCP (b) composed by millions of microchannels.

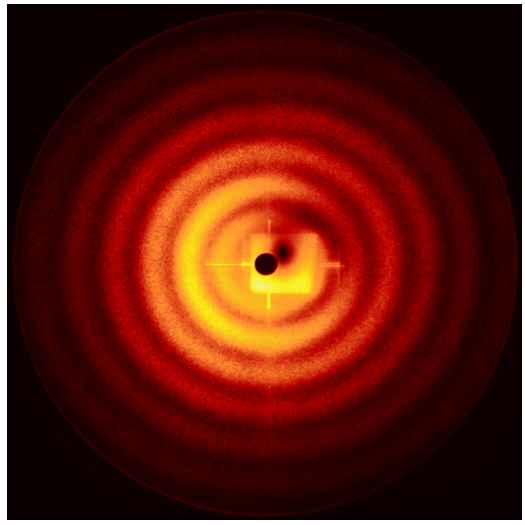


Figure 5.10: Example of typical acquired diffraction pattern. The low response in the upper right part and the experimental noise in the central zone are clearly visible.

whose values are below the saturation region, the latter problem is totally dependent on the experimental setup and cannot be eluded.

A third point regards the noise produced by the experimental tools that affects the acquired data.

These issues are well described by Fig.5.10, which represents a diffraction pattern. In particular, experimental instrumentation causes the rectangular and cross artifact visible in the central part. The different response of the MCP is, instead, well visible in the upper right part of diffraction data, which is less intense with respect to the rest of the pattern.

After these considerations, neglecting the signal saturation issue (i.e. considering only diffraction data below the saturation limit), the relationship between the true diffraction

signal I^{t} and the one measured by the CCD I can be modeled by the following formula:

$$I_{i,j} = B_{i,j} + C_{i,j} \cdot I^{\mathrm{t}}_{i,j} \tag{5.1}$$

where $C_{i,j}$ is the response coefficient of the MCP and $B_{i,j}$ is the noise produced by the experimental instrumentation.

Its worth noting that, at a first approximation, the matrices B and C depends only on the experimental configuration, thus all the N patterns acquired in the same conditions will have the same B and C matrices. This consideration opens the possibility to use the information provided by the N patterns I^n with $n = 1, ..., N$, to retrieve B and C.

A first estimation of the background matrix B is computed by averaging all the diffraction data corresponding to a target *miss*. In fact, if nothing is present in the interaction zone, each *miss* diffraction pattern corresponds to an estimation of the background signal. This operation requires to discriminate between *hits* and *misses*, and the operation is accomplished in the same way described in section 5.1.1. In this case, blank patterns are averaged to get an estimation of the background matrix B. Thanks to this operation, a new matrix I^{sub} is obtained for each diffraction data, such that (5.1) turns into:

$$I^{\mathrm{sub}}_{i,j} = C_{i,j} \cdot I^{\mathrm{t}}_{i,j} \tag{5.2}$$

At this point, the mission to accomplish is to retrieve the coefficient map C without the knowledge of the true diffracted intensity I^{t}. This can be done by exploiting two properties of diffraction data:

1. There is a region in the pattern where the response of the MCP is optimum and independent on the coordinates, i.e. $C_{i,j} = C_{\mathrm{opt}}$ for all i, j in this region.

2. Patterns that have circular symmetry must have the same intensity at a given radius, independently on the angle.

The first consideration introduces the need to identify a *trusted region*, depicted in green in Fig.5.11, where

$$\frac{I^{\mathrm{sub}}_{i,j}}{C_{\mathrm{opt}}} = I^{\mathrm{t}}_{i,j} \tag{5.3}$$

If only circular patterns are selected, then a *trusted* radial profile $I^{\mathrm{trust}}(r)$ can be computed in this region, by averaging pixels at the same distance from the center. This radial profile can be assumed to be proportional to the *true* radial profile, that is $I^{\mathrm{trust}}(r) = C_{\mathrm{opt}} \cdot I^{\mathrm{t}}(r)$. In this way, for any given pixel i, j at radius r outside the *trusted* region, it holds that:

$$I_{i,j} = \frac{B_{i,j}}{C_{\mathrm{opt}}} + \frac{C_{i,j}}{C_{\mathrm{opt}}} \cdot I^{\mathrm{trust}}(r_{i,j}) = \tilde{B}_{i,j} + \tilde{C}_{i,j} \cdot I^{\mathrm{trust}}(r_{i,j}) \tag{5.4}$$

where $r_{i,j}$ is the radius related the coordinates i, j, $\tilde{B}_{i,j} = \frac{B_{i,j}}{C_{\mathrm{opt}}}$ and $\tilde{C}_{i,j} = \frac{C_{i,j}}{C_{\mathrm{opt}}}$.

