Technische Universität München
Physik Department
Lehrstuhl für Biomedizinische Physik

Development and Application of Decoherence Models in Ptychographic Diffraction Imaging

Björn Enders

Vollständiger Abdruck der von der Fakultät für Physik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Prof. Dr. Martin Zacharias
Prüfer der Dissertation:	1. Prof. Dr. Franz Pfeiffer
	2. Prof. Dr-Ing. Jan J. Wilkens

Die Dissertation wurde am 12.05.2016 bei der Technischen Universität München eingereicht und durch die Fakultät für Physik am 30.06.2016 angenommen.

Abstract

In the past decade, ptychography has emerged as a disruptive tool for microscopy. In its simplest variant, the method scans the specimen through a coherent illumination and images the diffraction contrast for each transversal shift. As it samples the specimen in both real and reciprocal space simultaneously, ptychography creates high-dimensional and redundant images of phase space. In contrast to conventional microscopy, ptychographic imaging does not require any objective lensing system but relies on iterative phase-retrieval algorithms. They utilize the redundancy in the data to reconstruct the wave field of the illumination *and* the complex-valued transmission of the specimen at diffraction limited resolution – a feature sought after particularly for X-ray microscopy at synchrotrons.

Recent developments indicate that the high-dimensional phase space of ptychography is suited to compensate for coherence reducing effects caused by a certain degree of randomness in the measurement procedure. However, these partially coherent conditions require substantial alterations to the diffraction model of numerical phase-retrieval and also to the reconstruction algorithms employed.

This thesis condenses many of the current trends, including partial coherence, into a comprehensive forward scattering model for ptychographic phase retrieval. It presents a flexible computational framework developed in particular for non-ideal experimental conditions. This framework, called "Ptypy", is available online to any researcher in the field of ptychography and has already found a user base at synchrotrons.
Furthermore, this thesis reports about a selection of experiments and simulations in the field of partially coherent ptychography in the visible and X-ray regime of light. A broad range of applications of the extended forward scattering model are found, ranging from high-flux conditions to fast fluctuations of the specimen.
Special emphasis is given to broad-bandwidth conditions. Here, the extended model proves to be so robust such that it retrieves simultaneously both the spectral characteristics of the source and the microscopic image of the specimen.

Kurzzusammenfassung

Im letzten Jahrzehnt hat sich Ptychographie als bahnbrechende Methode in der Mikroskopie etabliert. In seiner einfachsten Variante rastert es die Probe durch eine kohärente Beleuchtung und nimmt dabei je ein Bild des Beugungskontrastes für jeden Verschiebungspunkt auf. Da es so gleichzeitig realen und reziproken Raum abtastet, entstehen hochdimensionale und redundante Bilder des Phasenraums. Im Unterschied zu konventioneller Mikroskopie benötigt ptychographische Bildgebung kein Objektivlinsensystem sondern basiert auf iterativen Phasenrekonstruktionsalgorithmen. Diese errechnen aus der Redundanz in den Daten sowohl die komplexe Phasenfront der Beleuchtung als auch die komplexe Transmissionsfunktion der Probe mit beugungsbegrenzter Auflösung. Diese Eigenschaft ist besonders für Röntgenmikroskopie an speicherringbasierten Röntgenquellen von Bedeutung.

Aktuelle Forschungsergebnisse legen nahe, dass sich der hochdimensionale Phasenraum der Ptychography gut eignet um kohärenzreduzierende Effekte zu kompensieren, die von statistischen Störungen in der Messung herrühren. Allerdings verlangt ein teilkohärentes Messverfahren substanzielle Änderungen sowohl am numerischen Modell der Beugung als auch an den Rekonstruktionsalgorithmen.

Diese Dissertation fasst aktuelle Entwicklungen – einschließlich der Teilkohärenz – in einem umfassenden erweiterten Beugungsmodell für iterative Phasenrekonstruktion zusammen. Darauf aufbauend wird eine flexible Rekonstruktionssoftware vorgestellt, die besonders die nicht-idealen Bedingungen bei aktuellen Anwendungen der Ptychography berücksichtigt. Diese Software mit Namen „Ptypy" ist online verfügbar und wird bereits heute an Speicherringen von anderen Wissenschaftlern angewandt.
Darüber hinaus berichtet diese Dissertation über eine Reihe von Simulationen und Experimenten im Themenbereich der teilkohärenten Ptychography sowohl mit sichtbarem Licht als auch mit Röntgenstrahlen. Verschiedene potenzielle Anwendungen des erweiterten Beugungsmodells werden identifiziert, die von hochintensiven Quellen bis hin zu schnellen Fluktuationen in der Probe reichen.
Ein besonderes Augenmerk richtet sich auf polychromatische, breitbandinge Lichtquellen. Hier erweist sich sich das erweiterte Beugungsmodell als derart robust, dass sich sogar spektrale Eigenschaften der Lichtquelle gleichzeitig mit dem mikroskopischen Bild der Probe erfassen lassen.

Contents

1. Introduction

In the last decade, Coherent Diffractive Imaging (CDI) has shown great promise as a dose-efficient and high-resolution complement to traditional (lens-based) X-ray microscopy (Sayre and Chapman, 1995). In contrast to conventional microscopy where an objective lensing system catches the diffracted X-ray photons to form a direct image on the detector, the X-ray photons are allowed to freely propagate to the detector forming a diffraction image (Miao et al., 1999; Nugent, 2009; Thibault and Elser, 2010).

At far distance from the specimen, the diffraction image bears an important advantage that has fuelled the research over the past years : High resolution features of the sample are mapped to large scattering angles effectively decoupling specimen resolution from the pixel density of the detector. This characteristic of diffraction imaging allows the use of efficient photon-counting area detectors (Kraft et al., 2009; Johnson et al., 2012) which have a rather large pixel size as a result of their integrated electronics.

However, the diffraction image itself is of limited use for the human observer who is not blessed by nature with inverse vision. The diffracted wave has to be (back-)propagated to the specimen plane to form a projection image of the specimen. In traditional microscopy, this task is performed by the objective lens of the microscope and must now be carried out numerically. The disadvantage of CDI is now apparent: Half of the wave field's information that has once stored the phase is lost in the intermediate acquisition process which only records the wave field's intensity. The lost phase information has to be retrieved before the numerical back-propagation delivers the desired single projection image and not a cross-correlation thereof (so-called Patterson map). This peculiarity of CDI is commonly dubbed the *phase problem* of diffraction imaging – a problem encountered early on in X-ray and electron crystallography (Sayre, 2002).

In some cases, the phase problem may be circumvented when in-plane reference structures encode the phase through interferences in the diffraction image. Possible structures range from small pinholes (Fourier Transform Holography (Eisebitt et al., 2004)) to sharp-edged extended shapes (HERALDO (Guizar-Sicairos and Fienup, 2008a; Enders et al., 2009)). An image of the specimen may then be re-

trieved in a single-step analytical process. However, these techniques abolish the original advantage of diffraction limited imaging as any reference structure will eventually impose a limit on the resolution originating from imprecision in its manufacturing process.

Research effort was therefore directed at numerically retrieving the phase for a single diffraction image applying iterative algorithms with constraint sets derived from a priori information. Gerchberg and Saxton (1972) suggested one of the first algorithms to retrieve the phase from intensity measurements in sample and diffraction plane. Later developments replaced the impractical modulus constraint in the sample plane with other constraints, like a masked modulus (support constraint) or non-negativity or a shrinking support (Fienup, 1986; Marchesini et al., 2003). A plethora of iterative projection algorithms promised uniqueness and fast convergence (Marchesini, 2007). Despite intense research, the single-pattern CDI has not become a ubiquitous microscopy technique for X-ray microscopy beyond a niche of imaging small isolated particles in Free Electron Lasers (Seibert et al., 2011). The small field of view that is impractical for larger samples may be a dominant rationale against daily usage. A good overview of possible reasons is reviewed by Dierolf (2015).

For ptychography, an important member of the CDI family, one scans a finite and coherent illumination, the probe, across a sample, collecting a diffraction image at each scan point (Hegerl and Hoppe, 1970; Rodenburg, 2008). The sample image is then reconstructed with an iterative phase retrieval algorithm which enforces self-consistency among adjacent overlapping scan points. This so-called "overlap constraint" is such a powerful constraint that it makes unnecessary other a priori knowledge on the sample or the illumination. Apart from its remarkable robustness compared to other CDI techniques, ptychography provides simultaneous reconstructions of the sample transmission function and of the probe (Thibault, 2008; Guizar-Sicairos and Fienup, 2008b; Thibault et al., 2009; Maiden and Rodenburg, 2009), both two-dimensional complex-valued quantities in general. Unlike traditional X-ray microscopy or other CDI techniques, ptychography can deliver quantitative amplitude and phase images of extended specimen beyond the resolution limit of optics.

Ptychographic data is inherently four-dimensional as it samples both the lateral shift *and* transverse component of the scattered wave vector. This four-dimensional *phase space* of the measurements is strongly constrained because it is formed from two-dimensional quantities, the wave front of the illumination and the sample transmission function. As a consequence, it suffices to sample the phase space in a sparse manner and let the phasing algorithms fill the gaps using coherent self-consistence properties of the phase-space (Edo et al., 2013).

The constraints on the phase space originate from the static nature of coherent wave fields. If dynamic spontaneous processes introduce decoherence to probe or object, the phase space becomes less constraint. Phase retrieval algorithms relying on the former self-consistence property are now incapable to accurately model the degraded data and struggle for convergence. Thibault and Menzel (2013) provided relieve by demonstrating that the phase space of partially coherent ptychography can be modelled with a mixture of states from the eigenvalue decomposition of the probe and object fields. In contrast to single-pattern CDI (Whitehead et al., 2009; Abbey et al., 2011), the additional dimensions from scanning provide enough information such that adapted phasing algorithms are able to recover the states without additional a-priori knowledge. The concept was quickly transferred to other data degradation effects such as point-spread-function and air scattering (Enders et al., 2014) or sample jitter (Clark et al., 2014).

Beyond its far-field diffraction version, ptychography is today applied to other microscopy techniques by including transverse scanning of the sample to generate data redundancy. Ptychographic principles can be transferred to the holographic regime (Stockmar et al., 2013, 2015b) or to Bragg geometry (Godard et al., 2011; Chamard et al., 2015), and have also been mapped to reciprocal space (Ou et al., 2013). In addition the quantitative nature of the technique lends itself naturally to tomographic applications, thus yielding quantitative volumetric information at the nano-scale (Dierolf et al., 2010a; Trtik et al., 2013; Zanette et al., 2015).

1.1. About this thesis

Many novel methodological developments for ptychography have in common that they extend the simple forward scattering model of coherent diffraction imaging. This thesis summarises many of the current trends including partial coherence into a comprehensive forward scattering model for ptychographic phase retrieval. It presents a flexible computational framework developed in particular for today's diverse experimental conditions. Furthermore, it reports about a selection of experiments and simulations in the field of partially coherent ptychography in the visible and X-ray regime of light. Ptychography in broad-bandwidth conditions is given special emphasis.

Chapter 2 aims at providing all mathematical relations that are necessary to understand scalar waves as a model for X-rays and scalar wave propagation in paraxial approximation. Furthermore, it concentrates on partial coherence as second order correlation of statistical optics and ptychography is introduced as imaging in four-

dimensional phase space. In the end, the chapter introduces a comprehensive forward scattering model for ptychography.

Chapter 3 presents a computational framework for ptychography, developed with complex models in mind. It is available online for academic research under the link `http://ptycho.github.io/ptypy`. With 11000 lines of instructions and a comparable amount of documentation, the volume of the software exceeds the practical volume of a thesis. Consequently, only a brief summary and overview of the main concepts are presented here, while an excerpt of the online documentation is included in the appendix.

As examples for ptychography with complex forward models, the next chapters provide results from experiments or simulations all centred around partial coherence.

Chapter 4 addresses data degradation from partial spatial coherence and vibration. Apart from published material (Enders et al., 2014) this chapter contains novel insights like an iterative deconvolution from a mode decomposition and the first experiments that require a mode decomposition of the sample's transmission function as a model for phase-retrieval.

Chapter 5 addresses data degradation from partial temporal coherence which means broadband sources. Broadband degradation and counter strategies are analysed thoroughly in simulations. As a novel proof of concept this chapter demonstrates ptychographic imaging with spectrum recovery using an ultra-broadband source – white light.

This thesis contains material which is published or is about to be published in peer-reviewed journals as first author:

- B. Enders, M. Dierolf, P. Cloetens, M. Stockmar, F. Pfeiffer, and P. Thibault. *Ptychography with broad-bandwidth radiation.* Applied Physics Letters, **104**(17), 2014. doi: 10.1063/1.4874304
  Material appears in Chapter 4 and here in this in the introduction.

- B. Enders, M. Dierolf, F. Pfeiffer, and P. Thibault. *Ptypy – A computational framework for ptychography.* Proceedings of the Royal Society A (in preparation for submission)
  Material appears in Chapter 2, in this introduction and in the theory (Sec. 2.5).

2. Theory

All light - from gamma-rays and X-rays to the visible range to heat waves and radio waves - travels through vacuum at constant velocity $c = 29\,979\,245\,\mathrm{m\,s^{-1}}$ and is both particle (*photon*) and electro-magnetic wave. The discovery of the wave particle dualism was a fundamental breakthrough of quantum mechanics at the beginning of the 20th century. Inherited from particle and wave description, light can be distinguished by wave properties such as *frequency* ν or *wavelength* λ or by particle properties such as *energy* E_{ph} and *momentum*. The energy of a single photon is related to the wave field's frequency,

$$E_{\mathrm{ph}} = h\nu = \frac{hc}{\lambda} \tag{2.1}$$

where $h = 6.626 \cdot 10^{-34}\,\mathrm{Js} = 4.136 \cdot 10^{-15}\,\mathrm{eVs}$ is Planck's constant. The photon energy increases linearly with frequency as does the significance of the quantized particle nature of light. The names given to different kinds of light emphasize this change of light properties. We speak of Gamma-*rays* but of radio *waves*. Consequently, the energy, as a single-particle property, is commonly used to characterize light in the range from gamma-rays over X-rays to the visible spectrum while frequency is more frequently used for the opposite end of the light spectrum. For example, hard X-rays span the range from a few keV to a few MeV. The cSAXS synchrotron beamline at the Swiss Light Source (SLS) operates often at roughly 6.2 keV which corresponds to a wavelength of 0.2 nm in vacuum. Red light, as used in LEDs or lasers, starts at a wavelength of 620 nm which corresponds to an energies below 2 eV.

In this thesis, light is modelled by the scalar wave. It means, that a scalar field variable $\phi(\mathbf{r}, t)$ depending on a three-dimensional space vector and a time coordinate suffices to describe all physical phenomena relevant for this thesis. Scalar wave theory is especially well suited to describe the phenomenon of *diffraction* for coherent wave fields.

Section 2.1 provides all necessary steps to derive the theory of scalar diffraction for coherent light – the de-facto standard in forward scattering geometry. Furthermore, it formulates diffraction as a propagation operator between two-dimensional planes and introduces the *projection approximation* to model wave-matter interaction as two-dimensional transmission.

Section 2.2 discusses briefly the basic principles of Coherent Diffraction Imaging like the phase-problem and sampling requirements for numerically phase retrieval.

Section 2.3 addresses the changes to the coherent diffraction theory when the wave fields exhibit a partially random and statistical behaviour. Fundamentals of 2nd order correlation theory, like the complex degree of coherence and the cross-spectral density are presented. The most important topic is the *mode* or *state* decomposition of the cross spectral density. The theorem allows the reuse of propagators from coherent scalar diffraction theory as long as the measurement is composed of an incoherent sum. The section closes with the well-known van Cittert-Zernike theorem which describes the conditions under which incoherent wave fields become coherent after free-space propagation.

The last section (2.4) introduces ptychography as sparse sampling in four-dimensional phase space. The consistent use of the four-dimensional ambiguity function provides an excellent understanding for data degradation from translational invariant sources, like vibrations or partial transverse coherence. The mixed state approach (Thibault and Menzel, 2013) is reinterpreted as a form of coherent ambiguity function syntheses of phase space. The section concludes with an overview about how the common coherent model of ptychographic phase retrieval needs to be adapted in order to comply with partial coherence and different scattering geometries.

Some basic mathematical conventions are explained in Appendix A.1. Other conventions will be explained when they are first used and if they differ from the usual terminology in physics.

2.1. Scalar waves

Goodman (2005) motivates the scalar wave model from Maxwell's equations. In *linear, isotropic, and non-magnetic* media and in absence of free charges, these equations are

$$\nabla \times \mathbf{E}(\mathbf{r},t) = -\mu_0 \frac{\partial}{\partial t}\mathbf{H}(\mathbf{r},t) \tag{2.2}$$

$$\nabla \times \mathbf{H}(\mathbf{r},t) = \varepsilon(\mathbf{r}) \frac{\partial}{\partial t}\mathbf{E}(\mathbf{r},t) \tag{2.3}$$

$$\nabla \cdot (\varepsilon(\mathbf{r})\,\mathbf{E}(\mathbf{r},t)) = 0 \tag{2.4}$$

$$\nabla \cdot (\mu_0 \mathbf{H}(\mathbf{r},t)) = 0 \tag{2.5}$$

where $\mathbf{E}$ is the electric field, $\mathbf{H}$ the magnetic field and μ and ε are the permeability and permittivity of the medium. Here, μ and ε are scalar quantities as we assumed an isotropic medium and $\mu = \mu_0 = 4\pi \cdot 10^7 \frac{N}{A^2}$ as we assumed a non-magnetic medium. The operation $\times$ represents a vector cross product and $\cdot$ a vector dot product (scalar product) if applied to vectorized quantities. ∇ denotes the three-dimensional differential operator

$$\nabla = \left(\frac{\partial}{\partial x_\mathrm{I}}, \frac{\partial}{\partial x_\mathrm{II}}, \frac{\partial}{\partial z} \right), \tag{2.6}$$

for a differentiable function

$$\psi : (x_\mathrm{I}, x_\mathrm{II}, z) \in \mathbb{R}^3 \to \psi(x_\mathrm{I}, x_\mathrm{II}, z) \in \mathbb{C}, \tag{2.7}$$

Using the vacuum permittivity $\varepsilon_0 = 8.854 \cdot 10^{-12} \frac{F}{m}$ the *refractive index* is defined as

$$n(\mathbf{r}) = \sqrt{\frac{\varepsilon(\mathbf{r})}{\varepsilon_0}}\,. \tag{2.8}$$

When we apply $(\nabla\times)$ to eq.(2.2) and use the information from eq.(2.4) that

$$\nabla \cdot \mathbf{E}(\mathbf{r},t) = -\mathbf{E}(\mathbf{r},t) \cdot \frac{\nabla\varepsilon(\mathbf{r})}{\varepsilon(\mathbf{r})} = -2\,\mathbf{E}(\mathbf{r},t) \cdot \nabla(\log n(\mathbf{r})) \tag{2.9}$$

we obtain a differential equation for the electric field alone,

$$\nabla^2\mathbf{E}(\mathbf{r},t) - \frac{n^2(\mathbf{r})}{c^2}\frac{\partial^2}{\partial t^2}\mathbf{E}(\mathbf{r},t) = -2\,\mathbf{E}(\mathbf{r},t) \cdot \nabla(\log n(\mathbf{r}))\,. \tag{2.10}$$

Eq.(2.10) can be regarded as an inhomogeneous differential equation when the right side of the equation is considered as the perturbation term. The left side of the

equation denotes a differential equation for each component, while the perturbation term on the right side couples all three components of **E**. The equations can be decoupled if we further assume that the medium is *homogeneous*. Hence, the refractive index is constant (we replace it with an average $n(\mathbf{r}) = \bar{n}$) and the right side of eq.(2.10) vanishes, yielding

$$\nabla^2\mathbf{E}(\mathbf{r},t) - \frac{\bar{n}^2}{c^2}\frac{\partial^2}{\partial t^2}\mathbf{E}(\mathbf{r},t) = 0\,. \tag{2.11}$$

A similar expression as (2.11) can be retrieved for **H**. As the differential equation holds for any component E_i of **E** and the behaviour is similar for **E** and **H**, a single scalar differential equation may be used to characterize the dynamics of electromagnetic waves in homogeneous media. The according differential equation

$$\nabla^2\phi(\mathbf{r},t) - \frac{\bar{n}^2}{c^2}\frac{\partial^2}{\partial t^2}\phi(\mathbf{r},t) = 0\,, \tag{2.12}$$

is called the *homogeneous, time-dependent Helmholtz* equation and defines how space and time of a scalar wave field are coupled. That means, it describes how a feature in the wave-field *propagates* with time. The Helmholtz equation needs to be solved independently for all of the three components of **E**
In case of monochromatic light of frequency ν, the time dependent fluctuations of $\phi(\mathbf{r},t)$ are harmonic oscillations,

$$\phi(\mathbf{r},t) = \psi(\mathbf{r},\nu)\exp(-2\pi\mathrm{i}\nu t) \tag{2.13}$$

Inserting this ansatz in eq.(2.12) disposes of the explicit time dependence,

$$c^2\,\nabla^2\psi(\mathbf{r},\nu) + (2\pi)^2\nu^2\bar{n}^2\psi(\mathbf{r},\nu) = 0\,. \tag{2.14}$$

This *homogeneous, time-independent Helmholtz* equation apparently changes with frequency which means any homogeneous medium exhibits some kind of dispersion from propagation. That statement holds true even though a non-dispersive medium was assumed in the Maxwell equation with explicitly time-independent permittivity. We get back to this topic in Section 2.3.

For now, we continue with only a single frequency, i.e. monochromatic light, and omit the frequency dependence. Instead of a time-related constant, we will use the so-called *wavenumber*

$$k = \frac{2\pi}{\lambda} = \frac{2\pi\nu\bar{n}}{c} \tag{2.15}$$

to characterize the wave. Then, the time-independent Helmholtz takes on the very concise and common form

$$\nabla^2\psi(\mathbf{r}) + k^2\psi(\mathbf{r}) = 0\,. \tag{2.16}$$

We highlight two important solutions to (2.16) here.

- One solution is the *plane wave*,

$$\psi(\mathbf{r}) = \sqrt{I} e^{\mathrm{i}\mathbf{k}\mathbf{r}}\,, \tag{2.17}$$

  where $\mathbf{k} = \mathbf{e}|\mathbf{k}| = \mathbf{e}\,k$ is the *wave-vector*, a three-dimensional vector of length k. It is called plane wave as the wave fronts (areas of equal phase) are two-dimensional *planes* normal to the wave vector who points in the propagation direction $\mathbf{e} = \mathbf{k}/k$. The wavelength λ corresponds to the distance between two wave fronts.

- Another important solution is the *spherical wave* originating from a source point $\mathbf{r}_0$

$$\psi(\mathbf{r}) = \sqrt{I} \frac{1}{|\mathbf{r} - \mathbf{r}_0|} e^{\mathrm{i}kn|\mathbf{r}-\mathbf{r}_0|}\,, \tag{2.18}$$

  which has *spheres* as areas of constant phase separated by the wavelength. The wave *diverges* from $\mathbf{r}_0$ because the sign in the exponential of (2.18) is positive - a negative sign would mean that the wave converges at $\mathbf{r}_0$. Spherical waves resemble plane waves when viewed far away from the source.

The spherical wave is one possible choice of Green's function for the Helmholtz equation, which means that

$$(\nabla^2 + k^2) G_\circledcirc(\mathbf{r} - \mathbf{r}') = \delta(\mathbf{r} - \mathbf{r}') \qquad \text{for} \qquad G_\circledcirc(\mathbf{r}) = \frac{\exp(\mathrm{i}k|\mathbf{r}|)}{4\pi|\mathbf{r}|}\,. \tag{2.19}$$

In principle, solutions to (2.10) can be generated from convolving the Green's function with the disturbance. However, in practice, the Green functions will have to obey certain boundary conditions. Furthermore, the wave field appears in the disturbance too meaning that a convolution with a Green's function would only formally solve the differential equation but not be an actual practical solution.

With the assumption of a thin screen as disturbance and the right choice of Green's function it is, however, possible to find a closed-form solution as exercised by most common optics textbooks (Goodman, 2005).

For example, if the screen was an opaque aperture at $\mathbf{r}' = (\mathbf{x}, 0)$ with an area $\mathcal{D}$, we would find the diffracted field at point $\mathbf{r}$ behind aperture ($z > 0$) to be

$$\psi(\mathbf{r}) = \frac{1}{\mathrm{i}\lambda} \int_{\mathcal{D}} \psi(\mathbf{r}')\, G_\circledcirc(\mathbf{r} - \mathbf{r}')\, \kappa(\mathbf{r} - \mathbf{r}') \mathrm{d}\mathbf{x} \tag{2.20}$$

under the assumption that the wave was a plane wave travelling in positive z-direction before the aperture. The so-called *obliquity*-factor $\kappa(\mathbf{r})$ guarantees that light is not

diffracted into the plane $z = 0$. The integral (2.20) conforms to the *Rayleigh-Sommerfeld* diffraction formula, when the obliquity factor takes on the following form

$$\kappa(\mathbf{r}) = \frac{\mathbf{r} \cdot \mathbf{e}_z}{|\mathbf{r}|} . \tag{2.21}$$

The obliquity factor is maximal ($\kappa \approx 1$) whenever the diffracted ray $\mathbf{r} - \mathbf{r}'$ is parallel to the z-axis which we will also call *optical* axis from here on. Equation (2.20) also denotes the mathematical equivalent to the *Huygens-Fresnel* principle, which constructs the secondary wave-field from spherical waves who emanate from the screen of disturbance.

2.1.1. Propagation between two parallel planes

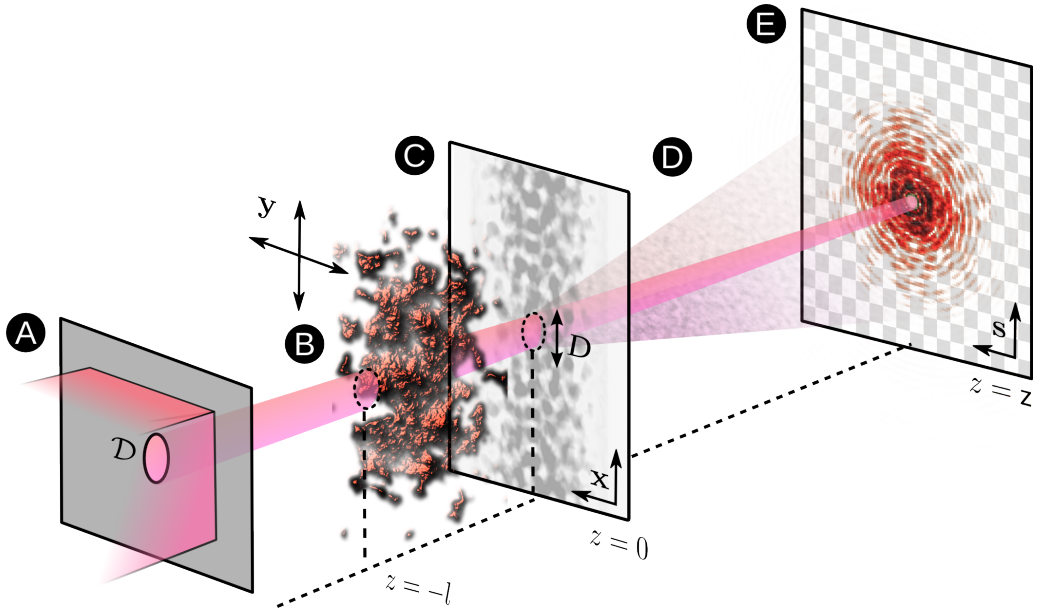


Figure 2.1. – Schematic for forward scattering in paraxial geometry. **A** Pinhole aperture to shape the incoming beam and filter a coherent probe $p(\mathbf{x})$. **B** Scattering volume $\varrho(\mathbf{r})$ extending until $z = -l$ **C** Projected transmission $o(\mathbf{x})$ onto specimen plane $\mathbf{r} = (\mathbf{x}, 0)$. **D** Free space propagation path of length z. **E** Pixelated detector to record far-field intensity image at the screen plane $\mathbf{r} = (\mathbf{s}, \mathrm{z})$.

Let us now assume we want to know the wave field in a plane $(\mathbf{s}, z)$ at distance z perpendicular to the optical axis. The wave field is assumed to be known in the plane $(\mathbf{x}, 0)$ which is where the opaque screen was situated in (2.20). If the field is bandwidth-limited, we can construct the field from a Fourier decomposition $\psi(\mathbf{s}, z) =$

$\underset{\mathbf{v},\mathbf{s}}{\mathcal{F}^*}\{\vartheta(\mathbf{v},z)\}$ where $\mathbf{v}$ are the *transverse* space frequency components. Since ψ has to fulfil the Helmholtz equation, we obtain an implicit differential equation

$$0 = (\nabla_\mathbf{s}^2 + \partial_z^2 + k^2)\underset{\mathbf{v},\mathbf{s}}{\mathcal{F}^*}\{\vartheta(\mathbf{v},z)\} = \underset{\mathbf{v},\mathbf{s}}{\mathcal{F}^*}\left\{(k^2 - (2\pi)^2\mathbf{v}^2 + \partial_z^2)\vartheta(\mathbf{v},z)\right\}\,, \tag{2.22}$$

which has

$$\vartheta(\mathbf{v},z) = \exp\left(2\pi\mathrm{i}\,z\sqrt{\lambda^{-2} - \mathbf{v}^2}\right)\vartheta(\mathbf{v},0)\,, \tag{2.23}$$

as elementary and consistent solution. $\vartheta(\mathbf{v},0)$ is the Fourier transform of the field distribution in the source plane if that field is space-limited in $\mathbf{x}$. The factor

$$H(\mathbf{v}) = \exp(2\pi\mathrm{i}\,z\sqrt{\lambda^{-2} - \mathbf{v}^2}) \tag{2.24}$$

acts as a filter on the so-called *angular spectrum* $\vartheta(\mathbf{v},0)$ when the wave is propagated from the source plane to the screen. In principle, the filter is the Fourier transform of the impulse response of the medium (2.19) with respect to $\mathbf{x}$.
Now we can write the propagation from source to screen as an integral operator that involves two Fourier transforms,

$$\psi(\mathbf{s},z) = \underset{\mathbf{x},\mathbf{s}}{\mathcal{P}}(\lambda,z)\{\psi(\mathbf{x})\} = \underset{\mathbf{v},\mathbf{s}}{\mathcal{F}^*}\left\{\underset{\mathbf{x},\mathbf{v}}{\mathcal{F}}\{\psi(\mathbf{x},0)\}\,\exp\left(2\pi\mathrm{i}\,z\sqrt{\lambda^{-2} - \mathbf{v}^2}\right)\right\}\,, \tag{2.25}$$

which we call *angular spectrum propagator* from here on.

For the applicability of the angular spectrum propagator we assumed that the transverse components of ψ are limited both in extent and spatial bandwidth, i.e.

$$|\psi(\mathbf{x},0)| \approx 0 \quad \text{for} \quad |\mathbf{x}| > \frac{D}{2} \qquad \text{and} \qquad |\vartheta(\mathbf{v},0)| \approx 0 \quad \text{for} \quad |\mathbf{v}| > V \tag{2.26}$$

Since ϑ and ψ are Fourier transforms of each other, conditions (2.26) cannot be true at the same time in the strict sense. However, many fields fall sufficiently rapidly for large transverse components in space and frequency, such that conditions (2.26) are approximately true. We will call the product DV of the signal extent D and maximum occupied spatial frequency V as the *space-bandwidth product* which is a measure for the complexity of a signal (Goodman, 2005).

In the following chapters, the free-space propagation distance between source and screen is usually the fixed distance $z = \mathrm{z}$. Hence, from now on we use,

$$\psi^\mathrm{z}(\mathbf{s}) \equiv \psi(\mathbf{s},\mathrm{z}) \qquad \text{and} \qquad \psi^0(\mathbf{x}) \equiv \psi(\mathbf{x},0) \tag{2.27}$$

as a form of short notation for the wave field at observation point $\mathbf{s} = (s_\mathrm{I}, s_\mathrm{II})$ on the screen and at point $\mathbf{x} = (x_\mathrm{I}, x_\mathrm{II})$ at the source plane respectively. The superscript indices are dropped whenever it does not lead to ambiguity. The explicit dependence of the propagator $\mathcal{P}$ on distance z and wavelength λ as well as the transformation from $\mathbf{x}$ to $\mathbf{s}$ is also implied if not noted otherwise.

2.1.2. Paraxial approximations

If the wave-field in the source plane only has the lower frequency components populated in comparison to the inverse wavelength, i.e. the $|\mathbf{v}| \ll 1/\lambda$, the angular spectrum filter is slowly varying. In this case, the Fresnel approximation $\sqrt{\lambda^{-2} - \mathbf{v}^2} \sim \lambda^{-1} - 0.5\lambda\mathbf{v}^2$ holds:

$$\exp\left(2\pi\mathrm{i}\, z\sqrt{\lambda^{-2} - \mathbf{v}^2}\right) = \exp\left(\mathrm{i}kz\right) \exp\left(-\pi\mathrm{i}\, z\lambda\mathbf{v}^2\right) . \tag{2.28}$$

The angular spectrum propagator in real space coordinates becomes

$$\psi(\mathbf{s}) = \mathcal{P}\{\psi(\mathbf{x})\} = \frac{\exp\left(\mathrm{i}kz\right)}{\mathrm{i}\lambda z} \int \psi(\mathbf{x}) \exp\left[\frac{\pi\mathrm{i}\,(\mathbf{s}-\mathbf{x})^2}{\lambda z}\right] \mathrm{d}\mathbf{x}\,, \tag{2.29}$$

where the convolution kernel

$$G_{\asymp}(\mathbf{x}, z) = \frac{\exp\left(\mathrm{i}kz\right)}{4\pi z} \exp\left[\frac{\pi\mathrm{i}\,\mathbf{x}^2}{\lambda z}\right] \tag{2.30}$$

is a fundamental solution to the paraxial Helmholtz equation

$$\left(\nabla_{\mathbf{x}}^2 + 2\mathrm{i}k\,\frac{\partial}{\partial z} + 2k^2\right) \psi(\mathbf{x}, z) = 0\,. \tag{2.31}$$

The shape of (2.30) indicates that the wave field in paraxial approximation can be seen as a carrier wave $\exp(\mathrm{i}kz)$ modulated by an envelope function $\psi_{=}(\mathbf{x}, z)$ for which we find the Helmholtz equation to be

$$\left(\nabla_{\mathbf{x}}^2 + 2\mathrm{i}k\,\frac{\partial}{\partial z}\right) \psi_{=}(\mathbf{x}, z) = 0\,, \tag{2.32}$$

with the fundamental solution

$$G_{=}(\mathbf{x}, z) = \frac{1}{4\pi z} \exp\left[\frac{\pi\mathrm{i}\,\mathbf{x}^2}{\lambda z}\right] . \tag{2.33}$$

The Fresnel propagator (2.29) does not need to be calculated from a (computational expensive) convolution. Expanding the quadratic exponential $(\mathbf{s}^2 - \mathbf{x}^2)$ yields a single Fourier transform

$$\psi(\mathbf{s}) = \mathcal{P}\{\psi(\mathbf{x})\} = \frac{\exp\left(\mathrm{i}kz\right)}{\mathrm{i}\lambda\mathrm{z}} \exp\left[\frac{\pi\mathrm{i}\,\mathbf{s}^2}{\lambda\mathrm{z}}\right] \mathcal{F}_{\mathbf{x}}\left\{\psi(\mathbf{x}) \exp\left[\frac{\pi\mathrm{i}\,\mathbf{x}^2}{\lambda\mathrm{z}}\right]\right\}\left(\frac{\mathbf{s}}{\lambda\mathrm{z}}\right) \tag{2.34}$$

The Fresnel propagator (2.34) takes on a simpler form if the quadratic factor in the Fourier transform can be approximated with unity. This is the case if the so-called *Fresnel-number*

$$F = \frac{D^2}{4\lambda\mathrm{z}} \tag{2.35}$$

is much smaller than one. The case $F \ll 1$ is usually fulfilled if the source plane is far away from the screen. The approximation

$$\psi(\mathbf{s}) = \mathcal{P}\{\psi(\mathbf{x})\} = \frac{\exp(\mathrm{i}kz)}{\mathrm{i}\lambda z} \exp\left[\frac{\pi \mathrm{i}\,\mathbf{s}^2}{\lambda \mathrm{z}}\right] \mathcal{F}\psi\left(\frac{\mathbf{s}}{\lambda \mathrm{z}}\right) \tag{2.36}$$

is thus called the *far-field* or *Fraunhofer* approximation to the Fresnel-Kirchhoff diffraction integral.

The detector in the plane of the screen eventually records the intensity of the wave field,

$$I(\mathbf{s}) = \frac{1}{\lambda^2 \mathrm{z}^2} \left|\mathcal{F}\psi\left(\frac{\mathbf{s}}{\lambda \mathrm{z}}\right)\right|^2 , \tag{2.37}$$

which is constant with respect to *scattering angle* $\mathbf{s}/\mathrm{z}$ or *momentum transfer*

$$\mathbf{q} = \frac{\mathbf{s}}{\lambda \mathrm{z}} \tag{2.38}$$

which are the reciprocal arguments to $\mathbf{x}$ in the Fourier transform. Therefore, it is sometimes more practical to express the intensity in terms of reciprocal coordinates,

$$I(\mathbf{q}) = I(\mathbf{s}(\mathbf{q}))\left|\frac{\partial \mathbf{s}}{\partial \mathbf{q}}\right| = \frac{1}{\lambda^2 \mathrm{z}^2} |\mathcal{F}\psi(\mathbf{q})|^2 \lambda^2 \mathrm{z}^2 = |\mathcal{F}\psi(\mathbf{q})|^2 . \tag{2.39}$$

The vector $2\pi\mathbf{q}$ can also be regarded the transverse component of the wave vector $\mathbf{k}$ of a plane wave scattered from the sample plane origin and recorded at $\mathbf{s}$ in the detector plane.

The prerequisite of a limited bandwidth for paraxial approximation translates into a limited solid angle into which the wave diffracts. Corresponding with (2.26), there is a maximum momentum transfer Q such that

$$|I(\mathbf{q})| \approx 0 \quad \text{for} \quad |\mathbf{q}| > Q , \tag{2.40}$$

and λQ is the maximum scattering angle that still carries signal. The inverse of the maximum momentum transfers is commonly considered as the smallest feature $\mathrm{d_x}$ of the wave front and thus Q also reprents the *resolution*-limit of the diffraction setup.