At this point, it is clear that retrieving the matrices $\tilde{B}_{i,j}$ and $\tilde{C}_{i,j}$ means retrieving the correct MCP response up to a constant value. Given N^{circ} diffraction patterns that corresponds to a *hit* signal and have circular symmetry, for each coordinate i, j there are N^{circ} couples of values: the measured diffraction data $I_{i,j}$ and the expected value at that coordinate $I^{\mathrm{trust}}_{i,j}$. Thus, once that the N^{circ} couples $\{I_{i,j}, I^{\mathrm{trust}}_{i,j}\}$ are computed for each coordinate i, j, a linear interpolation is performed for each pixel following Eq.(5.4) where $\tilde{B}_{i,j}$ and $\tilde{C}_{i,j}$ are set as free parameters.

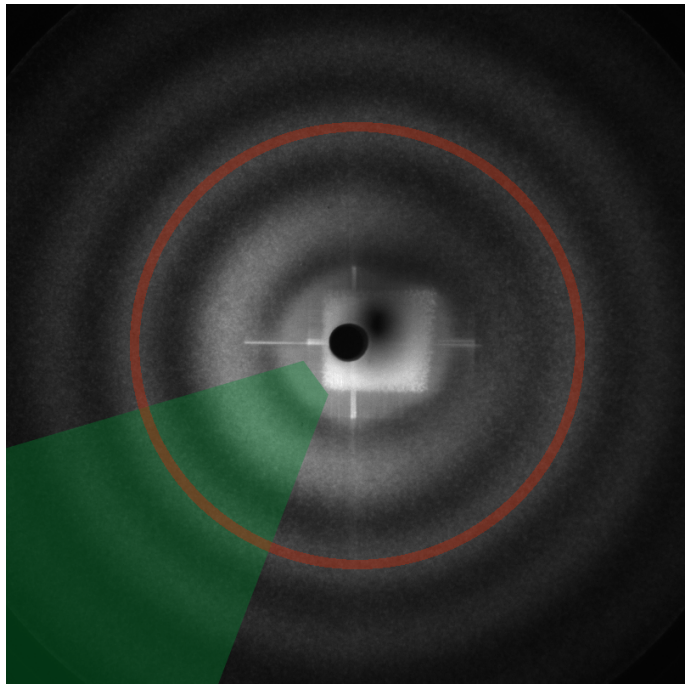
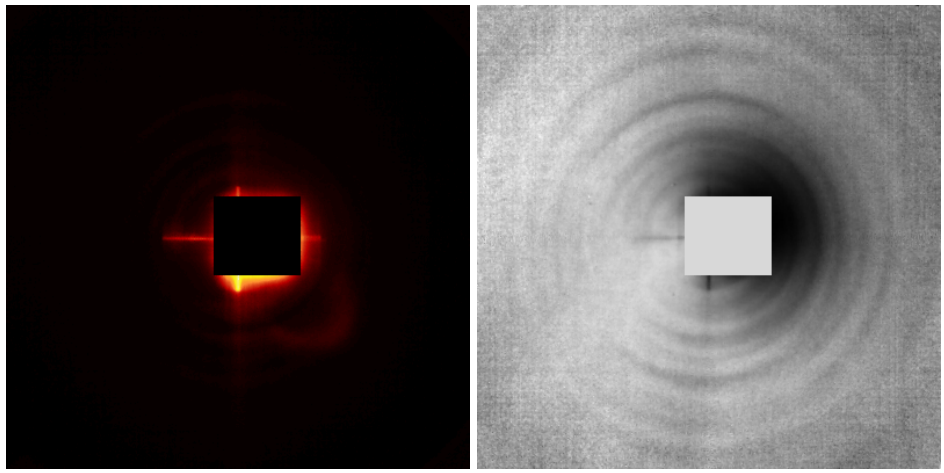


Figure 5.11: A circular pattern should provide the same diffracted intensities at a fixed distance from the origin (red circle). In green, the *trusted* area, where the scaling factor between the incoming radiation on the MCP and the output one is expected to be constant, i.e. independent on the pixel coordinates.



(a) **(b)**

Figure 5.12: The background map B (a) and the linear coefficients map C (b) extracted from diffraction data characterized by a circular shape

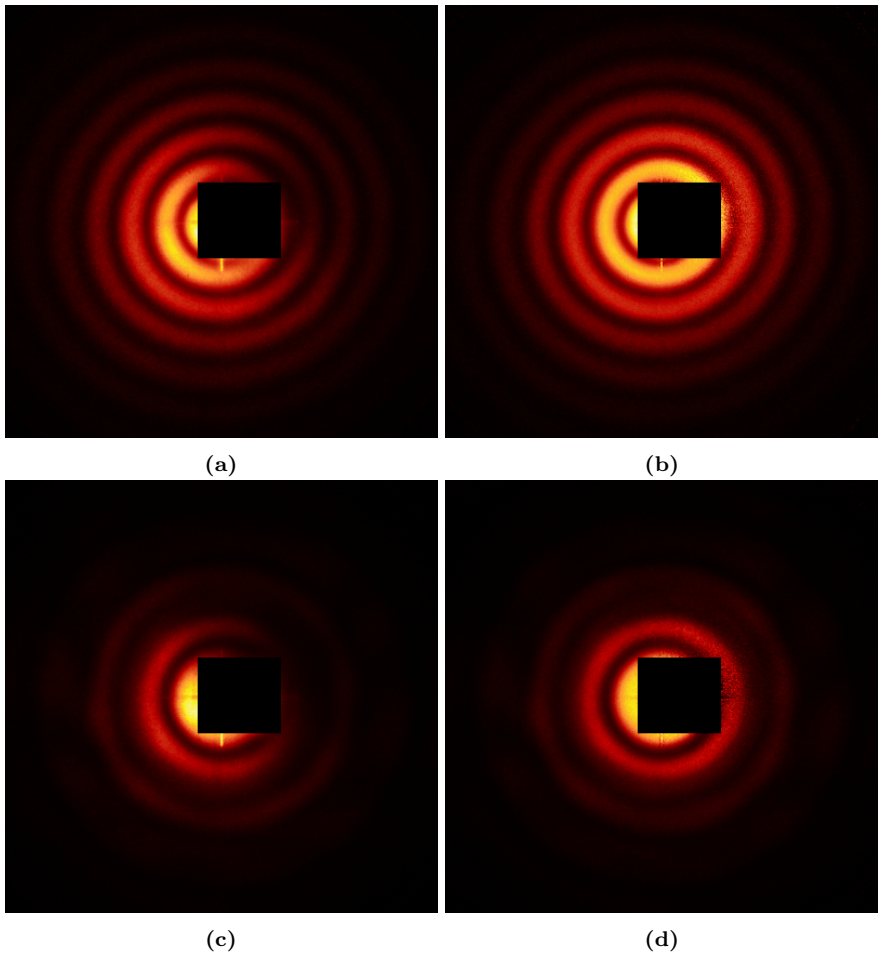


Figure 5.13: Original diffraction data with circular shape (a)(c) and their version corrected by the use of maps in Fig.5.12 (b)(d). The circular symmetry of the retrieved patterns (b) and (d) suggests that diffraction data is well retrieved by the algorithm.

Fig.5.12 is an example of the approach. In particular, Fig.5.12a is a representation of the matrix $\tilde{B}_{i,j}$ while Fig.5.12b is the matrix $\tilde{C}_{i,j}$. These maps have been extracted by performing a linear interpolation for each pixel, as previously described. Diffraction data used to generate these maps has been selected among the ones inside the dataset, by choosing, for the interpolation step, only patterns that provides circular symmetry, as the one depicted in Fig.5.13a. In all these images it can be noted a masked area near the center of the diffraction pattern. This data has been excluded from the analysis because, in those pixels, experimental noise dominates, totally hiding any meaningful signal.

Fig.5.13 shows the application of Eq.(5.4), using the maps in Fig.5.12a and Fig.5.12b, on symmetrical diffraction data. On the left (Fig.5.13a and Fig.5.13c) two diffraction patterns are shown, as they were acquired by the detector. On the right, instead, there are the *restored* versions of them. The approach demonstrates to work well both on very symmetric data, in Fig.5.13b, and on nearly circular patterns, in Fig.5.13d.

The software developed and described in this section turns out to be particularly