2.1.3. Interaction with matter

In microscopy, the probing light waves eventually interact with matter and will therefore pass through inhomogeneities of the refractive index. Since light waves oscillate

at comparatively high frequencies, we can argue that the terms on left side of eq.(2.10) are considerably larger than the perturbation term. In fact, the left side scales with k^2 as can be seen in the homogeneous Helmholtz equation (2.16) while the coupling perturbation term on right side scales linear with k due to the single gradient. Therefore, the coupling is only relevant when the refractive index changes fast (within length λ) and when the wave field is observed close (also within a few λ) to that fluctuation. Otherwise we may safely neglect the coupling term and obtain

$$\nabla^2\psi(\mathbf{r}) + k^2\frac{n^2(\mathbf{r})}{\bar{n}^2}\psi(\mathbf{r}) = 0\,, \tag{2.41}$$

which is the homogeneous Helmholtz equation for a smoothly varying medium in which $k^2n^2(\mathbf{r})/\bar{n}^2$ acts as a potential. We can formally attribute the changes in refractive index in eq.(2.41) to be a perturbation for the Helmholtz equation in homogeneous matter (2.16). The form

$$\nabla^2\psi(\mathbf{r}) + k^2\psi(\mathbf{r}) = -f(\mathbf{r}) \tag{2.42}$$

with

$$f(\mathbf{r}) = k^2\left(\frac{n^2(\mathbf{r})}{\bar{n}^2} - 1\right)\psi(\mathbf{r}) \equiv k^2\varrho(\mathbf{r})\psi(\mathbf{r}) \tag{2.43}$$

has the advantage to possess a formal solution with help of a Green's function (2.19). We assume again that the paraxial approximation holds and detach the carrier wave, which allows us to use the Green's function (2.33). Instead of an opaque screen, we assume that all inhomogeneities ϱ are confined to the interval $z \in [-l, 0]$ and that either their lateral extent or that of the wave ψ fits into an area of size $\mathcal{D}$ around the optical axes. Given these conditions, we can formally write the contribution from the inhomogeneity as

$$u(\mathbf{r}) = \int_{\mathcal{D}}\int_{-l}^{0} G(\mathbf{r}-\mathbf{r}')f(\mathbf{r}')\mathrm{d}\mathbf{x}'\,\mathrm{d}z'\,. \tag{2.44}$$

Although eq.(2.44) is formally a solution to eq.(2.41) it is not yet very practical as it also applies for $\mathbf{r} \in \mathcal{D} \times [-l, 0]$ which means that eq.(2.41) denotes an integral equation. In terms of the Huygen-Fresnel principle, it means that the secondary waves generated will produce secondary waves themselves, etc.
The problem can be alleviated to some extent in paraxial approximation. With division by the carrier wave, we apply (2.33) as Green function and obtain at far

enough distance from the disturbance (when $z \gg l$)

$$u(\mathbf{s},\mathrm{z}) = \frac{1}{4\pi\mathrm{z}} \int_{\mathcal{D}} \exp\left[\frac{\pi\mathrm{i}\,(\mathbf{s}-\mathbf{x})^2}{\lambda\mathrm{z}}\right] \left\{ \int_{-l}^{0} k^2 \varrho(\mathbf{x},z')\psi_{=}(\mathbf{x},z')\mathrm{d}z' \right\} \mathrm{d}\mathbf{x} \tag{2.45}$$

$$= \frac{1}{\mathrm{i}\lambda\mathrm{z}} \int \exp\left[\frac{\pi\mathrm{i}\,(\mathbf{s}-\mathbf{x})^2}{\lambda\mathrm{z}}\right] \left\{ \int_{-l}^{0} \frac{\partial\psi_{=}}{\partial z}(\mathbf{x},z')\mathrm{d}z' - \frac{\mathrm{i}}{2k} \int_{-l}^{0} \nabla_{\mathbf{x}}^2 \psi_{=}(\mathbf{x},z')\mathrm{d}z' \right\} \mathrm{d}\mathbf{x} \tag{2.46}$$

where the second term in the integral can be neglected for high energies or smooth wave fronts. Hence, the inhomogeneous contribution to the wave field is the propagated difference of the wave field u before and after the disturbance.

$$u(\mathbf{s},\mathrm{z}) = \mathcal{P}\{\psi_{=}(\mathbf{x},0) - \psi_{=}(\mathbf{x},-l)\} \tag{2.47}$$

For $\psi_{=}$ within the volume $\mathcal{D} \times [-l,0]$, Thibault (2008) assumes that the effect of the disturbance also acts multiplicatively as envelope o on the probing wave field $p_{=}$ which itself is a homogeneous solution:

$$\psi_{=}(\mathbf{r}) = p_{=}(\mathbf{r})o(\mathbf{r}) \qquad \text{for} \qquad \mathbf{r} \in \mathcal{D} \times [-l,0]\,. \tag{2.48}$$

After some calculations and estimations, the change in $\partial_z\psi_{=}$ arises mainly from changes in the additional envelope o,

$$k^2\varrho(\mathbf{r})o(\mathbf{r}) \approx 2\mathrm{i}k\frac{\partial}{\partial z}o(\mathbf{r}) \tag{2.49}$$

which implies

$$o(\mathbf{x},z) = \exp\left[\int_{-l}^{z} \frac{k}{2\mathrm{i}} \left(\frac{n^2(\mathbf{x},z')}{\bar{n}^2} - 1\right) \mathrm{d}z'\right] \quad \text{for} \quad z \in [-l,0]\,. \tag{2.50}$$

Equation (2.50) is commonly called the *projection approximation* which applies to multiplicative changes on a probing wave p within the paraxial approximation. The projection approximation fails, when the field is observed close to the disturbance and when the probing wave front changes significantly within the disturbance.

Finally, the wave field in the plane of the screen can be approximated as

$$\begin{aligned} \psi_{=}(\mathbf{s},\mathrm{z}) &= p(\mathbf{s},\mathrm{z}) + u(\mathbf{s},\mathrm{z}) \\ &= \mathcal{P}\{\psi_{=}(\mathbf{x},0) - p(\mathbf{x},-l) + p(\mathbf{x},0)\} \\ &\approx \mathcal{P}\{p_{=}(\mathbf{x},0)\, o(\mathbf{x},0)\} \end{aligned} \tag{2.51}$$

which now has the same shape as the diffraction of a scalar wave after a thin screen. We drop the subscript "=" as the carrier wave is usually not needed for paraxial

scattering geometry. Together with the short notions in (2.27), the general model for forward scattering can be written as

$$\psi(\mathbf{s}) = \mathcal{P}\{\psi(\mathbf{x})\} = \mathcal{P}\{p(\mathbf{x})o(\mathbf{x})\}\,, \tag{2.52}$$

where we will denote $\psi(\mathbf{x}) = p(\mathbf{x})o(\mathbf{x})$ as the *exit wave*.

2.1.4. X-ray scattering and absorption

If the refractive index differs only slightly from the average, linear approximation yields

$$\frac{n(\mathbf{x},z)}{\bar{n}} \approx 1 - \delta(\mathbf{x},z) + \mathrm{i}\beta(\mathbf{x},z) \;\Rightarrow\; \frac{n^2(\mathbf{x},z)}{\bar{n}^2} \approx 1 - 2\delta(\mathbf{x},z) + 2\mathrm{i}\beta(\mathbf{x},z)\,, \tag{2.53}$$

where δ and β are real quantities. As a consequence, the specimen's transmission function takes on a simpler form,

$$o(\mathbf{x}) = a_o(\mathbf{x})e^{\mathrm{i}\varphi_o(\mathbf{x})} \approx \exp\left[\int_{-l}^{0} \mathrm{i}k\delta(\mathbf{x},z)\right] \exp\left[-\int_{-l}^{0} k\beta(\mathbf{x},z))\,\mathrm{d}z\right]\,, \tag{2.54}$$

where we will call a_o the *absorption image* and φ_o the *phase image* of o. The so-called refractive index decrement δ is thus related to a phase shift and β relates to attenuation.

In the hard X-ray regime, the linear approximation for the refractive index holds for almost all solid matter. We may also safely assume $\bar{n} = 1$ if we take the ambient air as reference. X-ray interaction with matter occurs on a microscopic level with the electronic structure of the material, and δ and β originate from different microscopic properties.

The origin of δ is called small angle forward Thomson scattering. Electrons are excited individually by the oscillation of the incoming wave and radiate spherical waves as response. Since the response stems from individual electrons, Thomson scattering is directly proportional to the electron density $\varrho_e(\mathbf{x},z)$ of the material:

$$\delta(\mathbf{x},z) = \varrho_e(\mathbf{x},z)\frac{r_e\lambda^2}{2\pi}\,, \tag{2.55}$$

where $r_e = 2.818 \times 10^{-15}\,\mathrm{m}$ is the classical electron radius and λ the vacuum wavelength. This approximation for δ holds only far from absorption edges. Otherwise the complex atomic scattering factors need to be considered.

The other part β essentially depends on the accumulation of all absorption cross sections σ_{abs}, which includes the dominant photoelectric absorption and minor contribution from Compton scattering or coherent scattering.

$$\beta(\mathbf{x}, z) = \varrho_a(\mathbf{x}, z)\sigma_{\text{tot}}\frac{\lambda}{4\pi}\,, \tag{2.56}$$

where $\varrho_a(\mathbf{x}, z)$ is the atomic density. The absorption cross section usually depends on element and wavelength.

2.2. Coherent diffraction imaging

In the 80's of the last century, it became apparent that the phase problem associated with the recovery of the transmission of isolated non-periodic objects from their Fourier modulus may be uniquely solved (Gerchberg and Saxton, 1972; Fienup, 1986, 1982; Bates, 1982). It funded the scientific field of *coherent diffraction imaging (CDI)* with important applications in microscopy. Twenty years later, several overview articles demonstrated that the problem of iterative phase-retrieval is very well understood (Spence et al., 2002; Elser, 2002, 2003; Bauschke et al., 2002; Marchesini, 2007).

Here, we will not go into detail, but instead briefly introduce the fundamentals of single-pattern[1] CDI, starting from an explanation of the phase problem in Sec. 2.2.1, over sampling requirements in Sec. 2.2.2, to numerical phase retrieval in Sec. 2.2.3 with emphasis on the difference map algorithm in Sec. 2.2.4.

Though a member of CDI, ptychography is covered later in Sec. 2.4.

2.2.1. The phase problem

The general mission of CDI is to retrieve a complex-valued field ψ from a measurement of just its Fourier amplitudes,

$$\mathsf{M}(\mathbf{v}) = |\mathcal{F}\psi(\mathbf{v})|^2 \quad . \tag{2.57}$$

If the phase $\varphi_\psi(\mathbf{v})$ was known additionally, ψ could be retrieved simply from Fourier inversion,

$$\psi(\mathbf{x}) = \mathcal{F}^*\left\{\sqrt{\mathsf{M}(\mathbf{v})}\, e^{\mathrm{i}\phi_\psi(\mathbf{v})}\right\} . \tag{2.58}$$

[1] Sometimes it is also called *single-shot* CDI.

In the forward-scattering geometry (see Fig. 2.1), ψ corresponds to the exit wave, $\psi(\mathbf{x}) = p(\mathbf{x})\, o(\mathbf{x})$ and the Fourier amplitudes can be measured in the far-field $\mathbf{v} = \mathbf{q}$. If the illumination p was now suitably flat within a region of interest, ψ readily contains an image of the investigated specimen. However, the absence of phase information in the measurement and the inability to unambiguously retrieve it from a single diffraction pattern is commonly called the *phase problem* of CDI.

The actual real space quantity that is directly retrievable from the intensity measurement by Fourier inversion is the *auto-correlation*,

$$(\psi \star \psi)(\mathbf{x}) = \mathcal{F}^{*}\{\mathsf{M}(\mathbf{v})\} \tag{2.59}$$

where "$\star$" is the cross-correlation (see eq.(A.5)). The auto-correlation in eq.(2.59) is also called Patterson (function) in crystallography (Patterson, 1935). If a peak-like offset reference was added to ψ as done in Fourier Transform Holography or other holographic variants (Eisebitt et al., 2004; Guizar-Sicairos and Fienup, 2008a; Enders et al., 2009), the Patterson function may contain easily accessible reconstructions in specific regions. In general, it is not possible to directly retrieve an image of a non-symmetric, continuous specimen without additional peak-references.

2.2.2. Sampling conditions for far-field imaging

In paraxial approximation, the cross section of the exit wave is of limited extent D. Sayre (1952) noted that for the extent $2D$ of the Patterson function it theoretically suffices to take samples every $1/2D$ in reciprocal space $\mathbf{v}$ due to the *Nyquist-Shannon* sampling theorem (Shannon, 1949). Assuming that the reciprocal space is sampled with a pixelated detector in the far-field $\mathbf{q} = (q_{\mathrm{I}}, q_{\mathrm{II}})$, we write the Shannon sampling,

$$\triangle q_{\mathrm{I}} = \frac{\triangle s_{\mathrm{I}}}{\lambda z} < \frac{1}{2D} \quad \text{and} \quad \triangle q_{\mathrm{II}} = \frac{\triangle s_{\mathrm{II}}}{\lambda z} < \frac{1}{2D} \tag{2.60}$$

where $\triangle s_{\mathrm{I}}$ and $\triangle s_{\mathrm{II}}$ are the pixel sizes of the detector, and

$$\mathsf{M}_{\mathsf{ab}} = \mathsf{M}(\mathbf{q}_{\mathsf{ab}}) = \mathsf{M}(\mathsf{a}\triangle q_{\mathrm{I}}, \mathsf{b}\triangle q_{\mathrm{II}}) \quad \text{with} \quad \mathsf{a}, \mathsf{b} \in \mathbb{Z} \tag{2.61}$$

are the measured intensities on the grid $\mathbf{q}_{\mathsf{ab}}$. The direct space Patterson function may then be retrieved by placing the `sinc` function at each sampling point in reciprocal space and Fourier inverse the approximated function:

$$(\psi \star \psi)(\mathbf{x}) \simeq \underset{\mathbf{q},\mathbf{x}}{\mathcal{F}^{*}} \left\{ \sum_{\mathsf{a}=-\infty}^{\infty} \sum_{\mathsf{b}=-\infty}^{\infty} \mathsf{M}_{\mathsf{ab}} \operatorname{sinc}(2D\,(\mathbf{q} - \mathbf{q}_{\mathsf{ab}})) \right\}, \tag{2.62}$$

where the input to the inverse Fourier transform is know as *Whittaker-Shannon* interpolation formula. Eq. (2.62) can also be interpreted as a convolution of a Dirac-comb with the `sinc` function.

$$(\psi \star \psi)(\mathbf{x}) \simeq \operatorname{rect}\left(\frac{\mathbf{x}}{2D}\right) \sum_{\mathsf{a}=-\infty}^{\infty} \sum_{\mathsf{b}=-\infty}^{\infty} \mathsf{M}_{\mathsf{ab}}\, e^{2\pi \mathrm{i}\, \mathbf{x}\mathbf{q}_{\mathsf{ab}}} \,. \tag{2.63}$$

If the paraxial approximation holds, ψ is bandwidth-limited and M only extends to a maximum angle Q. This implies that a finite number $\mathsf{N}^2 \in \mathbb{N}$ suffices to cover the full solid diffraction angles of M. Assuming that both extent and sampling of the signal on the screen is the same for both axes, i.e. $\triangle q_{\mathrm{I}} = \triangle q_{\mathrm{II}} = \triangle q$ we choose N high enough such that

$$\frac{\mathsf{N}}{2} \triangle q > Q = \frac{1}{\mathrm{d_x}} \,. \tag{2.64}$$

Apparently, the Fourier series on the right side of

$$(\psi \star \psi)(\mathbf{x}_{\mathsf{kl}}) \simeq \operatorname{rect}\left(\frac{\mathbf{x}_{\mathsf{kl}}}{2D}\right) \sum_{\mathsf{a}=-\mathsf{N}/2}^{\mathsf{N}/2-1} \sum_{\mathsf{b}=-\mathsf{N}/2}^{\mathsf{N}/2-1} \mathsf{M}_{\mathsf{ab}}\, e^{2\pi \mathrm{i}\, \mathbf{x}_{\mathsf{kl}}\mathbf{q}_{\mathsf{ab}}} \tag{2.65}$$

is equivalent to a two-dimensional *discrete Fourier transform (DFT)* for all samples of the Patterson function on the discrete grid $\mathbf{x}_{\mathsf{kl}} = (\mathsf{k} \triangle x, \mathsf{l} \triangle x)$ if

$$\triangle x \, \triangle q = \frac{1}{\mathsf{N}} \,. \tag{2.66}$$

Eq. (2.65) is an important simplification as it allows for space-bandwidth limited waves to compute the propagation to the far-field with a DFT. A DFT can be implemented in computationally effective *fast Fourier transform (FFT)* algorithms (Cooley and Tukey, 1965) which are standard elements of most platforms and programming languages.
Even though a discrete transform restricts the amount of available samples in real space, we cannot accidentally miss the smallest feature. Comparing eq.(2.66) with (2.64) reveals that

$$\triangle x < \frac{\mathrm{d_x}}{2} \,. \tag{2.67}$$

It means that the sampling in real space is always twice as fine when compared to what we expect as smallest feature. It also means that the sampling in real space is not to be mistaken for the actual resolution of the diffraction setup.

2.2.3. Basic iterative phase retrieval

A detector with N pixels for each axis provides N^2 real-valued samples in reciprocal space and effectively the same number of samples for the Patterson function $\psi \star \psi$

with a spacing $\triangle x$ in real space. The Patterson function is complex-valued but also centro-symmetric and thus the information content after the Fourier transform is preserved. As prerequisite for Shannon sampling, we assumed that the exit wave ψ is of limited extent D and thus confined to a support,

$$\operatorname{supp}(\psi) \subset \left\{ \mathbf{x} \in \mathbb{R}^2 \,|\, \operatorname{rect}(\mathbf{x}/D) = 1 \right\} =: S \tag{2.68}$$

that fits into a square with side length D. Since cross correlation and exit wave are sampled on the same real-space grid we can assume that the square support S in (2.68) only holds $\frac{\mathsf{N}^2}{4}$ complex-values samples (or $\frac{\mathsf{N}^2}{2}$ real-valued samples) in total. This property is called *oversampling* and lays the basis to retrieve the smaller number of unknown variables of the exit wave from the greater number of knowns in the measurement. While proper Shannon sampling of the Patterson function should provide enough information to uniquely solve the phase problem (Bates, 1982), the support may be larger by a factor of $\sqrt{2}$ until the exit wave contains half the number of complex-valued variables as the measurement provides real-valued samples (Miao et al., 1997; Miao and Sayre, 2000).

Even if the information content is preserved from proper sampling, the phase problem still has to be solved with a suited algorithm. Such algorithms are implemented with alternating *projections* onto constraint sets. For example, (2.68) can be used to define the so-called *support projector* (Fienup, 1986) which is a function π_S with the properties

$$\pi_S(\psi(\mathbf{x})) = \begin{cases} \psi(\mathbf{x}) & \text{if } \mathbf{x} \in S \\ 0 & \text{otherwise}\,. \end{cases} \tag{2.69}$$

Eq.(2.69) spans a subset of exit waves with

$$C_S := \{\psi \,|\, \pi_S(\psi) = \psi\} \tag{2.70}$$

which defines the *constraint set* of the support projector. The function π_S is called a projector as it is idempotent, $\pi_S \circ \pi_S = \pi_S$ and the projection $\pi_S(\psi)$ is closest to ψ while laying in the constraint set C_S.

The other projector stems from the measurement and replaces the Fourier modulus of ψ with the measured intensities,

$$\pi_F(\psi(\mathbf{x})) = \mathcal{F}^* \left\{ \sqrt{\mathsf{M}(\mathbf{q})} \frac{\mathcal{F}\psi(\mathbf{q})}{|\mathcal{F}\psi(\mathbf{q})|} \right\} \tag{2.71}$$

and is therefore often called *Fourier (modulus) projector*. The constraint set is given by $C_F := \{\psi \,|\, \pi_F(\psi) = \psi\}$.

One way to solve the phase-problem is to alternate between the two constraint sets C_S and C_F by iteratively applying their projectors onto the current estimate ψ^{it}. The most basic projection-based algorithm is an iterative update rule of the shape

$$\psi^{\mathsf{it}+1} = \pi_S \circ \pi_F(\psi^{\mathsf{it}})\,, \tag{2.72}$$

which is called the *error-reduction algorithm* (Gerchberg and Saxton, 1972; Fienup, 1986). The solution ψ_{sol} is found where

$$\pi_S(\psi_{\text{sol}}) = \psi_{\text{sol}} = \pi_F(\psi_{\text{sol}}) \tag{2.73}$$

as the solution is anticipated to be invariant under either projection. Apparently, the solution is a fixed point ($\psi^{\mathsf{it}+1}_{\text{fix}} = \psi^{\mathsf{it}}_{\text{fix}}$) to the update rule (2.72). But a fixed point is not necessarily the unique solution. For any ψ we note that $\pi_F(\psi)$ and $\pi_F(-\psi)$ are in C_F but not their linear combinations, $\psi_\alpha = (1-\alpha)\psi + \alpha(-\psi))$, $\alpha \in [0,1]$, making C_F a non-convex set. Depending on other symmetries, the Fourier projector may let the algorithm stagnate at or around a local minimum which is a fixpoint to the algorithm, but not the solution.

2.2.4. Difference Map

Elser (2002, 2003) formulated a meta algorithm to alleviate the problem of stagnation in local minima. For any two projections π_S, π_F, an iterated estimate ψ^{it} is found through repeated application of the *difference map (DM)*:

$$\psi^{\mathsf{it}+1} = \mathcal{D}(\psi^{\mathsf{it}}) := \psi^{\mathsf{it}} + \beta(\pi_F \circ f_S - \pi_S \circ f_F)(\psi^{\mathsf{it}})\,, \tag{2.74}$$

with $\alpha \in \mathbb{R} \setminus \{0\}$. Each projection π_i is connected with a map

$$f_i = (1+\eta_i)\pi_i - \eta_i, \quad (i = S, F)\,, \tag{2.75}$$

where η_F and η_S are real-valued. A fixed point of the difference map $\psi_{\text{fix}} = \mathcal{D}(\psi_{\text{fix}})$ is *not* identical to a solution of the form (2.73) as it was for the error reduction alogrithm. Still, a solution ψ_{sol} exists if and only if a fixed point for $\mathcal{D}$ can be found. At a fixed point the difference on the right side (2.74) vanishes yielding the identity

$$(\pi_F \circ f_S)(\psi_{\text{fix}}) = (\pi_S \circ f_F)(\psi_{\text{fix}}) = \psi_{\text{sol}}\,. \tag{2.76}$$

Eq.(2.76) implies condition (2.73) as can be readily verified through additional application of either π_S or π_F.