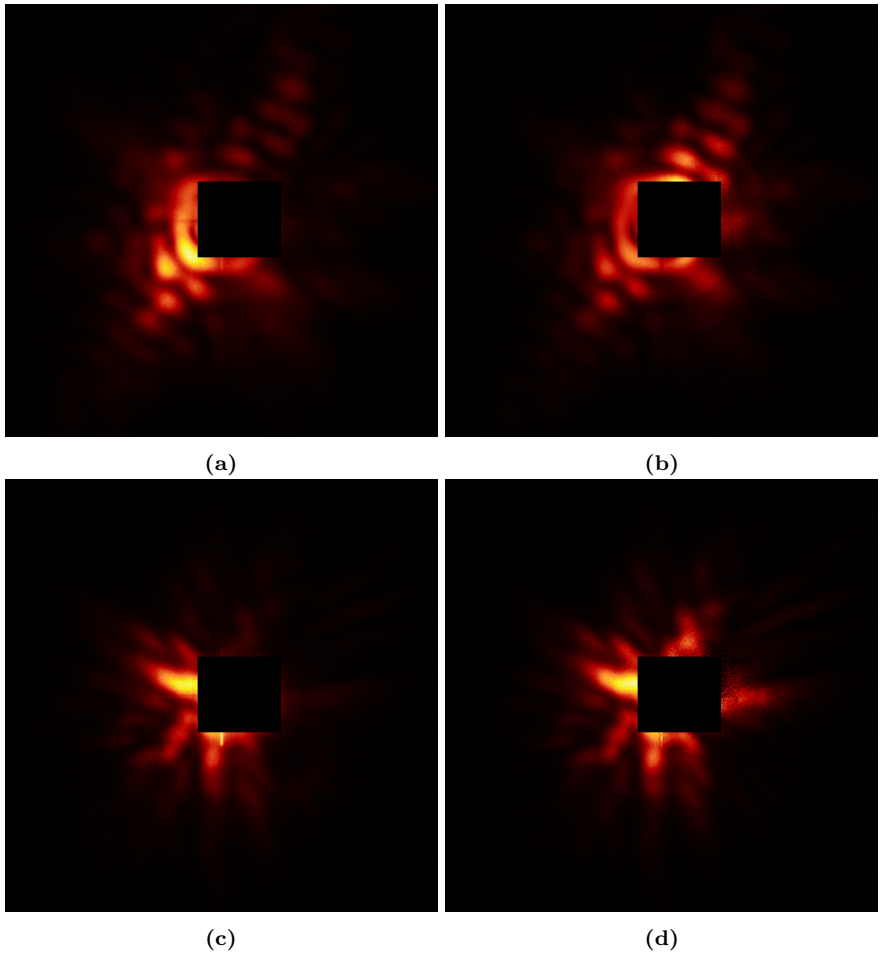


Figure 5.14: Original diffraction data with an asymmetric shape (a)(c) and their version corrected by the use of maps in Fig.5.12 (b)(d). Low intensities and unpredictable features in the upper right zone, hidden by the distorted response of the MCP, are now well visible, with an intensity value coherent with the rest of the diffraction pattern.

useful when diffraction data has no symmetries. In fact, when a diffraction pattern is, for example, centro-symmetric, half of the diffraction data is redundant, such that it is not necessary to retrieve it if damaged by experimental issues. Instead, in the case of non symmetric patterns, each pixel acquired by the CCD provides useful and unique information, such that the correction of acquisition defects turns out to be fundamental to maximize information on the sample of interest. For this reason, the correction software has been applied also on diffraction data, belonging to the same dataset, which is not symmetrical (or, at least, doesn't have a circular shape), exploiting the correction maps extracted from circular diffraction patterns in Fig.5.12a and Fig.5.12b. Fig.5.14a and 5.14c represent the acquired diffraction data, while 5.14b and 5.14d are the corrected counterparts. For all of these examples, the software is able to retrieve some unpredictable features of diffraction data that were hidden by the low response of the MCP. Fig.5.14 clearly shows that this approach is able to (at least partially) correct some artifacts

introduced by the acquisition procedure, and in particular the different response of the MCP as function of the coordinates i, j.

The main limitation of this approach concerns the starting hypothesis in Eq.(5.1): this equation assumes that the response of the MCP is linear. As said before, it can be assumed to be true if the output signal is far from the saturation value of the single micro-channels, but when the signal approaches the saturation value the relationship between the measured data I and the input signal I^{t} has a sub-linear behavior. Thus, a more reliable approach should include this feature. Work is still in progress, and some preliminary tests have been performed, but the results turn out to be very sensitive to the behavior of the response function near to the saturation value.

Conclusions

The aim of this work was to investigate new ways to face the *phase retrieval problem* in the field of Coherent Diffraction Imaging (CDI), and in particular its "standard" form, which requires the acquisition of single scattering patterns of isolated samples. The great interest in the field derives from the fact that, at the moment, CDI is, for example, one of the few imaging techniques well suited for time resolved imaging at Free Electron Lasers facilities. This CDI technique presents many challenges, that concerns experimental setups, data acquisition and data analysis. If the first and the second are strictly bounded to the performances of the experimental hardware, the latter is mainly a mathematical and computational challenge. In fact, in order to retrieve the specimen density distribution, it is necessary to solve the so–called *phase retrieval problem*, which represents a hard optimization task.

The last twenty years saw an improvement in the performances of computing hardware that was rarely (or maybe never) seen in other fields. As well as these improvements changed our lifestyle, science underwent a true revolution due to the opportunities opened by new technologies. When the capabilities of computational hardware increase, new methods of simulation and analysis become possible. The growth of computational resources in the last years was so fast that, often, science was not able to keep up with new methods. This is, in some sense, the case of phase retrieval algorithms for CDI.

In the 70's the first reliable phase retrieval algorithm was proposed, the Hybrid Input-Output (HIO). From that moment on, and in particular in the last ten years, many other algorithms were proposed. These new approaches, that in some cases outperform the HIO algorithm, didn't represent a true breakthrough with respect to performances and reliability, such that the original HIO algorithm is still widely used. The most part of the development of phase retrieval algorithms was focused on their mathematical aspect, reaching, in some cases, very high levels of sophistication. While the research was totally dedicated to answer the question "How do these algorithms work?", nearly no one cared about how to use them better.

In the introduction of this work, the search for a solution to the *phase retrieval problem* was likened to the search of the hole in a golf course. The search for the hole is usually performed by a team of blind golfers, which are only aware of their position and their altitude, without any knowledge about the hole direction. The only possible way to find the hole is to randomly place themselves in the golf course, throw their own ball and hope that it reaches the hole, which is placed in the lowest point of the field.

The aim of this work was to give them a voice, that is to make them able to communicate and share the information about their position and their performance.

The mean chosen for sharing information is a Genetic Algorithm, which is a well known stochastic optimization method able to perform a more global exploration of the configuration space. This stochastic approach treats a *population* of individuals, i.e. candidate solutions to the problem, which *evolves* by means of the *Selection*, *Mutation* and *Crossover* operators, imitating the survival-to-fitness of natural evolution. The existent iterative projection algorithms have been inserted in this stochastic framework, giving rise to the Memetic Phase Retrieval (MPR) approach.

MPR represents a novelty in this field under, at least, three aspects:

- It is the first approach to the *phase retrieval problem* that makes an extensive use of Computational Intelligence
- It is able to fully exploit nowadays High Performance Computing hardware
- It systematically outperforms the standard approach to phase retrieval.

This result was possible only thanks to a well suited implementation for the most recent High Performance Computing (HPC) facilities. In fact, the high amount of computational resources needed by MPR, especially when facing three dimensional CDI, is available only nowadays, while just few years ago this approach wouldn't have been possible (nor easily conceivable).

Tests on both synthetic and experimental data show that MPR is a convenient way when the standard approach fails, or when its results are not satisfactory. Moreover, as said before, MPR isn't actually a new phase retrieval algorithm (in the strict sense), but it is a new way to exploit the existent ones: this means that MPR will benefit from any further improvement concerning iterative algorithms. A clear example of this possibility is the integration of the *Shrinkwrap* algorithm inside MPR, that turns out to be highly reliable in retrieving the correct support function and very flexible and less dependent on a fine tuning of its parameters.

As showed in the last section on experimental 3D data, the statistical nature of the approach, which treats many candidate solutions to the phase problem at the same time, provides useful information about the space of configurations, and it makes the scientist able to extract, from this statistic, useful parameters to face issues in the experimental data, like noise and lacks in the diffraction patterns.

Furthermore, it has been shown that MPR performance is proportional to its computational weight, which can be tuned by setting different sizes of the *population* of candidate solutions. This means that it is somehow flexible, i.e. it can be adapted to the available computational resources, and an improvement in computing hardware will directly reflects in an improvement of MPR results.

Thanks to its unique features, MPR has recently been recognized as a novelty in the field, representing a solid basis for futher developements of Computational Intelligence applied to Coherent Diffraction Imaging (Truong et al. 2017).

Perspectives

The use of a new methodology on an existent problem in Science, like the *phase retrieval problem*, always opens many perspectives, both under the methodology point of view and the use of this new methodology in similar problems. Concerning the latter, it can be stated that the use of MPR is not limited to "classical" Coherent Diffraction Imaging (CDI), but it can be applied also to other imaging techniques where phase retrieval is needed and standard approaches present performance issues.

For what concerns the perspectives about the development of MPR, a further distinction should be done between implementation-related improvements and the method-related ones. Concerning the implementation, this work described the realization of MPR code, which is able to exploit the massively parallel hardware installed on nowadays High Performance Computing facilities. A careful reader may have noted that the so-called GPU computing technique wasn't mentioned. GPUs (Graphics Processing Units) are coprocessors, specialized processing unit that were born in the late 90's with the aim of accelerating image processing and rendering. In the last year, GPUs became one of the most used hardware for scientific computing thanks to their ability to provide amazing speedups for certain kinds of tasks, such as the *phase retrieval problem*. The main issue concerning a MPR implementation able to exploit GPU coprocessors is the huge amount of required memory. In fact, many data transfers between the host memory and the (limited) GPU memory are needed and would represent an insurmountable bottleneck in reaching the best performance. However, the most recent developments in GPU hardware now provide large memories installed on-chip (at the moment, up to 32GB) and the possibility to directly access the host memory space. Thus, MPR approach on GPUs now starts to be possible, and an implementation capable of GPU-computing is, among the implementation-related perspectives, the most interesting and promising one.

On the other side, many further developments concerning the methodology are possible, and the most relevant are listed below.

- At the current stage, MPR treats as *gene* each single value of the matrix, i.e. the single pixel of the image. This is the most logical and most trivial choice. However, other choices could be done, for example by treating all the phases relative to a spatial resolution in the direct space as a single gene. A different definition of *genes* may result in more efficient **Crossover** and **Mutation** steps.

- The **Mutation** step is performed randomly. This means that each phase may undergo a random shift. This way of mutating the genetic pool is particularly trivial, but it is often very destructive. A more global **Mutation** operator, which acts on groups of phases, may reduce the destructiveness of this step, improving its efficiency. A first step in this way has been done by developing a *smart mutation* (Mauri et al. 2018) which exploits the existing correlations between the coordinates where the discrepancy with the experimental data is high and where the phases are badly retrieved.