The benefit of the difference map rests in the separation of the manifold of fixed points from the actual (unique) solution. These fixed points can be made largely attractive

with appropriate choices of β, η_S and η_F, thus avoiding stagnation in local minima. For example, the DM implementation (see (2.201)) in ptychography uses the set

$$\beta = 1, \quad \eta_S = 1 \quad \text{and} \quad \eta_F = -1\,. \tag{2.77}$$

Convergence of a DM algorithm towards a fixed point can be conveniently traced by the error metric

$$\epsilon_{DM} := |\beta| \left\| (\pi_F \circ f_S - \pi_S \circ f_F)(\psi^{\mathsf{it}}) \right\| = \left\| \psi^{\mathsf{it}+1} - \psi^{\mathsf{it}} \right\| , \tag{2.78}$$

which we will call *difference map error*. $\|\cdot\|$ denotes the L^2 norm. Once the error is sufficiently small, the solution is found through a final application of $\pi_F \circ f_S$ or $\pi_S \circ f_F$ as in eq. (2.76)

2.3. Less coherent diffraction imaging

Up to now, we considered sources, light-matter interaction and propagation under *coherent* conditions. This means that the wave field ψ probed at any point will oscillate with a single frequency at fixed amplitude and fixed phase relative to all other points in the field – the correlation in space-time is maximal. As a consequence, diffraction phenomena (fringes,speckles) are observable when taking an intensity measurement.

In nature, however, diffraction is rarely observed such that the former assumption of a completely coherent scalar field for light propagation may be considered as somehow artificial and strict. Most natural light sources emit in a spontaneous statistical manner over a broad energy range. Furthermore, the illuminated medium cannot always be considered at rest with respect to the incoming optical field (at least on the molecular level) and is rarely non-dispersive.

This section discusses what happens to the diffraction contrast when the statistical randomness and dispersive nature of the physical reality is taken into account. We will see that, when not far away from complete coherence, the model of the diffraction contrast may be constructed from a set of monochromatic, spatially coherent sources and disturbances.

$$I(\mathbf{s}) = \sum_{m,n} \int_{-\infty}^{\infty} \left| \mathcal{P}(\nu) \left\{ p_m(\mathbf{x}, \nu) \cdot o_n(\mathbf{x}, \nu) \right\} \right|^2 \mathrm{d}\nu\,. \tag{2.79}$$

We call o_n and p_m *object-* and *probe-modes* respectively. Each mode belongs to a *convergent* series of mutually orthogonal functions oscillating with frequency ν. The propagator $\mathcal{P}(\nu)$ is the coherent paraxial propagator as derived in Sec. 2.1.2.

The contents of the following section contain to a large part textbook knowledge in the field of *statistical optics* (Born et al., 1999; Wolf, 2007; Goodman, 1985) rephrased to fit with a paraxial dual-plane geometry. In particular Section 2.3.4 follows the remarks and conclusions of Wolf (1982) where the mode expansion of the cross-spectral density is derived.

2.3.1. Mutual coherence

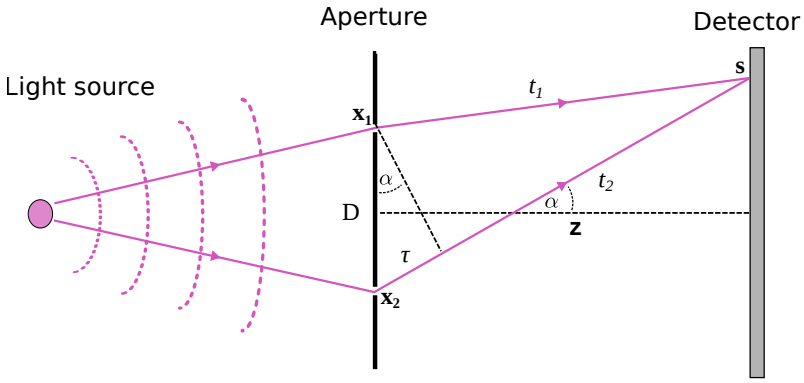


Figure 2.2. – Schematic for the correlation of a stationary wave field in paraxial geometry. A detailed explanation is given in the main text body.

Let us consider an illustrative example sketched in Fig. 2.2. An opaque screen with two pinholes present at $\mathbf{r}_2 = (\mathbf{x}_1, 0)$ and $\mathbf{r}_2 = (\mathbf{x}_2, 0)$ is illuminated by light under stationary conditions. The complex amplitude of the light fluctuations in the screen at any time t may be represented by $\phi(\mathbf{r}_1, t)$ and $\phi(\mathbf{r}_2, t)$. The field fluctuations from these pinholes propagates to another screen at distance z from the first. At a point $\mathbf{r}_s = (\mathbf{s}, z)$ the propagated fluctuations are superimposed. The light field at $\mathbf{r}_s$ is then given by

$$\phi(\mathbf{r}_s, t) = a_1\, \phi(\mathbf{r}_1, t - t_1) + a_2\, \phi(\mathbf{r}_2, t - t_2) \tag{2.80}$$

where a_j are constants[2] and $t_j = c/R_j$, $(j = 1, 2)$ is the flight time for light for either of the pinholes to $\mathbf{r}_s$. Due to the rapid fluctuations, the instantaneous intensity

$$I(\mathbf{r}_s, t) = \phi(\mathbf{r}_s, t)\phi^*(\mathbf{r}_s, t) \tag{2.81}$$

is not available as observable. Instead the expectation value

$$I(\mathbf{r}_s) = \langle I(\mathbf{r}_s, t)\rangle = \langle \phi(\mathbf{r}_s, t)\phi^*(\mathbf{r}_s, t)\rangle \tag{2.82}$$

[2] If the Huygen-Fresnel principle is considered, these constants are given by $a_j = -\mathrm{i}\mathrm{d}A_j/\bar{\lambda}R_j$ with $\mathrm{d}A$ being the pinhole aperture size and $\bar{\lambda}$ the mean wavelength of the (quasi-monochromatic) light. Here, we just assume that the pinholes are manufactured in a fashion that (2.80) holds true.

can be observed. The angle brackets denote the ensemble average or the time average if ergodicity can be assumed. As we considered only *stationary* light as illumination, the expectation value must be the same irrespective of time t (we implicitly assumed that by dropping the time dependence in $I(\mathbf{r}_s)$). Hence, only the relative flight time $t_1 - t_2$ matters and the expected intensity at point $\mathbf{r}_s$ on the second screen can be written as

$$I(\mathbf{r}_s) = |a_1|^2 I(\mathbf{r}_1) + |a_2|^2 I(\mathbf{r}_2) + 2\,\mathrm{Re}\left\{|a_1||a_2|\,\Gamma(\mathbf{r}_1, \mathbf{r}_2, t_1 - t_2)\right\}\,, \tag{2.83}$$

where

$$\Gamma(\mathbf{r}_1, \mathbf{r}_2, \tau) = \langle \phi^*(\mathbf{r}_1, t)\phi(\mathbf{r}_2, t+\tau)\rangle \tag{2.84}$$

denotes the cross-correlation of the light field at pinhole points $\mathbf{r}_1$ and $\mathbf{r}_2$. This second order correlation function is also called *mutual coherence function* and is the central element of second-order coherence theory. The mutual coherence is considered to be the simplest correlation available (Wolf, 1982) (for higher order correlations, see (Mandel and Wolf, 1965)). Although pinhole and screen distance control time delay and separation of the parts of the wave field here, the mutual coherence is defined irrespective of the point where that correlation can be made visible. It denotes an intrinsic characteristic of the light field. For example, $\mathbf{r}_1$ and $\mathbf{r}_2$ do not need to lie in the same plane.

The mutual coherence of the same point, $\Gamma(\mathbf{r}, \mathbf{r}, \tau)$, is commonly referred to as *self coherence function*

$$\tilde{\Gamma}(\mathbf{r}, \tau) = \Gamma(\mathbf{r}, \mathbf{r}, \tau) \tag{2.85}$$

and we note that the self coherence without time delay is the same as the obeservable intensity $I(\mathbf{r}) = \tilde{\Gamma}(\mathbf{r}, 0)$ of the optical field.

Intuitively, correlation (or statistical similarity) grows as the time delay τ or the spatial separation $|\mathbf{r}_1 - \mathbf{r}_2|$ approaches zero. As a consequence, the mutual coherence is bound by $\sqrt{I(\mathbf{r}_1)I(\mathbf{r}_2)}$ and may be normalized to

$$\gamma(\mathbf{r}_1, \mathbf{r}_2, \tau) = \frac{\Gamma(\mathbf{r}_1, \mathbf{r}_2, \tau)}{\sqrt{I(\mathbf{r}_1)I(\mathbf{r}_2)}} \tag{2.86}$$

which is called the *complex degree of coherence*. We observe that γ is bound by $0 \leq |\gamma| \leq 1$ and the fluctuating field is regarded (fully) *coherent* when $|\gamma| = 1$ for all $\mathbf{r}_1 \neq \mathbf{r}_2$ and $\tau \neq 0$ and *incoherent* when $|\gamma| = 0$ for the same cases. Apparently, most natural fields will have a degree of coherence somewhere in between 0 and 1 and these fields are considered to be *partially coherent*.

2.3.2. Coherence time and length

If we were to measure the degree of self coherence $\gamma(\mathbf{r},\mathbf{r},\tau)$ at a specific point, we would find that for sources that are not absolutely monochromatic, correlation would eventually vanish for $\tau \to \infty$. This observation leads to the so called *coherence time* τ_{coh}, which is the time after which the degree of coherence has fallen to zero. The coherence time is inversely proportional to the bandwidth $\Delta\nu$ of the spectrum of the light field at $\mathbf{r}$, i.e.

$$\tau_{\mathrm{coh}} \sim \frac{1}{\Delta\nu}\,, \tag{2.87}$$

which is to be expected, as for scalar waves the phasor's angular velocity depends on the wave's frequency. The coherence time may be understood as the time after which two waves with a frequency difference of $\Delta\nu$ have "dephased" (Pfeiffer et al., 2007). The proportionality factor depends on the type of spectrum. According to Goodman (1985), we find

$$\tau_{\mathrm{coh}} = 1/\Delta\nu \quad \text{for a rectangular line spectrum, and} \tag{2.88}$$

$$\tau_{\mathrm{coh}} = 0.664/\Delta\nu \quad \text{for a Gaussian line spectrum.} \tag{2.89}$$

Although the coherence time can be significantly longer than isolated fluctuations of the light field, it is most often too fast for any physical setup to detect. A constant that is easier to comprehend than the coherence time is the *(temporal) coherence length*. It is the distance, that light would travel during time interval τ_{coh},

$$\ell_{\mathrm{coh}} \equiv c\,\tau_{\mathrm{coh}} \sim \frac{c}{\Delta\nu} = \frac{\bar{\lambda}\bar{\nu}}{\Delta\nu} \approx \frac{\bar{\lambda}^2}{\Delta\lambda}\,, \tag{2.90}$$

where we assumed that the ratio of bandwidth to mean wavelength $\bar{\lambda}$ or mean frequency $\bar{\nu}$ is approximately the same. In this manner, we introduce the *relative* or *fractional bandwidth*

$$\Lambda = \frac{\Delta\lambda}{\bar{\lambda}} \approx \frac{\Delta\nu}{\bar{\nu}} \tag{2.91}$$

as dimensionless quantity of the temporal spectrum. When fluctuations propagate through the wave field, the correlation vanishes when the difference in path lengths exceed the coherence length $\ell_{\mathrm{coh}} = \lambda/\Lambda$. As can be seen in Fig. 2.2, path lengths differ prominently for off-axis interference of the diffracted waves. For two points at spacing D in the specimen plane $\mathbf{x}$, interference at scattering angle $\alpha \approx \lambda Q$ implies a path difference $D\sin\alpha$. For a correlation greater than zero that path difference should be less then the coherence length, i.e.

$$D\sin\alpha < \ell_{\mathrm{coh}}\,. \tag{2.92}$$

Eq.(2.92) can also be expressed in terms of the relative bandwidth

$$\Lambda \lesssim (DQ)^{-1} \tag{2.93}$$

which means that decoherence from temporal bandwidth becomes an issue when the space-bandwidth product is large.
As the time delay τ occurs along the direction of propagation, ℓ_{coh} is commonly called *longitudinal coherence length*. However the correlation between two points in the wave field may be reduced even though the time delay of their propagation paths to the point of interference is zero. The smallest distance between two points for which the degree of coherence vanishes is called *equal-time coherence length* $\ell_{\mathbf{x}}$,

$$\gamma(\mathbf{r}_1, \mathbf{r}_2, 0) = 0 \quad \text{for} \quad |\mathbf{r}_1 - \mathbf{r}_2| > \ell_{\mathbf{x}}\,. \tag{2.94}$$

Apparently these points are located on a sphere around the observation point, or in paraxial approximation, on a plane orthogonal to the optical axis. Hence, the term *transverse coherence length* is commonly used in forward scattering. We will later see in Sec. 2.3.8, that the extent of the source influences the transverse coherence.

2.3.3. Cross spectral density

While suited for statistical analysis, the mutual coherence is of limited practical use for experimental reality. The coherence time of light is too small for a measurement even for very small bandwidths and especially for high energy radiation such as X-rays. Additionally, sources and dispersion relations of matter are usually characterized over the energy or frequency of the interacting light. Hence, a time-independent correlation function would be favourable.

For a statistically stationary light field, let $\phi(\mathbf{r}, t)$ be again one (complex) sample function that is also a solution to the Helmholtz equations. For two points $\mathbf{r}_1$ and $\mathbf{r}_2$ we assume that the time dependent signal may be composed from a Fourier integral of frequencies

$$\phi(\mathbf{r}_j, t) = \mathcal{F}_\nu \left\{\psi(\mathbf{r}_j, \nu)\right\} = \int_{-\infty}^{\infty} \psi(\mathbf{r}_j, \nu) e^{-2\pi \mathrm{i}\, \nu t} \mathrm{d}\nu \qquad (j = 1, 2)\,. \tag{2.95}$$

Fourier inverting eq.(2.95) yields

$$\psi(\mathbf{r}_j, \nu) = \mathcal{F}^*_t \left\{\phi(\mathbf{r}_j, t)\right\} = \int_{-\infty}^{\infty} \phi(\mathbf{r}_j, t) e^{2\pi \mathrm{i}\, \nu t} \mathrm{d}t \qquad (j = 1, 2)\,. \tag{2.96}$$

The integral in eq.(2.96) does, in general, not exist, as sample functions of a stationary signal never vanish for $t \to \pm\infty$. The integral can be understood in a distributional sense in which case the integral boundaries are to be replaced by a finite T and $-T$. Then, the final integration to infinite boundaries ($T \to \infty$) is carried out last.[3]
With that in mind, we express the second-order space-frequency correlation between two points and frequencies as

$$\langle \psi^*(\mathbf{r}_1,\nu)\psi(\mathbf{r}_2,\bar{\nu})\rangle = \left\langle \iint_{-\infty}^{\infty} \phi^*(\mathbf{r}_1,t)\phi(\mathbf{r}_2,\bar{t})\, e^{2\pi \mathrm{i}\,(\bar{\nu}\bar{t}-\nu t)} \mathrm{d}t\mathrm{d}\bar{t} \right\rangle \tag{2.97}$$

$$= \iint_{-\infty}^{\infty} \langle \phi^*(\mathbf{r}_1,t)\phi(\mathbf{r}_2,t+\tau)\rangle\, e^{2\pi \mathrm{i}\,(\bar{\nu}-\nu)t} e^{2\pi \mathrm{i}\,\bar{\nu}\tau)} \mathrm{d}t\mathrm{d}\tau \tag{2.98}$$

$$= \int_{-\infty}^{\infty} \Gamma(\mathbf{r}_1,\mathbf{r}_2,\tau) e^{2\pi \mathrm{i}\,\bar{\nu}\tau} \mathrm{d}\tau \int_{-\infty}^{\infty} e^{2\pi \mathrm{i}\,(\bar{\nu}-\nu)t} \mathrm{d}t \tag{2.99}$$

$$= \mathcal{S}(\mathbf{r}_1,\mathbf{r}_2,\bar{\nu})\delta(\bar{\nu}-\nu)\,. \tag{2.100}$$

Due to the delta function, any two frequency components of the space-frequency correlation of a stationary light field are *uncorrelated*. This observation is quite important. It implies, that the spatial coherence properties of the light field may be treated independently for each frequency (or wavelength). Furthermore, in eq.(2.100) we introduced the *cross-spectral density (CSD)* or *cross power spectrum*

$$\mathcal{S}(\mathbf{r}_1,\mathbf{r}_2,\nu) = \mathcal{F}^*_\tau\left\{\Gamma(\mathbf{r}_1,\mathbf{r}_2,\tau)\right\} = \int_{-\infty}^{\infty} \Gamma(\mathbf{r}_1,\mathbf{r}_2,\tau) e^{2\pi \mathrm{i}\,\nu\tau} \mathrm{d}\tau\,, \tag{2.101}$$

as the Fourier transform of the mutual coherence with respect to τ. Due to the decline of the mutual intensity with $\tau \to \pm\infty$, the CSD can exist while the Fourier transform of a sample function may not.

If the light field is probed at a single point $\mathbf{r}$, the expression

$$\tilde{\mathcal{S}}(\mathbf{r},\nu) = \mathcal{S}(\mathbf{r},\mathbf{r},\nu) = \mathcal{F}^*_\tau\left\{\Gamma(\mathbf{r},\mathbf{r},\tau)\right\} = \int_{-\infty}^{\infty} \Gamma(\mathbf{r},\mathbf{r},\tau) e^{2\pi \mathrm{i}\,\nu\tau} \mathrm{d}\tau\,, \tag{2.102}$$

is called *spectral density* or *power spectrum*. With inverting eq.(2.101), we obtain

$$\Gamma(\mathbf{r}_1,\mathbf{r}_2,\tau) = \mathcal{F}_\nu\left\{\mathcal{S}(\mathbf{r}_1,\mathbf{r}_2,\nu)\right\} = \int_{-\infty}^{\infty} \mathcal{S}(\mathbf{r}_1,\mathbf{r}_2,\nu) e^{-2\pi \mathrm{i}\,\nu\tau} \mathrm{d}\nu\,, \tag{2.103}$$

[3] Similarly, we could work with an analytic expansion of the signal $\phi(\mathbf{r}_j,t)$, allow for a complex time and choose the appropriate integration path in the complex plane.

which we can use to calculate the observable intensity,

$$I(\mathbf{r}) = \Gamma(\mathbf{r}, \mathbf{r}, 0) = \int_{-\infty}^{\infty} \mathcal{S}(\mathbf{r}, \mathbf{r}, \nu) \mathrm{d}\nu \,, \tag{2.104}$$

as the integral over the spectral density.

2.3.4. Mode representation of the spectral density

For equations (2.101) to (2.104) to be valid, the mutual coherence function must be square integrable. This is the case for primary or secondary statistical sources under the prerequisite that, within a finite spatial domain $\mathcal{V} \subset \mathbb{R}^3$, the mutual coherence

- is bound by function of τ that falls fairly rapidly for $\tau \to \pm\infty$ and
- is a continuous function of $\mathbf{r}_1$ and $\mathbf{r}_2$.