- MPR implementation of the **Self-improvement** step consists in a given number of iterations of a given iterative algorithm (such as Hybrid Input-Output, Error Reduction, Difference Map etc.). All the individuals inside the population undergo the same algorithms with the same amount of iterations. A development may consist in randomizing the **Self-improvement**, such that different groups of individuals undergo different algorithms. This approach could reduce stagnation issues, due to the fact that different algorithms follow different trajectories in the space of configurations. A further step could be to insert the algorithm sequence inside the genetic pool, such that the choice of the best algorithm is itself subjected to an optimization process by the evolutionary dynamic.

- MPR optimization target, i.e. the functional that the genetic procedure has to optimize, is the same of the iterative projection algorithms. Actually, one of the strengths of GAs resides in their ductility, i.e. the possibility to make them optimize any given *cost function*, without restrictions. Setting a different optimization target

for the *fitness* evaluation may result in a less stagnant optimization. On the other side, the **Self-improvement** will always try to optimize the optimization target of iterative projection algorithms: this fact may result in instabilities of the approach and a careful and deep analysis should be done on this point.

This PhD work didn't pretend to be exhaustive, nor definitive. It just represents a first step of phase retrieval inside the world of Computational Intelligence. Different stochastic approaches may reveal to be much more efficient and reliable than Memetic Phase Retrieval in the future, maybe thanks to the improvements in theory and computing hardware. I would say that MPR, in a sense, is like the magnifying glass with respect to the following invention of the microscope, or the kite compared to the airplane, or the telegraph with the smartphone. But even a telegram is a breakthrough for a blind golfer looking for the hole.

Scientific acknowledgments

The first scientific acknowledgment is for the MIUR PRIN 2012 project NOXSS, which gave me the opportunity to work on this topic since my bachelor thesis in 2013. Inside NOXSS project, I am particularly grateful to Elvio Carlino[1], Liberato De Caro[2] and Francesco Scattarella[2]: thanks to their collaboration the first published application of Memetic Phase Retrieval was performed on Electron Diffraction data. Moreover, they provided a valuable contribution in the development of the algorithm and in my training as a scientist.

A particular thank to Yuriy Chushkin[3] and his collaborators, for having provided the three dimensional diffraction datasets. The development and testing of the 3D version of Memetic Phase Retrieval was possible also thanks to him and his trust in me.

I am grateful to prof. Paolo Piseri[4] for giving me the chance to participate in two XFEL beamtimes, at SACLA[5] and FERMI[6]. The part of the thesis concerning the online data analysis tool and pattern restoration was only possible thanks to his help and suggestions.

I acknowledge CINECA SCAI department for the huge amount of computational resources provided in these years, in particular via the LISA award PUMAS, the IscraC award IMAGES and the IscraB award MEMETICO. In particular, 3D imaging with MPR was only possible thanks to the huge amount of resources provided by the latter.

I acknowledge the organizers of ParCo2017 conference for the opportunity to present the Memetic Phase Retrieval implementation, coming into contact with the international High Performance Computing community. Moreover, I am particularly grateful to Cinzia Giannini and the other organizers of the XTOP 2018 conference for the possibility to tell about my PhD work to an audience composed by the top scientists in the field of X-ray Imaging.

[1]IMM-CNR, Lecce, Italy
[2]IC-CNR, Bari, Italy
[3]ESRF, Grenoble, France
[4]Università degli Studi, Milan, Italy
[5]RIKEN Harima Institute, Japan
[6]Trieste, Italy

Ringraziamenti

Nonostante fosse (forse) piú consono farlo negli *scientific acknowledgments*[1], dopo oltre cinque anni di lavoro insieme ti sei certamente guadagnato la *pole position* di questa sezione, decisamente piú personale e informale. Quindi, grazie Davide. Ci sono mille[2] cose per cui ringraziarti, ma é inutile elencarle tutte, perché tanto qualcuna la dimenticherei comunque, e il metro di misura della gratitudine non e' certo il tempo speso a ringraziare. Ma a due di queste tengo particolarmente. Grazie, per aver avuto fiducia in me sin dal principio, quando nel 2013 mi hai proposto una tesi[3] su un problema che avevamo ben poco chiaro[4]. E grazie per il fastidioso ottimismo che mi hai infuso in questi anni, non perché abbia reso possibile l'impossibile, ma perché mi ha insegnato che é meglio rendere i successi piú dolci che le sconfitte meno amare.

Ringraziare i genitori é spesso vissuto come un obbligo. Dopotutto, per un verso o per l'altro ci hanno messo al mondo e amato piú di ogni altra cosa, e solo per questo si é loro debitori. Ma forse pochi hanno avuto la fortuna di una famiglia come la mia, che raramente ha preteso e sempre ha incoraggiato, e ha sempre riposto fiducia in me forse anche quando non me la meritavo[5]. Grazie mamma e papá[6]. E grazie a Betty e Andrea[7], che si ricordano di avere un fratello, anche se é dal 2010 che quasi non mi faccio piú vedere a casa.

Grazie ai nonni. A nonna Nori per gli incoraggiamenti ad alto contenuto di zuccheri, a nonna Linda per quelli ad alto contenuto di preghiere[8].

Grazie ai suoceri[9] Patrizia e Roberto, per l'enorme affetto e il sostegno in questi anni.

Grazie agli amici, tutti. Non perdo tempo a ringraziarli uno ad uno[10].

Grazie alla donna[11] della mia vita. Questi ultimi tre anni sono stati una vera rivoluzione. Ho cominciato il dottorato da studente, l'ho concluso da marito. E queste cose non si fanno certo da soli. Quindi grazie Fede perché mi hai fatto sentire amato e mi hai dato la serenitá e il supporto che mi hanno permesso di arrivare fin qui. Le incertezze e le difficoltá sono e saranno tante, ma con te a fianco fanno meno paura. Grazie Fede, perché ho cominciato il dottorato da studente e l'ho concluso da papá. Grazie, perché quando un giorno Cecilia[12] leggerá queste parole sarà difficile inventarmi qualche scusa per nascondere la commozione.

[1] dopotutto, sei un professore!
[2] forse un pó meno, ma almeno una dozzina sí.
[3] destinata a fallire
[4] dicendola con un eufemismo
[5] peró l'ho meritata quasi sempre
[6] Eliana e Roberto li ho usati troppo spesso!
[7] non ho scritto "Taccio e Tia" per evitare di mettervi in imbarazzo
[8] Athos e Giannino non rimaneteci male, tanto voi due, da lì, mi leggete direttamente nel cuore.
[9] mi é sfuggito in un momento di debolezza
[10] perché tanto non avranno mai né il tempo né la voglia di leggere questa tesi
[11] seconda donna (N.d.A.)
[12] la mia prima donna (N.d.A)

Bibliography

Abbey, B., Nugent, K. A., Williams, G. J., et al. 2008, Nature Physics, 4, 394

Abrahams, J. & Leslie, A. 1996, Acta Crystallographica Section D: Biological Crystallography, 52, 30

Ashlock, D. 2006, Evolutionary computation for modeling and optimization (Springer Science & Business Media)

Bäck, T., Fogel, D. B., & Michalewicz, Z. 1997, Handbook of evolutionary computation (CRC Press)

Barty, A., Boutet, S., Bogan, M. J., et al. 2008, Nature photonics, 2, 415

Bates, R. 1982, Optik (Stuttgart), 61, 247

Batson, P., Dellby, N., & Krivanek, O. 2002, Nature, 418, 617

Bauschke, H. H., Combettes, P. L., & Luke, D. R. 2002, JOSA A, 19, 1334

Bauschke, H. H., Combettes, P. L., & Luke, D. R. 2003, JOSA A, 20, 1025

Bogan, M. J., Boutet, S., Barty, A., et al. 2010, Physical Review Special Topics-Accelerators and Beams, 13, 094701

Bonifacio, R., Pellegrini, C., & Narducci, L. 1984, 118, 236

Born, M. & Wolf, E. 1999, Cambridge U. Press, Cambridge, UK, 890

Chapman, H. N., Barty, A., Bogan, M. J., et al. 2006, Nature Physics, 2, 839

Chapman, H. N. & Nugent, K. A. 2010, Nature Photonics, 4, 833

Chen, C.-C., Miao, J., Wang, C., & Lee, T. 2007, Physical Review B, 76, 064113

Cherkas, O., Beuvier, T., Breiby, D. W., et al. 2017, Crystal Growth & Design, 17, 4183

Colombo, A., De Caro, L., & Galli, D. E. 2018, Parallel Computing is Everywhere, 32, 67

Colombo, A., Galli, D. E., De Caro, L., Scattarella, F., & Carlino, E. 2017, Scientific Reports, 7, 42236

Cooley, J. W. & Tukey, J. W. 1965, Mathematics of Computation, 19, pp. 297

Dagum, L. & Menon, R. 1998, IEEE computational science and engineering, 5, 46

De Caro, L., Carlino, E., Caputo, G., Cozzoli, P. D., & Giannini, C. 2010, Nat Nano, 5, 360

De Caro, L., Carlino, E., Siliqi, D., & Giannini, C. 2013, in Handbook of Coherent-Domain Optical Methods (Springer), 291–314

De Caro, L., Carlino, E., Vittoria, F. A., Siliqi, D., & Giannini, C. 2012, Acta Crystallographica Section A, 68, 687

Eiben, A. E., Smith, J. E., et al. 2003, Introduction to evolutionary computing, Vol. 53 (Springer)