The first condition ensures that the Fourier integrals are allowed, while the second ensures, that the cross spectral density $\mathcal{S}$ is continuous itself with respect to $\mathbf{r}_1$ and $\mathbf{r}_2$. Most physical sources or fields of interest fulfil these condition as they are at least narrowband.[4] Wolf (1982) further assumes that the fluctuating field is localised and bound by $\mathcal{V}$. In paraxial geometry we can only assert that the extent of the wave fields is finite on planes orthogonal to the optical axis. Hence, we continue here with selecting the source plane $z = 0$,

$$\mathcal{S}(\mathbf{x}_1, \mathbf{x}_2, \nu) \equiv \mathcal{S}((\mathbf{x}_1, 0), (\mathbf{x}_2, 0), \nu) \tag{2.105}$$

and assume that the field fits in an area $\mathcal{D}$. Like Wolf (1982), we observe that the cross-spectral density becomes a *Hilbert-Schmidt* kernel as

$$\iint_{\mathcal{D}} |\mathcal{S}(\mathbf{x}_1, \mathbf{x}_2, \nu)|^2 \, \mathrm{d}\mathbf{x}_1 \mathrm{d}\mathbf{x}_2 < \infty \,. \tag{2.106}$$

The kernel is also *hermitian* and *non-negative definite* due to

$$\mathcal{S}(\mathbf{x}_1, \mathbf{x}_2, \nu) = \mathcal{S}^*(\mathbf{x}_2, \mathbf{x}_1, \nu) \tag{2.107}$$

and

$$\iint_{\mathcal{D}} f^*(\mathbf{x}_1) \mathcal{S}(\mathbf{x}_1, \mathbf{x}_2, \nu) f(\mathbf{x}_2) \, \mathrm{d}\mathbf{x}_1 \mathrm{d}\mathbf{x}_2 \geq 0 \,, \tag{2.108}$$

[4] We explain the implications for quasi-monochromatic sources at the end of the chapter.

where f is any square-integrable function. According to Mercers theorem, a fundamental theorem in functional analysis, a continuous, non-negative definite, hermitian Hilbert-Schmidt kernel may be expressed as a uniformly convergent series,

$$\mathcal{S}(\mathbf{x}_1, \mathbf{x}_2, \nu) = \sum_n \chi_n(\nu)\psi_n^*(\mathbf{x}_1, \nu)\psi_n(\mathbf{x}_2, \nu)\,. \tag{2.109}$$

The functions $\psi_n(\mathbf{x}, \nu)$ are eigenfunctions and χ_n to the Fredholm integral equation, i.e.

$$\int_D \mathcal{S}(\mathbf{x}_1, \mathbf{x}_2, \nu)\psi_n(\mathbf{x}_1, \nu)\,\mathrm{d}\mathbf{x}_1 = \chi_n(\nu)\psi_n(\mathbf{x}_2, \nu)\,. \tag{2.110}$$

Due to the hermiticity and non-negative definiteness of $\mathcal{S}$, all eigenvalues are real and positive,

$$\chi_n(\nu) \geq 0\,, \tag{2.111}$$

and there exists at least one non-zero eigenvalue. Furthermore, the eigenfunctions may be orthonormal or, in case of eigenvalue degeneracy, orthonormised by a Gram-Schmidt procedure.

$$\int_D \psi_n^*(\mathbf{x}, \nu)\psi_m(\mathbf{x}, \nu)\,\mathrm{d}\mathbf{x} = \delta_{nm}(\nu)\,. \tag{2.112}$$

Consistent with Thibault and Menzel (2013), we will call these eigenfunctions of the cross-spectral density as *modes* or *states*. For partially coherent light fields, the number of non-zero eigenvalues, called the *rank* k, is either finite or the eigenvalues fall to zero fairly rapidly for $n > \mathsf{k}$.

In Wolf (2007), the modes ψ_m are three-dimensional fields, completely spatial coherent, and also solutions to the time-independent Helmholtz equation. Here, we limited the eigenvalue decomposition to a single plane $z = 0$ which means we can not make the same assertions. Nevertheless, we can use the angular spectrum propagator (2.25) to build a three-dimensional field from the mode ψ_n at the source plane,

$$\psi_n(\mathbf{r}, \nu) = \psi_n((\mathbf{s}, z), \nu) \equiv \underset{\mathbf{x},\mathbf{s}}{\mathcal{P}}(\nu, z)\left\{\psi_n(\mathbf{x})\right\}\,. \tag{2.113}$$

By design, these fields are now solutions to the time-independent Helmholtz equation and spatially coherent. Conveniently, the orthonormality is also preserved. For any observation plane $z = \mathrm{z}$ we find

$$\int \psi_n^*((\mathbf{s}, \mathrm{z}), \nu)\psi_m((\mathbf{s}, \mathrm{z}), \nu)\,\mathrm{d}\mathbf{s} = \delta_{nm}(\nu)\,, \tag{2.114}$$

after some Fourier transforms and a final application of the source plane orthonormality (2.112).

Keeping in mind, that orthogonality only applies along transverse planes, the modes $\psi_m(\mathbf{r}, \nu)$ span the space of monochromatic representations of the fluctuating field,

$$\psi(\mathbf{r}, \nu) = \sum_n \alpha_n(\nu)\psi_n(\mathbf{r}, \nu) \qquad \text{with} \qquad \langle|\alpha_n(\nu)|^2\rangle = \chi_n(\nu)\,. \tag{2.115}$$

The unit of the eigenvalue $\chi_n(\nu)$ is transmitted power per frequency which, in turn, is an energy. Hence, $\chi_n(\nu)/(h\nu)$ is interpreted as the number of photons occupying the n-th mode.

2.3.5. Mutual intensity

The mode decomposition also applies if the light field is (quasi)-monochromatic but partial spatial coherent. In fact, in many paraxial diffraction setups, the time delay τ introduced by path length differences may be small compared to the bandwidth,

$$|\tau| << \frac{1}{\Delta\nu}\,. \tag{2.116}$$

This is the case, when light phenomena are investigated close to the optical axis or if the setup provides (rotational) symmetry around the optical axis or when the spectrum is narrow. We approximate the mutual coherence as follows

$$\Gamma(\mathbf{r}_1, \mathbf{r}_2, \tau) \approx \Gamma(\mathbf{r}_1, \mathbf{r}_2, 0)e^{-2\pi\mathrm{i}\,\bar{\nu}\tau} \equiv J(\mathbf{r}_1, \mathbf{r}_2)e^{-2\pi\mathrm{i}\,\bar{\nu}\tau} \tag{2.117}$$

$$\gamma(\mathbf{r}_1, \mathbf{r}_2, \tau) \approx \gamma(\mathbf{r}_1, \mathbf{r}_2, 0)e^{-2\pi\mathrm{i}\,\bar{\nu}\tau} \equiv j(\mathbf{r}_1, \mathbf{r}_2)e^{-2\pi\mathrm{i}\,\bar{\nu}\tau} \tag{2.118}$$

where $\bar{\nu}$ is the average frequency of the narrow-band light field. The equations above implicitly define the so-called *mutual intensity* $J(\mathbf{r}_1, \mathbf{r}_2)$ (also called *equal-time coherence function* (Wolf, 2007)) and *equal-time complex degree of coherence* $j(\mathbf{r}_1, \mathbf{r}_2)$. As conditions (2.106) to (2.108) also hold for $J(\mathbf{r}_1, \mathbf{r}_2)$, the same mode decomposition applies to the mutual intensity as it does to the cross-spectral density. Frequency dependent functions can be replaced by their spectral average,

$$\psi_n(\mathbf{r}) \equiv \psi_n(\mathbf{r}, \bar{\nu}) \tag{2.119}$$

$$\chi_n \equiv \int \chi_n(\nu)\mathrm{d}\nu \approx \Delta\nu\,\chi_n(\bar{\nu}) \tag{2.120}$$

which means that the unit of eigenvalue χ_n now is transmitted power. Henceforth, we will denote the mode with the highest power as *main mode* and label a mode's contribution to the total power $\chi_n/(\sum_j \chi_j)$ as *relative power* or *relative occupancy*.

2.3.6. Propagation of mutual intensity or cross-spectral density

Let now $\Psi(\mathbf{r}_1, \mathbf{r}_2)$ now be either the cross-spectral density to a single frequency component $\bar{\nu}$[5] or the mutual intensity, possibly after an incident partially coherent field has interacted with a partially coherent scatterer. It can be shown that Ψ is a solution of the time-independent Helmholtz equation for each of its positional arguments, i.e.

$$\nabla^2_{\mathbf{r}_1} \Psi(\mathbf{r}_1, \mathbf{r}_2) + k^2 \Psi(\mathbf{r}_1, \mathbf{r}_2) = 0\,, \tag{2.121}$$

and

$$\nabla^2_{\mathbf{r}_2} \Psi(\mathbf{r}_1, \mathbf{r}_2) + k^2 \Psi(\mathbf{r}_1, \mathbf{r}_2) = 0\,, \tag{2.122}$$

where

$$k = \frac{2\pi}{\bar{\lambda}} = \frac{2\pi\bar{\nu}}{c_0 n}\,. \tag{2.123}$$

Equations (2.121) and (2.122) indicate that Ψ can be propagated individually for each of its positional arguments. Similar to the coherent case, we assume that Ψ is known in the source plane orthogonal to the optical axes at $\mathbf{r} = (\mathbf{x}, 0)$,

$$\Psi^0(\mathbf{x}_1, \mathbf{x}_2) \equiv \Psi((\mathbf{x}_1, 0), (\mathbf{x}_2, 0))\,, \tag{2.124}$$

which we will call *exit state correlation*. The exit state at detector plane $\mathbf{r} = (\mathbf{s}, \mathrm{z})$ is given by applying the paraxial propagator (2.34) twice:

$$\Psi^{\mathrm{z}}(\mathbf{s}_1, \mathbf{s}_2) \equiv \Psi((\mathbf{s}_1, \mathrm{z}), (\mathbf{s}_2, \mathrm{z})) \tag{2.125}$$

$$= \hat{\mathcal{P}}_{\mathbf{x}_1} \left\{ \mathcal{P}_{\mathbf{x}_2} \left\{ \Psi^0(\mathbf{x}_1, \mathbf{x}_2) \right\} \right\} (\mathbf{s}_1, \mathbf{s}_2)\,, \tag{2.126}$$

where $\hat{\mathcal{P}}_{\mathbf{x}_1} \mathcal{P}_{\mathbf{x}_2}$ is the paraxial propagator for the cross-spectral density. The propagator with respect to $\mathbf{x}_1$ is reversed (complex conjugated) here in order to preserve the (hermitian) invariance towards complex conjugation with coordinate exchange. In explicit form, eq. (2.126) is written as

$$\Psi^{\mathrm{z}}(\mathbf{s}_1, \mathbf{s}_2) = \frac{\bar{\nu}^2}{c^2 \mathrm{z}^2} \iint \Psi^0(\mathbf{x}_1, \mathbf{x}_2) \exp\left[\frac{\pi \mathrm{i} \bar{\nu} (\mathbf{s}_2 - \mathbf{x}_2)^2}{c\mathrm{z}}\right] \exp\left[\frac{-\pi \mathrm{i} \bar{\nu} (\mathbf{x}_1 - \mathbf{s}_1)^2}{c\mathrm{z}}\right] \mathrm{d}\mathbf{x}_1 \mathrm{d}\mathbf{x}_2\,, \tag{2.127}$$

We omit a proof here but the interested reader is advised to look up how the propagation is carried out in Wolf (2007) or Born et al. (1999) where the same result

[5] Here, this is an arbitrary choice for the cross-spectral density, but we use it in order for the upcoming equations to be also valid for the mutual intensity

is achieved using the Huygen-Fresnel principle and spherical waves emanating from disturbances at $\mathbf{x}_1$ and $\mathbf{x}_2$. With inserting the mode decomposition into (2.127) we realise that the propagated cross-spectral density is composed of the propagated modes (2.113) of the exit state correlation,

$$\Psi^{\mathrm{z}}(\mathbf{s}_1, \mathbf{s}_2) = \sum_n \chi_n \psi_n^*(\mathbf{s}_1, \mathrm{z}) \psi_n(\mathbf{s}_2, \mathrm{z}). \tag{2.128}$$

2.3.7. Interaction with linear or random media

The exit state correlation Ψ fulfils all conditions ((2.106) to (2.108)) to guarantee a mode decomposition,

$$\Psi(\mathbf{x}_1, \mathbf{x}_2) = \sum_k \chi_n \, \psi_n^*(\mathbf{x}_1) \psi_n(\mathbf{x}_2) \tag{2.129}$$

where ψ_k are the *exit states* or *exit modes*. For convenience, we temporarily dropped the dispersion relation. In absence of any diffracting sample, eq.(2.129) also holds for the mutual intensity of illumination:

$$\Psi(\mathbf{x}_1, \mathbf{x}_2) = P(\mathbf{x}_1, \mathbf{x}_2) = \sum_m \chi_m \, p_m^*(\mathbf{x}_1) p_m(\mathbf{x}_2) \, . \tag{2.130}$$

The question is now what happens when partially coherent light propagates through a medium. Both Wolf (2007) and Goodman (1985) address this issue but the former only analyses the influence on the spectrum of the scattered wave while the latter focusses on the point-spread from atmospheric air scattering. Both approaches are not entirely useful for us. Here, we are mainly interested if there is some form of mutual intensity of the object transmission, and if a mode decomposition applies to it.

For a static object, we assume that the projection approximation from section 2.1.3 is transferable to the probe modes. In that case we may identify the exit states as

$$\psi_m(\mathbf{x}) = p_m(\mathbf{x}) \, o(\mathbf{x}) \tag{2.131}$$

Similarly, from a statistical point of view, static modulations can be factored out from the time average of the mutual coherence function. Effectively, the exit state correlation after a linear and static transmission sample o is then written as

$$\Psi(\mathbf{x}_1, \mathbf{x}_2) = P(\mathbf{x}_1, \mathbf{x}_2) \, o^*(\mathbf{x}_1) o(\mathbf{x}_2) \, . \tag{2.132}$$

Ideally, we would like to identify the right factor in eq.(2.132) as the coherent limit of a more general object correlation $O(\mathbf{x}_1, \mathbf{x}_2)$ such that

$$\Psi(\mathbf{x}_1, \mathbf{x}_2) = P(\mathbf{x}_1, \mathbf{x}_2) \, O(\mathbf{x}_1, \mathbf{x}_2) \, , \tag{2.133}$$

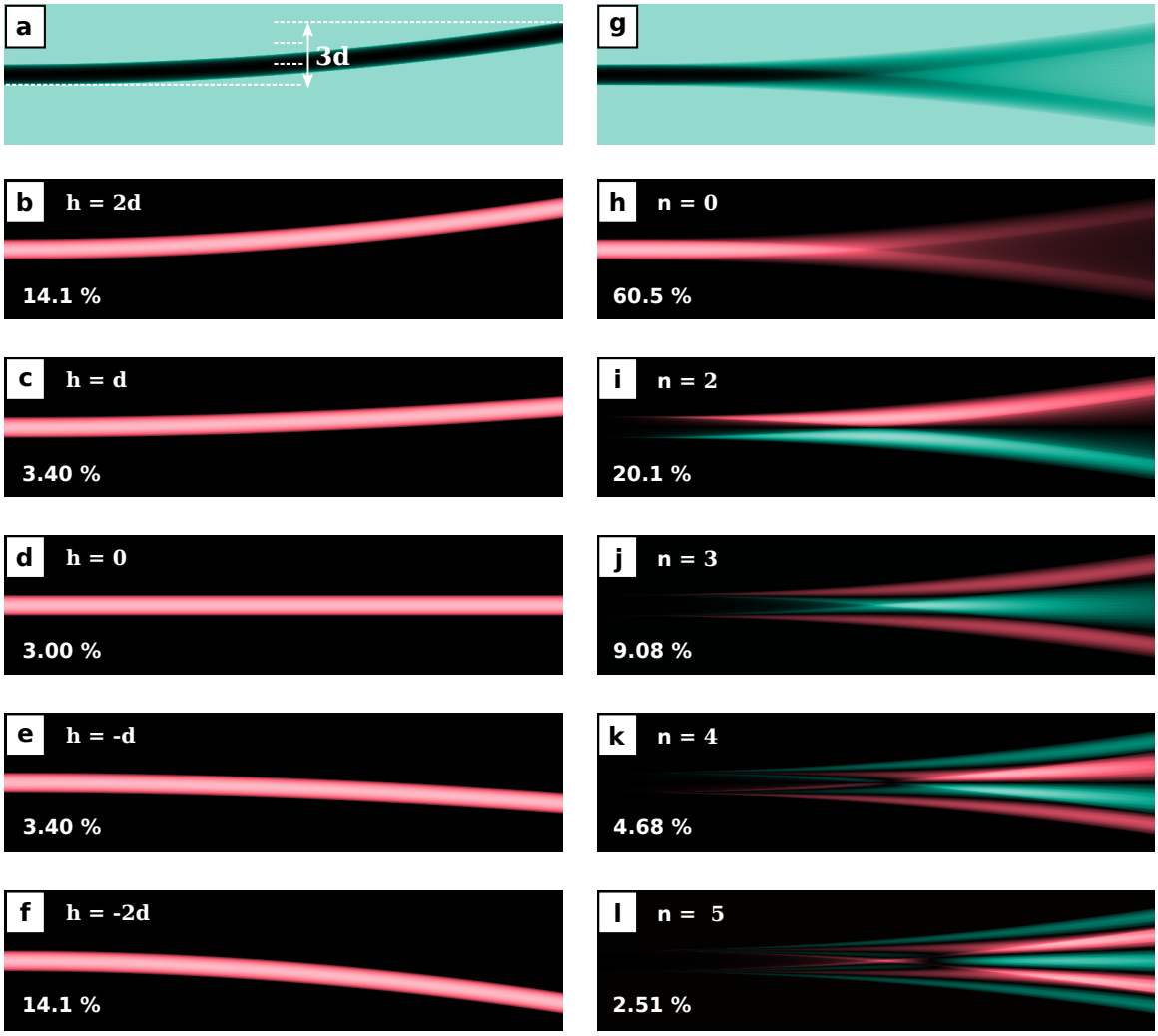
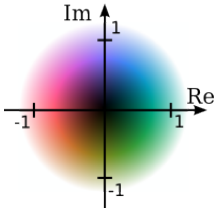


Figure 2.3. – Mode decomposition analysis of an oscillating wire fixed at one end. The tip oscillates sinusoidally, $h(t) = 2d\sin(2\pi\, t/T)$. **a** Wire at reversal point of maximum amplitude $h = 2d$ which is twice its diameter in this case. The oscillation was binned in amplitude into 21 states. **b-f** Oscillation states corresponding to full, half and zero tip displacement (there are four more states in between). The constant background transmission of air was subtracted to obtain the scattering density. The percentage denotes the relative amount of time spent in a similar state as the one shown, i.e. in the same 5% amplitude bracket. **g-l** display the respective mode decomposition of the same oscillation. **g** is the main mode, $o_0(\mathbf{x}) = \langle o(\mathbf{x})\rangle = \langle \tilde{o}(\mathbf{x})\rangle + 1$, of which the free air transmission was subtracted in **h** to compare it with the higher modes shown in **i-l**. The percentage denotes the ratio of the eigenvalue of a mode to the sum of all eigenvalues. All fields in the panels are normalized to their maximum value and take values within the unit circle in the complex plane.



where

$$O(\mathbf{x}_1, \mathbf{x}_2) = \sum_n \kappa_n \, o_n^*(\mathbf{x}_1) o_n(\mathbf{x}_2) \tag{2.134}$$

is the corresponding mode decomposition for the object correlation. The modes should be eigenvectors and the decomposition should be characteristic for the sample.

The problem with eq.(2.133) is that the mode decomposition may not be unique but instead depend on the actual shape of the integration domain $\mathcal{D}$ as the object transmission usually is not inherently confined by a compact support.However the actual active volume which is causing diffraction,

$$\tilde{o}(\mathbf{r}, t) = o(\mathbf{r}, t) - 1 \,, \tag{2.135}$$

may be confined to a certain cross section $\mathcal{D}_o$ at the source plane $z = 0$. We will denote $\tilde{o}$ as *scattering density*. Here, we assume a transmission of unity outside the active volume, but any other constant transmission value is also valid. An explicit calculation of the mutual intensity of the exit state yields

$$\begin{aligned}
\Psi(\mathbf{x}_1, \mathbf{x}_2) &= \langle p^*(\mathbf{x}_1, t)(\tilde{o}^*(\mathbf{x}_1, t) + 1) p(\mathbf{x}_2, t)(\tilde{o}(\mathbf{x}_2, t) + 1) \rangle_t && (2.136)\\
&= \langle p^*(\mathbf{x}_1, t) p(\mathbf{x}_2, t) \rangle_t \left[\langle (\tilde{o}^*(\mathbf{x}_1, t) \tilde{o}(\mathbf{x}_2, t) \rangle_t + \langle \tilde{o}(\mathbf{x}_2, t) \rangle_t + \langle \tilde{o}^*(\mathbf{x}_1, t) \rangle_t + 1 \right] && (2.137)\\
&= P(\mathbf{x}_1, \mathbf{x}_2) \left[\tilde{O}(\mathbf{x}_1, \mathbf{x}_2) + \langle \tilde{o}(\mathbf{x}_2) \rangle + \langle \tilde{o}^*(\mathbf{x}_1) \rangle + 1 \right] , && (2.138)
\end{aligned}$$

where we assumed the probe and object fluctuations to be statistically independent.

$$\Psi_{\text{inh}}(\mathbf{x}_1, \mathbf{x}_2) = P(\mathbf{x}_1, \mathbf{x}_2) \tilde{O}(\mathbf{x}_1, \mathbf{x}_2) \tag{2.139}$$

is the mutual intensity of the diffracted wave at the source plane. Due to the confinement of $\tilde{O}$ within a finite domain $\mathcal{D}_o$ we can be sure that an eigenvector decomposition $\{\tilde{o}_n(\mathbf{x})\}$ is independent of the integration domain if $\mathcal{D} \supset \mathcal{D}_o$ and thus be a characteristic of the sample.