Ekeberg, T., Svenda, M., Abergel, C., et al. 2015, Physical review letters, 114, 098102

Elser, V. 2003, JOSA A, 20, 40
Fienup, J. R. 1975, Stanford University
Fienup, J. R. 1978, Opt. Lett., 3, 27
Fienup, J. R. 1982, Appl. Opt., 21, 2758
Fienup, J. R. 1987, JOSA A, 4, 118
Fienup, J. R. & Wackerman, C. C. 1986, J. Opt. Soc. Am. A, 3, 1897
Gaffney, K. & Chapman, H. 2007, Science, 316, 1444
Gerchberg, R. W. & Saxton, W. O. 1972, Optik, 35, 237
Goldberg, D. E. 1989, Genetic Algorithms in Search, Optimization and Machine Learning, 1st edn. (Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.)
Gorkhover, T., Ulmer, A., Ferguson, K., et al. 2018, Nature Photonics, 12, 150
Gropp, W. D., Gropp, W., Lusk, E., & Skjellum, A. 1999, Using MPI: portable parallel programming with the message-passing interface, Vol. 1 (MIT press)
Hagena, O. & Obert, W. 1972, The Journal of Chemical Physics, 56, 1793
Hennessy, J. L. & Patterson, D. A. 2011, Computer architecture: a quantitative approach (Elsevier)
Hoefler, T., Dinan, J., Thakur, R., et al. 2015, ACM Transactions on Parallel Computing, 2, 9
Huang, W., Zuo, J., Jiang, B., Kwon, K., & Shim, M. 2009, Nature Physics, 5, 129
Koza, J. R. 1994, Statistics and computing, 4, 87
Langbehn, B., Sander, K., Ovcharenko, Y., et al. 2018, arXiv preprint arXiv:1802.10584
Lengeler, B., Schroer, C., Richwin, M., et al. 1999, Applied physics letters, 74, 3924
Lentzen, M., Jahnen, B., Jia, C., et al. 2002, Ultramicroscopy, 92, 233
Loh, N., Bogan, M. J., Elser, V., et al. 2010, Physical review letters, 104, 225501
Luke, D. R. 2004, Inverse problems, 21, 37
Marchesini, S. 2007, Review of Scientific Instruments, 78,
Marchesini, S., Chapman, H., Barty, A., et al. 2005, IPAP Conf. Series, 7, 380
Marchesini, S., He, H., Chapman, H. N., et al. 2003, Phys. Rev. B, 68, 140101
Matyas, J. 1965, Automation and Remote control, 26, 246
Mauri, M., Galli, D. E., & Colombo, A. 2018, Toward a Science Campus in Milan
Miao, J., Charalambous, P., Kirz, J., & Sayre, D. 1999, Nature, 400, 342
Miao, J., Ishikawa, T., Johnson, B., et al. 2002, Physical review letters, 89, 088303
Miao, J., Sayre, D., & Chapman, H. 1998, JOSA A, 15, 1662
Moscato, P. & Cotta, C. 2003, in Handbook of metaheuristics (Springer), 105–144
Moscato, P., Cotta, C., & Mendes, A. 2004, in New optimization techniques in engineering (Springer), 53–85
Moscato, P. et al. 1989, Caltech concurrent computation program, C3P Report, 826, 1989
Ong, Y.-S., Lim, M. H., & Chen, X. 2010, IEEE Computational Intelligence Magazine, 5, 24
Rastrigin, L. 1963, Automaton & Remote Control, 24, 1337
Rupp, D., Adolph, M., Flückiger, L., et al. 2014, The Journal of chemical physics, 141, 044306
Sarma, M. 1990, Journal of Optimization Theory and Applications, 66, 337
Sayre, D. 1952, Acta Crystallographica, 5, 843
Schmitt, L. M. 2001, Theoretical Computer Science, 259, 1
Sodani, A. 2015, in Hot Chips 27 Symposium (HCS), 2015 IEEE, IEEE, 1–24

Spence, J. 2008, Nature Photonics, 2, 390

Spiller, E. et al. 1994, Soft X-ray optics (SPIE Optical Engineering Press Bellingham, WA)

Storn, R. & Price, K. 1997, Journal of global optimization, 11, 341

Taylor, L. 1981, IEEE Transactions on Antennas and Propagation, 29, 386

Truong, N. X., Whittaker, E., & Denecke, M. A. 2017, Journal of Applied Crystallography, 50, 1637

Williams, D. B. & Carter, C. B. 1996, in Transmission electron microscopy (Springer), 3–17

Yang, C., Qian, J., Schirotzek, A., Maia, F., & Marchesini, S. 2011, arXiv preprint arXiv:1105.5628

Zheng, G., Horstmeyer, R., & Yang, C. 2013, Nature photonics, 7, 739

Zuo, J., Vartanyants, I., Gao, M., Zhang, R., & Nagahara, L. 2003, Science, 300, 1419