Eventually, we compare eq.(2.138) with eq.(2.134) and eq.(2.133) to realize that the decomposition of O can be considered as the extension of $\tilde{O}$ with one special mode,

$$o_0(\mathbf{x}) = \langle \tilde{o}(\mathbf{x}) \rangle + 1 = \langle o(\mathbf{x}) \rangle \,, \tag{2.140}$$

which is a time or ensemble average of all possible object transmission states. All other modes of O are the same as $\tilde{O}$, i.e $o_{j \neq 0} = \tilde{o}_{j \neq 0}$, which implies that the average of the active volume is an eigenvector of $\tilde{O}$,

$$\tilde{o}_0(\mathbf{x}) = \langle \tilde{o}(\mathbf{x}) \rangle \,. \tag{2.141}$$

An exemplary mode decomposition of O is depicted in Fig. 2.3. Here, the simulated specimen is a wire fixed at one end and bend by two times its diameter. After release, the sinusoidal oscillation of the wire results in statistical stationary transmission.

Eventually, under the assumption that a mode decomposition is possible for transmission and illumination, we write

$$\Psi(\mathbf{x}_1, \mathbf{x}_2, \nu) = \sum_{mn} \chi_m(\nu)\kappa_n(\nu)\, p_m^*(\mathbf{x}_1, \nu) p_m(\mathbf{x}_2, \nu)\, o_n^*(\mathbf{x}_1, \nu) o_n(\mathbf{x}_2, \nu) \tag{2.142}$$

for broadband partially coherent source and a dispersive transmission of a possible fluctuating object. With eq. (2.126) and eq. (2.104) we finally calculate the intensity in the detector plane as

$$I(\mathbf{s}) = \int_{-\infty}^{\infty} \Psi^z(\mathbf{s}, \mathbf{s}, \nu)\mathrm{d}\nu = \sum_{mn} \int_{-\infty}^{\infty} \left| \underset{\mathbf{x},\mathbf{s}}{\mathcal{P}}(\nu) \left\{ p_m(\mathbf{x}, \nu)\, o_n(\mathbf{x}, \nu) \right\} \right|^2 \mathrm{d}\nu\,, \tag{2.143}$$

which is the expression we were looking for in the first place. For the sake of a compacter form of eq. (2.143), we rescaled probe and object modes by the root of their respective eigenvalues.

2.3.8. Ambiguity function and van Cittert-Zernike theorem

Let $\Psi(\mathbf{x}, \mathbf{x})$ be either the mutual intensity or the cross-spectral density for frequency $\bar{\nu}$. We define the *ambiguity function* of Ψ in the observation plane as

$$\mathcal{A}_\Psi^z(\mathbf{v}, \mathbf{y}) = \int \Psi^z\left(\mathbf{s} - \frac{\mathbf{y}}{2}, \mathbf{s} + \frac{\mathbf{y}}{2}\right) e^{-2\pi \mathrm{i}\, \mathbf{s}\mathbf{v}} \mathrm{d}\mathbf{s} = \underset{\mathbf{s},\mathbf{v}}{\mathcal{F}} \left\{ \Psi^z\left(\mathbf{s} - \frac{\mathbf{y}}{2}, \mathbf{s} + \frac{\mathbf{y}}{2}\right) \right\}, \tag{2.144}$$

and in the source plane as

$$\mathcal{A}_\Psi^0(\mathbf{v}, \mathbf{y}) = \int \Psi^0\left(\mathbf{x} - \frac{\mathbf{y}}{2}, \mathbf{x} + \frac{\mathbf{y}}{2}\right) e^{-2\pi \mathrm{i}\, \mathbf{x}\mathbf{v}} \mathrm{d}\mathbf{x} = \underset{\mathbf{x},\mathbf{v}}{\mathcal{F}} \left\{ \Psi^0\left(\mathbf{x} - \frac{\mathbf{y}}{2}, \mathbf{x} + \frac{\mathbf{y}}{2}\right) \right\}. \tag{2.145}$$

In the case of paraxial propagation, we find the following identity

$$\mathcal{A}_\Psi^z(\mathbf{v}, \mathbf{y}) = \mathcal{A}_\Psi^0(\mathbf{v}, \mathbf{y} + \bar{\lambda} z \mathbf{v}) \tag{2.146}$$

meaning that a propagation over distance z or a shearing operation of the ambiguity function at the source plane is the same. For the case $\mathbf{y} = 0$, the ambiguity function corresponds to the Fourier transform of the intensity,

$$\mathcal{F} I^z(\mathbf{v}) = \underset{\mathbf{s},\mathbf{v}}{\mathcal{F}} \left\{ I^z(\mathbf{s}) \right\} = \mathcal{A}_\Psi^z(\mathbf{v}, 0) = \mathcal{A}_\Psi^0(\mathbf{v}, \bar{\lambda} z \mathbf{v})\,. \tag{2.147}$$

This spatial frequency spectrum is found as slice in four-dimensional space one the surface $(\mathbf{v}, \bar{\lambda} z \mathbf{v})$.

If we assume, for example, that the mutual intensity of the probe originates from an incoherently illuminated aperture $a(\mathbf{x})$ at distance $z = -l$, the probe ambiguity at object plane takes the form

$$\mathcal{A}_P^0(\mathbf{v}, \mathbf{y}) = \mathcal{A}_P^{-l}(\mathbf{v}, \mathbf{y} + \bar{\lambda} l \mathbf{v}) = \int P^{-l}\left(\mathbf{x} - \frac{\mathbf{y} + \bar{\lambda} l \mathbf{v}}{2}, \mathbf{x} + \frac{\mathbf{y} + \bar{\lambda} l \mathbf{v}}{2}\right) e^{-2\pi \mathrm{i}\, \mathbf{x}\mathbf{v}} \mathrm{d}\mathbf{x} \tag{2.148}$$

$$= \int a\left(\mathbf{x} - \frac{\mathbf{y} + \bar{\lambda} l \mathbf{v}}{2}\right) \delta(\mathbf{y} + \bar{\lambda} l \mathbf{v}) e^{-2\pi \mathrm{i}\, \mathbf{x}\mathbf{v}} \mathrm{d}\mathbf{x}\,. \tag{2.149}$$

Inverting (2.144) and inserting (2.149) yields the mutual intensity of the probe,

$$P^0(\mathbf{x}_1, \mathbf{x}_2) = \int \mathcal{A}_P^0(\mathbf{v}, \mathbf{x}_2 - \mathbf{x}_1) e^{\pi \mathrm{i}\, (\mathbf{x}_2 + \mathbf{x}_1)\mathbf{v}} \mathrm{d}\mathbf{x} \tag{2.150}$$

$$= \frac{1}{\bar{\lambda} l} \exp\left[\frac{\pi \mathrm{i} \mathbf{x}_1^2}{\lambda l}\right] \exp\left[\frac{-\pi \mathrm{i} \mathbf{x}_2^2}{\lambda l}\right] \mathcal{F}a\left(\frac{\mathbf{x}_1 - \mathbf{x}_2}{\bar{\lambda} l}\right), \tag{2.151}$$

where we can identify the Fourier transform of intensity in the aperture, i.e. the last factor in eq.(2.151), as the complex degree of coherence $\gamma(\mathbf{x}_1, \mathbf{x}_2)$. Regardless of the actual shape of a, we can choose l large enough such that $\gamma \approx 1$ for any combination of $\mathbf{x}_1$ and $\mathbf{x}_2$. Eq.(2.151) is known as the *van Cittert-Zernike* theorem (Born et al., 1999) and means that coherent wave fields can be extracted from any (physical) source given that the propagation distance is long enough. This principle is used at synchrotrons to allow for coherent imaging with hard X-rays. In that case the propagation distance is fixed but the upstream aperture can be opened and closed to fine tune the coherence to the requirements of the experiment. We will see another practical application of the van Cittert-Zernike theorem later in this thesis (sec. 5.4)

Having the complex degree of coherence invariant to translation is a special case, but one of particular importance due to the van Cittert-Zernike theorem. We choose $\gamma(\mathbf{x}_1, \mathbf{x}_2) = \tilde{\gamma}(\mathbf{x}_1 - \mathbf{x}_2)$ (which implies $\tilde{\gamma}(-\mathbf{x}) = \tilde{\gamma}^*(\mathbf{x})$) investigate a bit further and write

$$\Psi^0(\mathbf{x}_1, \mathbf{x}_2) = p^*(\mathbf{x}_1) p(\mathbf{x}_2)\, \tilde{\gamma}(\mathbf{x}_1 - \mathbf{x}_2)\, o^*(\mathbf{x}_1) o(\mathbf{x}_2) \tag{2.152}$$

for the partially coherent wave field behind the object, which implies

$$\Psi^0_{\mathrm{coh}}(\mathbf{x}_1, \mathbf{x}_2) = p^*(\mathbf{x}_1) p(\mathbf{x}_2)\, o^*(\mathbf{x}_1) o(\mathbf{x}_2) \tag{2.153}$$

for the completely coherent case. Using (2.147), the spectrum of the intensity in the detection plane is now given by

$$\mathcal{F}I^z(\mathbf{v}) = \mathcal{A}(\mathbf{v}, \bar{\lambda}z\mathbf{v}) = \underset{\mathbf{x},\mathbf{v}}{\mathcal{F}} \left\{ \Psi^0 \left(\mathbf{x} - \frac{\bar{\lambda}z\mathbf{v}}{2}, \mathbf{x} + \frac{\bar{\lambda}z\mathbf{v}}{2} \right) \right\} \tag{2.154}$$

$$= \tilde{\gamma}(-\bar{\lambda}z\mathbf{v}) \underset{\mathbf{x},\mathbf{v}}{\mathcal{F}} \left\{ \Psi^0_{\mathrm{coh}} \left(\mathbf{x} - \frac{\bar{\lambda}z\mathbf{v}}{2}, \mathbf{x} + \frac{\bar{\lambda}z\mathbf{v}}{2} \right) \right\}$$

$$= \tilde{\gamma}^*(\bar{\lambda}z\mathbf{v})\, \mathcal{A}_{\mathrm{coh}}(\mathbf{v}, \bar{\lambda}z\mathbf{v})$$

$$= \tilde{\gamma}^*(\bar{\lambda}z\mathbf{v})\, \mathcal{F}I^z_{\mathrm{coh}}(\mathbf{v})\,. \tag{2.155}$$

When Fourier transforming eq.(2.155), we obtain

$$I^z(\mathbf{s}) = \frac{1}{\bar{\lambda}z}\, \mathcal{F}^* \tilde{\gamma}^* \left(\frac{\mathbf{s}}{\bar{\lambda}z} \right) \otimes_{\mathbf{s}} I^z_{\mathrm{coh}}(\mathbf{s})\,, \tag{2.156}$$

a *convolution* of the Fourier inverse of the complex degree of coherence with a diffraction image taken under complete spatial coherence. The visible effect on diffraction image is a blurring of the speckle images and takes on the same shape as a *point spread function, (PSF)* for general imaging systems. A PSF may be used to describe non-ideal detectors, where one detection event in the sensitive screen of the detection plane spreads over multiple detector pixels. We would like to point out here that it does not at all matter if the translational invariant $\tilde{\gamma}$ originates from probe or object fluctuations.

2.4. Ptychography

In X-ray microscopy (or light microscopy in general) we are interested in retrieving a high resolution image of the object. The principal idea of conventional (imaging) microscopy is to collect all possible diffracted information up to a maximum diffraction angle Q. If we can reconstruct the phase or if we have a phasing element such as a lens (visible light) or a zone plate (X-ray) available, we can recover an image at a resolution proportional to Q or ultimately to $\sim 2/\lambda$, provided that the sample scatters to that large angle.

Another way is to focus or form the probe to smallest possible diameter, D, and *scan* the object with respect to the probe or vice-versa in order to stitch a real space image from individual scan points. The resolution of the image is then limited by the inverse of the probe diameter $1/D$ or the scanning precision whichever is smaller. Today, the precision of translating probe or specimen using piezo actuators reaches down to atomic level. Atomic force microscopy and scanning tunneling micrsocopy

are remarkable examples for the capabilities of scanning actuators (Binning et al., 1982; Binnig et al., 1986; Binnig and Rohrer, 1987). They profit from a very small probe.

However, for visible light and X-ray microscopy, forming a very small probe for a scanning transmission scheme is challenging. While latest achievements in light manipulation have led to probe sizes below the actual diffraction limit of light (Sotomayor et al., 2007), X-ray probes are still larger than what diffraction limited scattering could provide. Deconvolving the intensity distribution of the probe from the measurement is one possible post-processing approach to increase resolution but it requires to determine the (two-dimensional) beam profile from a separate measurement.

A different approach to reach *super-resolution*, which we understand here as a resolution that exceeds the inverse probe size, is to record X-ray diffraction along with the translational shifts of the probe. This requires to replace the point detector (diode or quadrant detector) with a pixelated one. As the maximum resolution is then set by the scattering amplitude just like for single-pattern CDI, the technique provides the information overcome resolution restrictions imposed by the probe's diameter.

In brief, *Ptychography* is the method to numerically retrieve the object transmission function from four-dimensional datasets $I(\mathbf{y}, \mathbf{s})$ that contains intensity measurement with respect to a the lateral shift $\mathbf{y}$ and for multiple scattering angles $\mathbf{s}/\mathrm{z}$.

This section is dedicated to a variety of aspects of ptychography. In 2.4.1 we will see that the four-dimensional dataset allows to deconvolve the probe from the measurement almost analytically in a process called *Wigner distribution deconvolution*. Though the phase space of the input data is technically four-dimensional it is highly constrained for coherent sources. Subsection 2.4.3 will derive formal conditions for ptychography reinterpreted as a reconstruction technique for a sparse sampled phase space. Due to its additional two scanning dimensions, ptychography withholds sample vibrations or jitter as a new source for data degradation. Subsection 2.4.2 addresses this issue in detail and unifies partial coherence and vibration into a unified four-dimensional convolution kernel.

2.4.1. Wigner distribution deconvolution

We assume now that sample correlation O and illumination correlation P are shifted with respect to each other in the sample plane ($z = 0$). For each shift $\mathbf{y}$ a diffraction pattern is recorded, leading to a 4-dimensional measurement of intensity. In paraxial

approximation (2.127), the measurement may be expressed as

$$I(\mathbf{y},\mathbf{s}) = \hat{\underset{\mathbf{x}_1,\mathbf{s}}{\mathcal{P}}}\,\underset{\mathbf{x}_2,\mathbf{s}}{\mathcal{P}}\left\{P(\mathbf{x}_1,\mathbf{x}_2)O(\mathbf{x}_1-\mathbf{y},\mathbf{x}_2-\mathbf{y})\right\} \tag{2.157}$$

which simplifies for the far-field to

$$I(\mathbf{y},\mathbf{q}) = \iint P(\mathbf{x}_1,\mathbf{x}_2)O(\mathbf{x}_1-\mathbf{y},\mathbf{x}_2-\mathbf{y})e^{2\pi\mathrm{i}(\mathbf{x}_1-\mathbf{x}_2)\mathbf{q}}\mathrm{d}\mathbf{x}_1\mathrm{d}\mathbf{x}_2. \tag{2.158}$$

We switch to other integration axes, $\mathbf{x}=\mathbf{x}_2-\mathbf{x}_1$ and $\mathbf{a}=(\mathbf{x}_2+\mathbf{x}_1)/2$ with the same volume $\mathrm{d}\mathbf{a}\mathrm{d}\mathbf{x}=\mathrm{d}\mathbf{x}_1\mathrm{d}\mathbf{x}_2$ in four-dimensional space.

$$I(\mathbf{y},\mathbf{q}) = \iint P\left(\mathbf{a}-\frac{\mathbf{x}}{2},\mathbf{a}+\frac{\mathbf{x}}{2}\right)O\left(\mathbf{a}-\frac{\mathbf{x}}{2}-\mathbf{y},\mathbf{a}+\frac{\mathbf{x}}{2}-\mathbf{y}\right)e^{-2\pi\mathrm{i}\mathbf{x}\mathbf{q}}\mathrm{d}\mathbf{x}\mathrm{d}\mathbf{a}\,. \tag{2.159}$$

Next, we Fourier transform with respect to $\mathbf{y}$ and shift the integration by an offset $\mathbf{a}$. We obtain

$$\underset{\mathbf{y},\mathbf{w}}{\mathcal{F}}\left\{I(\mathbf{y},\mathbf{q})\right\} = \iiint P\left(\mathbf{a}-\frac{\mathbf{x}}{2},\mathbf{a}+\frac{\mathbf{x}}{2}\right)O\left(\mathbf{a}-\frac{\mathbf{x}}{2}-\mathbf{y},\mathbf{a}+\frac{\mathbf{x}}{2}-\mathbf{y}\right)e^{-2\pi\mathrm{i}\mathbf{x}\mathbf{q}}e^{-2\pi\mathrm{i}\mathbf{y}\mathbf{w}}\mathrm{d}\mathbf{x}\mathrm{d}\mathbf{a}\mathrm{d}\mathbf{y} \tag{2.160}$$

$$= \iiint P\left(\mathbf{a}-\frac{\mathbf{x}}{2},\mathbf{a}+\frac{\mathbf{x}}{2}\right)O\left(\mathbf{a}'-\frac{\mathbf{x}}{2},\mathbf{a}'+\frac{\mathbf{x}}{2}\right)e^{2\pi\mathrm{i}(\mathbf{a}'-\mathbf{a})\mathbf{w}}e^{-2\pi\mathrm{i}\mathbf{x}\mathbf{q}}\mathrm{d}\mathbf{a}\mathrm{d}\mathbf{a}'\mathrm{d}\mathbf{x} \tag{2.161}$$

$$= \int \mathcal{A}_P(\mathbf{w},\mathbf{x})\mathcal{A}_O(-\mathbf{w},\mathbf{x})e^{-2\pi\mathrm{i}\mathbf{x}\mathbf{q}}\mathrm{d}\mathbf{x}\,, \tag{2.162}$$

which corresponds to a convolution with respect to $\mathbf{q}$. The convolution can be factorized into the individual ambiguity functions by means of another Fourier transform (inverse or forward).

$$\underset{\mathbf{q},\mathbf{x}}{\mathcal{F}^*}\left\{\underset{\mathbf{y},\mathbf{w}}{\mathcal{F}}\left\{I(\mathbf{y},\mathbf{q})\right\}\right\} = \mathcal{A}_P(\mathbf{w},\mathbf{x})\mathcal{A}_O(-\mathbf{w},\mathbf{x}) \qquad \text{or} \tag{2.163}$$

$$\underset{\mathbf{q},\mathbf{x}}{\mathcal{F}}\left\{\underset{\mathbf{y},\mathbf{w}}{\mathcal{F}}\left\{I(\mathbf{y},\mathbf{q})\right\}\right\} = \mathcal{F}I(\mathbf{w},\mathbf{x}) = \mathcal{A}_P(\mathbf{w},-\mathbf{x})\mathcal{A}_O(-\mathbf{w},-\mathbf{x}) \tag{2.164}$$

The diffraction pattern stack itself provides a direct interpretation. It can be understood as a four dimensional convolution,

$$I(\mathbf{y},\mathbf{q}) = \mathcal{R}_P(\mathbf{y},-\mathbf{q}) \otimes \mathcal{R}_O(-\mathbf{y},-\mathbf{q})\,, \tag{2.165}$$

where

$$\mathcal{R}_P(\mathbf{y},\mathbf{q}) = \int P\left(\mathbf{y}-\frac{\mathbf{x}}{2},\mathbf{y}+\frac{\mathbf{x}}{2}\right)e^{2\pi\mathrm{i}\,\mathbf{x}\mathbf{q}}\mathrm{d}\mathbf{x} = \mathcal{F}^*\mathcal{A}_P(\mathbf{y},\mathbf{q}) \tag{2.166}$$

is known as *generalized radiance function*(Nugent, 2007). It is the Fourier inverse to the ambiguity function. We conclude that the ambiguity function of the illumination can be retrieved from the stack of diffraction patterns $I(\mathbf{y},\mathbf{q})$ if the ambiguity function

of the object is known and vice versa, by means of a *deconvolution in four-dimensional space*. In practice, the deconvolution may be carried out in Fourier space (eq.(2.164)) using a Wiener filter:

$$\mathcal{A}_O(-\mathbf{w},-\mathbf{x}) = \mathcal{F}I(\mathbf{x},\mathbf{w})\frac{{\mathcal{A}_P}^*(\mathbf{w},-\mathbf{x})}{\left|\mathcal{A}_P(\mathbf{w},-\mathbf{x})\right|^2+\varepsilon}\,. \tag{2.167}$$

A slight variation of the ambiguity function is also known as *Wigner-distribution function*,

$$\mathcal{W}_P(\mathbf{q},\mathbf{x}) \equiv \mathcal{A}_P(-\mathbf{q},\mathbf{x})\,, \tag{2.168}$$

with which all calculations from before may be carried out in a similar fashion. Hence, the technique to deconvolve the diffraction pattern stack with ambiguity functions is termed *Wigner-distribution deconvolution (WDD)* in literature (Chapman, 1996).

Assuming, the diffraction pattern stack was deconvolved due to detailed knowledge of the illumination correlation, we would find, *ab initio* an intensity image of the object be Fourier transforming eq.(2.167)

$$I_O(\mathbf{y}) = O(\mathbf{y},\mathbf{y}) = \underset{\mathbf{w},\mathbf{y}}{\mathcal{F}^*}\left\{\mathcal{A}_O(\mathbf{w},0)\right\}(\mathbf{y})\,. \tag{2.169}$$

The image of the object transmission is sampled by the same coordinate $\mathbf{y}$ that was used to laterally shift object versus probe in the source plane. *The resolution of the image that is retrieved ab initio from a WDD corresponds to the pitch of the scanning raster*. This is a significant disadvantage when compared to the effort of taking samples in four dimensions, but it is also the least to expect without assuming any coherent correlation in P and O. Hence it would also hold in the *incoherent limit*. Nevertheless, the problem remains that either probe or object correlation needs to be known *a priori* for a successful deconvolution.

The deconvolution theorem applies to partially coherent wave fields as no special assumptions on the field correlation of probe or object have been made. However, coherent fields provide leverage for more evolved deconvolution schemes. For example, an entirely coherent ambiguity function

$$\mathcal{A}_{O_{\mathrm{coh}}}(\mathbf{w},\mathbf{x}) = \int o^*\left(\mathbf{s}-\frac{\mathbf{x}}{2}\right) o\left(\mathbf{s}+\frac{\mathbf{x}}{2}\right) e^{-2\pi\mathrm{i}\,\mathbf{s}\mathbf{w}}\mathrm{d}\mathbf{s} \tag{2.170}$$

is rather a four-dimensional image of a two-dimensional quantity. Both $\mathbf{x}$ and $\mathbf{w}$ are essentially tied to the same real-space coordinate. We may hence regard coherence as a way that constrains the four-dimensional phase space, effectively reducing its dimensionality. McCallum and Rodenburg (1993) demonstrated the applicability of an iterative blind deconvolution for the case of coherent ambiguity functions. In fact, the four-dimensional phase-space may become so strongly constrained that even few samples of the shift $\mathbf{y}_\mathrm{j}$ suffice to reconstruct the entire phase-space.

2.4.2. Vibrations

From the section about partial coherence, we learned that a complex degree of coherence γ which depends only on the difference between two points ($\gamma(\mathbf{x}_1, \mathbf{x}_2) = \tilde{\gamma}(\mathbf{x}_1 - \mathbf{x}_2)$) manifests itself as a convolution over spatial frequencies (see (2.156)) in the measured intensity,

$$I(\mathbf{y}, \mathbf{q}) = \mathcal{F}^*\tilde{\gamma}^*(\mathbf{q}) \otimes_{\mathbf{q}} I_{\mathrm{coh}}(\mathbf{y}, \mathbf{q}) \,, \tag{2.171}$$

where $\mathcal{F}^*\tilde{\gamma}^*(\mathbf{q})$ is the detector point spread function. Introducing additional scan dimensions $\mathbf{y}$ in the measurements also withholds a new error source. Assuming that for each shift $\mathbf{y}$ the sample or the illumination shake or vibrate with respect to each other, performing the (small) additional movement $\mathbf{u}(t)$, we need to rewrite the measurement as an average over the exposure time,

$$I(\mathbf{y}, \mathbf{q}) = \frac{1}{T} \int_{t_0}^{t_0+T} I_{\mathrm{coh}}(\mathbf{y} - \mathbf{u}(t), \mathbf{q}) \, \mathrm{d}t \,. \tag{2.172}$$

If the movement's period is shorter than the exposure and repeatable (i.e. stationary), the ergodicity principle allows to replace the time average integral by a space integral with an appropriate probability density function $\rho(\mathbf{u})$. $\rho(\mathbf{u})\mathrm{d}\mathbf{u}$ signifies the probability for an offset $\mathbf{u}$ to occur within one measurement.

$$I(\mathbf{y}, \mathbf{q}) = \int \rho(\mathbf{u}) \, I_{\mathrm{coh}}(\mathbf{y} - \mathbf{u}, \mathbf{q}) \, \mathrm{d}\mathbf{u} \,. \tag{2.173}$$

We readily identify the expression to the right term as a convolution with ρ which will be called *offset density function (ODF)* from here on.

$$I(\mathbf{y}, \mathbf{q}) = \rho(\mathbf{y}) \otimes_{\mathbf{y}} I_{\mathrm{coh}}(\mathbf{y}, \mathbf{q}) \,. \tag{2.174}$$

We assumed that the vibration $\mathbf{u}(t)$ is inherently invariant to the actual shift $\mathbf{y}$.

Together, with the PSF from partial coherence we note that translational invariant sources for decoherence act as a convolution in four-dimensional space.

$$I(\mathbf{y}, \mathbf{q}) = \rho(\mathbf{y})\mathcal{F}^*\tilde{\gamma}^*(\mathbf{q}) \otimes I_{\mathrm{coh}}(\mathbf{y}, \mathbf{q}) \tag{2.175}$$

The (real) convolution kernel $\rho(\mathbf{y})\mathcal{F}^*\tilde{\gamma}^*(\mathbf{q})$ blurs the four-dimensional speckles in $I(\mathbf{y}, \mathbf{q})$ which are essentially our information units. Whether vibrations or a partially coherent illumination have a stronger impact on degrading the speckle quality depends on the type of probe and object. Fig. 2.4 compares the effect of vibration and partial spatial coherence for three different probes. For simplicity we assumed both $\mathbf{y}$

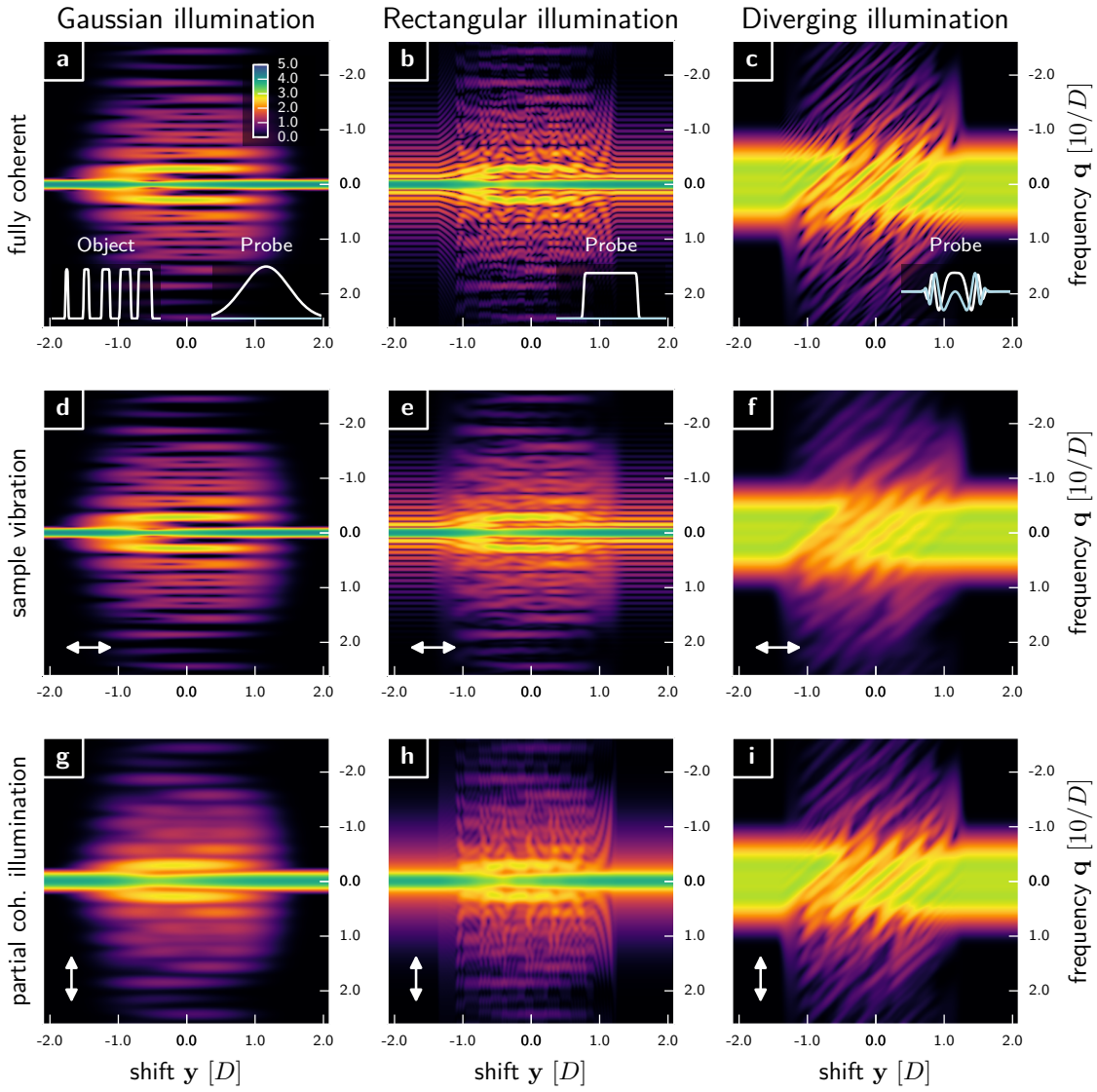


Figure 2.4. – Effect of partial spatial coherence and vibration on the $(\mathbf{y}, \mathbf{q})$ phase-space of ptychography. **a-c** Completely coherent case for an asymmetric absorption grid of length $2D$ as object (lower left inset) and different probes (lower right inset, real and real part are white and light blue, respectively) with the same FWHM D. **d-f** Phase-space blurring introduced by a vibration having a Gaussian ODF with a FWHM of $0.2D$. The blurring direction is indicated by a double arrow. Rectangular (**e**) and diverging probe (**f**) are stronger affected than the Gaussian probe (**d**). **g-i** Phase-space blurring introduced by partially coherent illumination of a Gaussian PSF with FWHM of $1.25/D$ ($\approx 0.7D$ coherence length FWHM). The blurring direction is indicated by a double arrow. Gaussian (**g**) and rectangular probe (**h**) are stronger affected than the diverging probe (**i**). All phase space images share the same logarithmic color scale as in **a**.

and $\mathbf{q}$ to have only one dimension each. The rectangular probe (which corresponds to a disc-like probe in 2D) creates the finest-grained speckle pattern and is affected both by vibration and partial coherence. The Gaussian probe creates almost no speckle in $\mathbf{y}$-direction, which is due its long smooth tails. However, these long tails make the Gaussian beam sensitive to partial coherence and its phase-space data suffers mostly from the convolution with the PSF. The diverging probe corresponds, for example, to a KB-mirror or fresnel zone plate illumination in 2D. It is mostly affected by the vibration ODF due to its fine structure in $\mathbf{y}$-direction while being a bit more resilient towards degradation from a PSF.

The effect of PSF and ODF may be best understood in reciprocal space. Here, they do not appear as convolution kernels but as complex-valued envelopes,

$$\mathcal{F}I(\mathbf{w},\mathbf{x}) = \mathcal{F}\rho(\mathbf{w})\,\tilde{\gamma}(-\mathbf{x})\,\mathcal{A}_{P_{\mathrm{coh}}}(\mathbf{w},-\mathbf{x})\mathcal{A}_{O_{\mathrm{coh}}}(-\mathbf{w},-\mathbf{x})\,, \tag{2.176}$$

where the relation $\tilde{\gamma}^*(\mathbf{x}) = \tilde{\gamma}(-\mathbf{x})$ was used. We readily observe that $\mathcal{F}\rho$ and $\tilde{\gamma}$ introduce partial coherence by truncating the (coherent) signal in $(\mathbf{w},\mathbf{x})$-space. However, if the product of $\mathcal{A}_{P_{\mathrm{coh}}}$ and $\mathcal{A}_{O_{\mathrm{coh}}}$ originally extended within a smaller volume than the limits imposed by $\mathcal{F}\rho$ and $\tilde{\gamma}$, the diffraction experiment remains unaffected. In practice, for the common ptychography setup of a confined probe scanning a larger object, it means that the probe must be smaller than the coherence length. In order to be resilient towards vibrations, either object or probe must have a smoother ratio of contrast to minimum feature size than the blurring kernel of the ODF.

2.4.3. Ptychography

In order to reach a resolution that is superior to the inverse step size, coherent interferences needs to be taken into account. In the *coherent limit*, the intensity measurements can be expressed as

$$I_{\mathrm{coh}}(\mathbf{y},\mathbf{s}) = \hat{\underset{\mathbf{x}_1,\mathbf{s}}{\mathcal{P}}}\,\underset{\mathbf{x}_2,\mathbf{s}}{\mathcal{P}}\left\{p^*(\mathbf{x}_1)p(\mathbf{x}_2)o^*(\mathbf{x}_1-\mathbf{y})o(\mathbf{x}_2-\mathbf{y})\right\} \tag{2.177}$$

$$= \left|\underset{\mathbf{x},\mathbf{s}}{\mathcal{P}}\left\{p(\mathbf{x})o(\mathbf{x}-\mathbf{y})\right\}\right|^2\,, \tag{2.178}$$

which, in the far-field, takes on the shape

$$I_{\mathrm{coh}}(\mathbf{y},\mathbf{q}) = \left|\underset{\mathbf{x},\mathbf{q}}{\mathcal{F}}\left\{p(\mathbf{x})o(\mathbf{x}-\mathbf{y})\right\}\right|^2\,. \tag{2.179}$$

The reciprocal phase space is then a product of coherent ambiguity functions,

$$\mathcal{F}I(\mathbf{w},\mathbf{x}) = \mathcal{A}_{P_{\mathrm{coh}}}(\mathbf{w},-\mathbf{x})\mathcal{A}_{O_{\mathrm{coh}}}(-\mathbf{w},-\mathbf{x}) \tag{2.180}$$

which is a special case of eq.(2.164). As mentioned previously, these ambiguity functions are now essentially four-dimensional images of two-dimensional quantities and thus highly constrained. Assuming that the object ambiguity was retrieved from deconvolution we readily identify its inverse Fourier transform

$$A(\mathbf{y},\mathbf{x}) \equiv \underset{\mathbf{w},\mathbf{y}}{\mathcal{F}^*}\{\mathcal{A}_{O_{\mathrm{coh}}}\}(\mathbf{y},\mathbf{x}) = o^*\left(\mathbf{y}-\frac{\mathbf{x}}{2}\right) o\left(\mathbf{y}+\frac{\mathbf{x}}{2}\right) \tag{2.181}$$

as an interference of the object function with itself. In particular, for any two shifts $\mathbf{y}_1$ and $\mathbf{y}_2$ we find a specific region in A,

$$A_{12}(\mathbf{x}) \equiv A\left(\frac{\mathbf{y}_1-\mathbf{y}_2}{2}, 2\mathbf{x}-\mathbf{y}_1-\mathbf{y}_2\right) = o^*(\mathbf{y}_1-\mathbf{x})\, o(\mathbf{x}-\mathbf{y}_2)\ , \tag{2.182}$$

that correlates the object transmission centred around the shifts. If the samples of the ambiguity function were independent, the same would hold true for A_{ij}. However, if the above equation (2.182) was repeated for two more shift combinations A_{23} and A_{31} (with a third shift $\mathbf{y}_3$), we would find enough independent equations to solve for o around all three of those shift vectors. That means, that o can be accurately determined from its ambiguity function with only a small number of shifts $\mathbf{y}_j$. If we added the probe to these equations as additional variable function, we would need more shifts $\mathbf{y}_j$ in total. Still, a small amount of shifts would suffice to fill the four dimensional phase space of both probe and object in a self-consistent manner.

We conclude that

- in forward scattering geometry (sec. 2.1.2), and,
- under coherent or almost coherent conditions,
- diffraction patterns I_j may be recorded for
- a limited number of shifts $\mathbf{y}_\mathrm{j}$ between probe and object

in order to self-consistently fill the four-dimensional phase space of probe and object ambiguity. Complex wave fields of probe and object may be retrieved simultaneously. The sparse-sampled measurement of phase space,

$$I_\mathrm{j}(\mathbf{q}) = \left|\underset{\mathbf{x},\mathbf{q}}{\mathcal{F}}\{p(\mathbf{x})o(\mathbf{x}-\mathbf{y}_\mathrm{j})\}\right|^2 \tag{2.183}$$

in combination with the retrieval process is commonly called *ptychography*. We will denote the outcome of such a process as a *ptychographic reconstruction* of probe and object or their phase-spaces. As other algorithmic reconstruction methods rarely appear in this thesis, we will drop the adjective "ptychographic" whenever it does not lead to ambiguity. Iterative algorithms for efficient reconstruction are the topic of Section 2.4.5.

Overlap

Sufficient *overlap* is an elementary imperative for ptychography associated with the additional scanning dimensions $\mathbf{y}$. Let us assume that the scan pattern or scan raster $\mathbf{y}_0, \mathbf{y}_1, \ldots \mathbf{y}_\mathsf{J}$ has a smallest pitch $\triangle y$ such that

$$\forall \mathsf{j}, \mathsf{i} \,:\, |\mathbf{y}_\mathsf{j} - \mathbf{y}_\mathsf{i}| > \triangle y \,. \tag{2.184}$$

If the extent of the probe D is smaller than in the scanning pattern then the exit waves at each scan point,

$$\psi_\mathsf{j}(\mathbf{x}) = p(\mathbf{x})\, o(\mathbf{x} - \mathbf{y}_\mathsf{j}) = p(\mathbf{x})\, \tilde{o}_\mathsf{j}(\mathbf{x}) \,, \tag{2.185}$$

are independent from each other because the shifted object transmission $\tilde{o}_\mathsf{j}$ has to comply only locally with the data. In this case, the phase pace $I(\mathbf{y}, \mathbf{q})$ is sampled to coarse in $\mathbf{y}$ and the phase-retrieval problem matches those of single-pattern CDI for each independent object transmissions $\tilde{o}_\mathsf{j}$.

Since its inception (Faulkner and Rodenburg, 2004), it is thus known that far-field ptychography with a localised probe requires a certain overlap of the probe between adjacent scan points. Bunk et al. (2008) studied the effect of reconstruction quality as a function of the linear overlap ratio

$$\sigma_\text{overlap} = 1 - \frac{\triangle y}{D} \tag{2.186}$$

and found an optimal value around $\sigma_\text{overlap} \approx 60\,\%$ within a bracket of acceptable overlaps of 30 %–85 %.

The linear overlap ratio is not suitable to characterise the diverse set of scanning patterns available, especially if the scanning pattern is asymmetric. Huang et al. (2014) and Wakonig (2015) analysed the dependence on the scanning pattern in detail and concluded that it a uniform scanning pattern is advised. Both studies proposed an alternative overlap ratio. For the latter study, the overlap ratio was computed with

$$\sigma_{OL} = \frac{\sum_\mathsf{j} \sum_{\mathsf{i}\neq\mathsf{j}} \int |p(\mathbf{x} - \mathbf{y}_\mathsf{j})||p(\mathbf{x} - \mathbf{y}_\mathsf{i})| \,\mathrm{d}\mathbf{x}}{\mathsf{J}(\mathsf{J}-1) \int |p(\mathbf{x})|^2 \,\mathrm{d}\mathbf{x}} \tag{2.187}$$

which is independent of the scan point order[6] and thus a good benchmark for a scanning pattern. Without integral in the numerator, the overlap ratio may also be interpreted as an "overlap map" to visualize overlap uniformity.

[6] The promoted overlap "ratio" from Huang et al. (2014) depends on the actual order of scan points and lacks the integral sum over $\mathbf{x}$ both in numerator and denominator. Their argument is thus difficult to comprehend and (2.187) seems to be a superior notion for the overlap ratio.

Similarly, we can derive an overlap ratio from the ambiguity function of the probe:

$$\tilde{\sigma}_{OL} = \frac{\sum_{\mathsf{j}} \sum_{\mathsf{i}\neq\mathsf{j}} A_p(0, \mathbf{y}_{\mathsf{j}} - \mathbf{y}_{\mathsf{i}})}{\mathsf{J}(\mathsf{J}-1) A_p(0,0)}\,, \tag{2.188}$$

which would be equivalent to (2.187) if an inner product was used instead of the modulus product. Definition (2.188) can be readily applied also to partially coherent conditions.

2.4.4. Coherent ambiguity function synthesis

Let us recall the effect of vibrations and partial spatial coherence on the phase space as described with eq. (2.176):

$$\mathcal{F}I(\mathbf{w},\mathbf{x}) = \mathcal{F}\rho(\mathbf{w})\,\tilde{\gamma}(-\mathbf{x})\,\mathcal{A}_{P_{\mathrm{coh}}}(\mathbf{w},-\mathbf{x})\mathcal{A}_{O_{\mathrm{coh}}}(-\mathbf{w},-\mathbf{x})\,.$$

We assume now that $\mathcal{F}\rho$ and $\tilde{\gamma}$ continuously squeeze the $(\mathbf{w},\mathbf{x})$-volume as a consequence of increasing vibrations or reduced transverse coherence. First, we might argue that these changes could be ignored, since any ptychographic algorithm would try its best to find a solution for the smaller volume confined by $\mathcal{F}\rho$ and $\tilde{\gamma}$. For example, it could converge towards a smaller probe or smoother object, since both ambiguity may be used independently to address the shrunk extent in $\mathbf{w}$ and $\mathbf{x}$. Unfortunately the model of ptychography as stated before in eq. (2.183) applies coherent ambiguity functions to fit to the data. For coherent ambiguity functions the arguments are *not* independent as they are four-dimensional transforms of two-dimensional quantities. Eventually, the algorithm will notice a discrepancy between the sampled measurement and what its coherent ambiguities can deliver to fit the data. In other words, cutting the tails in $(\mathbf{w},\mathbf{x})$-space feeds back to the cores of coherent ambiguity functions, rendering the problem ill-posed for coherent analysis. Of course, this problem can be alleviated with a priori knowledge about ρ and $\tilde{\gamma}$, but that is not always at hand or imprecise.

At this point the mode decomposition as proposed by Thibault and Guizar-Sicairos (2012) and derived in section 2.3.4 will provide relief. For example, let $\mathcal{A}_{p_m}$ and $\mathcal{A}_{o_n}$ be the (coherent) ambiguity functions of probe and object modes respectively, then eq. (2.176) may be rephrased to

$$\mathcal{F}I(\mathbf{w},\mathbf{x}) = \sum_m \sum_n \mathcal{A}_{p_m}(\mathbf{w},-\mathbf{x})\mathcal{A}_{o_n}(-\mathbf{w},-\mathbf{x})\,. \tag{2.189}$$

We may interpret this expression as some form of *coherent ambiguity function synthesis*. Any confined or constrained quantity in $(\mathbf{w},\mathbf{x})$-space may thus be synthesized by a series of coherent ambiguity functions who are not bound by these confinements – a method similar to Fourier synthesis where functions with compact carrier are modelled from a (possibly infinite) series of cosine and sine functions.

2.4.5. Iterative algorithms

Reconstruction algorithms form an essential part of ptychographic techniques. The last decade has seen important progress in the way computer algorithms (called "engines" in this paper) explore the high-dimensional phase space of possible sample transmission and illumination functions in the search for a unique solution. The first occurance of an iterative algorithm was the *Ptychographic Iterative Engine* (PIE) (Faulkner and Rodenburg, 2004) which addressed the ptychographic reconstruction problem through sequential updates inspired by previous reconstruction approaches (Gerchberg and Saxton, 1972; Fienup, 1986; Miao et al., 1999). The technique is still widely used today in one of its updated versions (*ePIE* (Maiden and Rodenburg, 2009), *3PIE* (Maiden et al., 2012)). Whereas the PIE familly iteratively cycles through the diffraction patterns, other engines act in a parallel fashion, following a generalized projection formalism. Notable examples are the *Difference Map (DM)* (Thibault, 2008; Dierolf et al., 2010b), the *Relaxed Averaged Alternating Relections* algorithm (RAAR) (Luke, 2004), and other similar formulations (Marchesini et al., 2012). Other parallel reconstruction techniques originate from cost-function optimisation techniques (Guizar-Sicairos and Fienup, 2008b) such as those based on *Maximum Likelihood principles (ML)* (Thibault and Guizar-Sicairos, 2012).

Here, we review DM and ML engine explicitly for the simplest model, coherent far-field ptychograpy. The extension to complexer models is described in section 2.5

Difference Map algorithm

Thibault (2008) were the first to propose an engine based on difference map (see Sec. 2.2.4) to ptychography. Other than operating on a single density distribution the DM implementation iterates on a high-dimensional state vector comprised of a (coherent) wave field for each shift $\mathbf{y}_\mathrm{j}$

$$\mho = \{\psi_0, \psi_1, ...\psi_J\} \tag{2.190}$$

The difference map formalism (sec. 2.2.4) requires two projections for two independent constraint sets. They arise from the assumption of ptychography, that for each lateral shift $\mathbf{y}_\mathrm{j}$ the *exit wave* ψ_j satisfies

$$\text{the } \textit{overlap contraint}, \quad \psi_\mathrm{j}(\mathbf{x}) = p(\mathbf{x})\, o(\mathbf{x} - \mathbf{y}_\mathrm{j}) \tag{2.191}$$

$$\text{and the } \textit{Fourier modulus contraint}, \quad |\mathcal{F}\psi_\mathrm{j}(\mathbf{q})|^2 = \mathsf{M}_\mathrm{j}(\mathbf{q})\,. \tag{2.192}$$

The state vector is closest to the first constraint if the euklidean distance

$$\mathcal{L} = \sum_{\mathrm{j}} \int \left| \psi_{\mathrm{j}}(\mathbf{x}) - p(\mathbf{x})\, o(\mathbf{x} - \mathbf{y}_{\mathrm{j}}) \right|^2 \mathrm{d}\mathbf{x} \tag{2.193}$$

is minimised. The minimum is found where the gradients $\nabla_p \mathcal{L}$ and $\nabla_o \mathcal{L}$ vanish,

$$0 \stackrel{!}{=} \frac{\partial \mathcal{L}}{\partial p(\mathbf{x})} = \sum_{\mathrm{j}} p^*(\mathbf{x}) |o(\mathbf{x} - \mathbf{y}_{\mathrm{j}})|^2 - \psi_{\mathrm{j}}^*(\mathbf{x}) o(\mathbf{x} - \mathbf{y}_{\mathrm{j}}) \tag{2.194}$$

$$0 \stackrel{!}{=} \frac{\partial \mathcal{L}}{\partial o(\mathbf{x})} = \sum_{\mathrm{j}} o^*(\mathbf{x}) |p(\mathbf{x} + \mathbf{y}_{\mathrm{j}})|^2 - \psi_{\mathrm{j}}^*(\mathbf{x} + \mathbf{y}_{\mathrm{j}}) p(\mathbf{x} + \mathbf{y}_{\mathrm{j}})\,. \tag{2.195}$$

We use the above equations to formalute the *overlap projection*:

$$\Pi_{\mathrm{O}}(\mho) : \psi_{\mathrm{j}}(\mathbf{x}) \to p(\mathbf{x}) o(\mathbf{x} - \mathbf{y}_{\mathrm{j}}) \tag{2.196}$$

where the fields p and o are found by alternately computing

$$p(\mathbf{x}) = \frac{\sum_{\mathrm{j}} \psi_{\mathrm{j}}^*(\mathbf{x}) o(\mathbf{x} - \mathbf{y}_{\mathrm{j}})}{\sum_{\mathrm{j}} |o(\mathbf{x} - \mathbf{y}_{\mathrm{j}})|^2} \tag{2.197}$$

$$\text{and} \quad o(\mathbf{x}) = \frac{\sum_{\mathrm{j}} \psi_{\mathrm{j}}^*(\mathbf{x} + \mathbf{y}_{\mathrm{j}}) p(\mathbf{x} + \mathbf{y}_{\mathrm{j}})}{\sum_{\mathrm{j}} |p(\mathbf{x} + \mathbf{y}_{\mathrm{j}})|^2}\,. \tag{2.198}$$

Steps (2.197) and (2.198) may need to be repeated a few times before the projection is complete (Thibault, 2008).

The other projection stems from the Fourier modulus constraint (2.71).

$$\Pi_{\mathrm{F}}(\mho) : \psi_{\mathrm{j}}(\mathbf{x}) \to \pi_F(\psi_{\mathrm{j}}) \tag{2.199}$$

With these two projectors and standard parameters (Elser, 2003, 2002), we can formulate a *difference map (DM)* algorithm with the update rule

$$\mho^{\mathsf{it}+1} = \mho^{\mathsf{it}} + \Pi_{\mathrm{F}}(2\Pi_{\mathrm{O}}(\mho^{\mathsf{it}}) - \mho^{\mathsf{it}}) - \Pi_{\mathrm{O}}(\mho^{\mathsf{it}})\,. \tag{2.200}$$

In this DM implementation, the Fourier modulus projection does not constrain the exit waves directly, but acts on an auxiliary functions v^{it} formed from difference between the most recent updated probe and object, o^{it} and the exit waves, $\psi_{\mathrm{j}}^{\mathsf{it}}$, from the last iteration. From a wave field (data-centric) perspective, DM consists of four steps.

1. From the exit waves, $\psi_{\mathrm{j}}^{\mathsf{it}}$, create current estimates for probe, p^{it}, and object function o^{it} enforcing the overlap constraint.

2. Build auxiliary wave functions

$$\upsilon_{\mathsf{j}}^{\mathsf{it}}(\mathbf{x}) = 2\, p^{\mathsf{it}}(\mathbf{x})\, o^{\mathsf{it}}(\mathbf{x} - \mathbf{x}_{\mathsf{j}}) - \psi_{\mathsf{j}}^{\mathsf{it}}(\mathbf{x})\,. \tag{2.201}$$

3. Apply the Fourier modulus projection on each auxiliary wave,

$$\upsilon_{\mathsf{j}}^{\mathsf{it}+1} = \pi_F(\upsilon_{\mathsf{j}}^{\mathsf{it}})\,. \tag{2.202}$$

4. Form an update of the exit waves with

$$\psi_{\mathsf{j}}^{\mathsf{it}+1}(\mathbf{x}) = \psi_{\mathsf{j}}^{\mathsf{it}}(\mathbf{x}) + \upsilon_{\mathsf{j}}^{\mathsf{it}+1}(\mathbf{x}) - p^{\mathsf{it}}(\mathbf{x})\, o^{\mathsf{it}}(\mathbf{x} - \mathbf{x}_{\mathsf{j}})\,. \tag{2.203}$$

Steps 2.-4. may be parallelised and distributed to concurrent processes where each holds a part of the diffraction data. Step 1. requires interprocess communication and data exchange in order to compute the sums in (2.197) and (2.198).

Maximum Likelihood algorithm

The maximum likelihood engine for ptychography was first derived (Thibault and Guizar-Sicairos, 2012). The basic principle is to derive a cost function for the model that includes the statistical properties of he measurements as weights. The cost function is the negative logarithm of the likelihood and is minimised using a conjugate gradient approach with the search direction determined via the Polak-Riebière formula. The final minimisation occurs along the new search direction. As the steps after gradient calculation are common for conjugate gradient minimisation, we will not delve deeper than gradient calculation here.

Consistent with (2.183), we assume that

$$I_{\mathsf{j}}(\mathbf{q}) = \left|\mathcal{F}\psi_{\mathsf{j}}(\mathbf{q})\right|^2 = \left|\underset{\mathbf{x},\mathbf{q}}{\mathcal{F}}\{p(\mathbf{x})o(\mathbf{x} - \mathbf{y}_{\mathsf{j}})\}\right|^2 \tag{2.204}$$

is the model for an accurate fit to the measurements $\mathsf{M}_{\mathsf{j}}(\mathbf{q})$. Then, for Poisson statistics, the negative log-likelihood may be written as

$$\mathcal{L} = -\sum_{\mathsf{j}} \int \mathsf{m}_{\mathsf{j}}(\mathbf{q}) \Big[\mathsf{M}_{\mathsf{j}}(\mathbf{q}) \log I_{\mathsf{j}}(\mathbf{q}) - I_{\mathsf{j}}(\mathbf{q}) - \log(\mathsf{M}_{\mathsf{j}}(\mathbf{q})!)\Big] \mathrm{d}\mathbf{q}\,. \tag{2.205}$$

where $\mathsf{m}_{\mathsf{j}}(\mathbf{q})$ masks the support of sampled regions in the detector plane. The last term in (2.205) is constant and can be neglected for minimisation. As the model

(2.183) depends only on the product ψ of p and o and not individually on either of the two we find the derivatives to be

$$\nabla_p \mathcal{L} = \frac{\partial \mathcal{L}}{\partial p(\mathbf{x})} = \sum_{\mathsf{j}} o(\mathbf{x} - \mathbf{x}_{\mathsf{j}}) \varphi_{\mathsf{j}}^*(\mathbf{x}) \tag{2.206}$$

$$\nabla_o \mathcal{L} = \frac{\partial \mathcal{L}}{\partial o(\mathbf{x})} = \sum_{\mathsf{j}} p(\mathbf{x} + \mathbf{x}_{\mathsf{j}}) \varphi_{\mathsf{j}}^*(\mathbf{x} + + \mathbf{x}_{\mathsf{j}}) \,, \tag{2.207}$$

where the auxiliary function φ in propagated space is written as

$$\mathcal{F}\varphi_{\mathsf{j}}(\mathbf{q}) = \frac{\partial \mathcal{L}}{\partial I_{\mathsf{j}}(\mathbf{q})} \mathcal{F}\psi_{\mathsf{j}}(\mathbf{q}) = \mathsf{m}_{\mathsf{j}}(\mathbf{q}) \frac{I_{\mathsf{j}}(\mathbf{q}) - \mathsf{M}_{\mathsf{j}}(\mathbf{q})}{I_{\mathsf{j}}(\mathbf{q})} \mathcal{F}\psi_{\mathsf{j}}(\mathbf{q}) \,. \tag{2.208}$$

If the likelihood follows a Gaussian statistic, the negative log-likelihood cost function $\mathcal{L}$ can be expressed as a weighted sum of squares:

$$\mathcal{L} = \sum_{\mathsf{j}} \int \frac{\mathsf{m}_{\mathsf{j}}(\mathbf{q})}{2\sigma_{\mathsf{j}}(\mathbf{q})^2} \big[I_{\mathsf{j}}(\mathbf{q}) - \mathsf{M}_{\mathsf{j}}(\mathbf{q}) \big]^2 \mathrm{d}\mathbf{q} \,, \tag{2.209}$$

where $\sigma_{\mathsf{j}}(\mathbf{q})$ are the spatially dependent uncertainties. The auxiliary function takes on a similar shape as in (2.208):

$$\mathcal{F}\varphi_{\mathsf{j}}(\mathbf{q}) = \mathsf{m}_{\mathsf{j}}(\mathbf{q}) \frac{\mathsf{M}_{\mathsf{j}}(\mathbf{q}) - I_{\mathsf{j}}(\mathbf{q})}{\sigma_{\mathsf{j}}(\mathbf{q})^2} \mathcal{F}\psi_{\mathsf{j}}(\mathbf{q}) \,. \tag{2.210}$$

In particular, for areas of large photon count, the squared uncertainty matches the photon count

$$\sigma_{\mathsf{j}}(\mathbf{q})^2 = \mathsf{M}_{\mathsf{j}}(\mathbf{q}) \tag{2.211}$$

as a result of dominant photon noise. However, the Gaussian cost function even applies for low intensity regions if the detector suffers from read-out noise or other similar noise forms. Then, the uncertainty has a lower limit of constant value, often greater than 1.

$$\sigma_{\mathsf{j}}(\mathbf{q})^2 >= \alpha \tag{2.212}$$

For example, the sensor KAF-1001E from ON Semiconductor, as used in the Fingerlakes PL1001E camera, has a read-out standard deviation of $\sigma_{\mathsf{j}}(\mathbf{q}) \approx 4 - 5$ photons for green light (500nm). Thibault and Guizar-Sicairos (2012) concluded that it is often safe to assume $\sigma_{\mathsf{j}}(\mathbf{q})^2 = \mathsf{M}_{\mathsf{j}}(\mathbf{q}) + 1$. For best results, the noise model should be calculated alongside other data preparation steps and be provided to the reconstruction algorithm together with the data.

The cost function lends itself naturally as another form of error metric which can be recorded to monitor convergence of the ML algorithm. Ideally, an error metric should

provide some sort of comparability between scans. Rescaling the cost-function by the number of samples or active detector area,

$$\epsilon_{LL} = \frac{\mathcal{L}}{\sum_{\mathsf{j}} \int \mathsf{m}_{\mathsf{j}}(\mathbf{q})\,\mathrm{d}\mathbf{q}}\,, \tag{2.213}$$

results in a *(pixel) average log-likelihood*. This kind of error metric is easy to calculate[7] but it can not compare well experiments of different flux. Hence,

$$\epsilon_{ll} = \frac{\sqrt{\mathcal{L}}}{\sqrt{\sum_{\mathsf{j}} \int \mathsf{m}_{\mathsf{j}}(\mathbf{q})\mathsf{M}_{\mathsf{j}}(\mathbf{q})\,\mathrm{d}\mathbf{q}}}\,, \tag{2.214}$$

which we will call *normalised log-likelihood error*, appears to be a well comparable benchmark among reconstructions of different datasets. Additionally, ϵ_{ll} has the advantage that it is bound by $[0, 1]$ and can thus be expressed as a percentage.

We leave as an open question, whether expressions (2.213) and (2.214) are universally applicable. Their main purposes in this thesis is to translate the log-likelihood in a more user-friendly expression.

2.4.6. Solution space of ptychography

Especially in the far-field, the result of a ptychographic reconstruction is unique only with respect to the autocorrelation of the exit waves $\psi_{\mathsf{j}}(\mathbf{x}) = p(\mathbf{x})o(\mathbf{x} - \mathbf{y}_{\mathsf{j}})$. With any $\mathbf{y}_{\mathrm{f}}, \mathbf{q}_{\mathrm{f}} \in \mathbb{R}^2$ and $c_{\mathrm{f}} \in \mathbb{C}$ we may construct alternative probes and objects,

$$\tilde{p}(\mathbf{x}) = c_{\mathrm{f}} e^{2\pi \mathrm{i}\mathbf{q}_{\mathrm{f}}\mathbf{x}} p(\mathbf{x} - \mathbf{y}_{\mathrm{f}}) \quad \text{and} \quad \tilde{o}(\mathbf{x}) = \frac{1}{c_{\mathrm{f}}} e^{-2\pi \mathrm{i}\mathbf{q}_{\mathrm{f}}\mathbf{x}} o(\mathbf{x} - \mathbf{y}_{\mathrm{f}})\,, \tag{2.215}$$

and note that all these alternatives are compatible with the model

$$I_{\mathsf{j}}(\mathbf{q}) = \left| \underset{\mathbf{x},\mathbf{q}}{\mathcal{F}} \{\tilde{p}(\mathbf{x})\tilde{o}(\mathbf{x} - \mathbf{y}_{\mathsf{j}})\} \right|^2 \tag{2.216}$$

These ambiguities need to be removed or reversed when ptychography is combined with tomography (Dierolf, 2015; Guizar-Sicairos et al., 2014; Daurer, 2013) or when modelled wave fields are compared with their ptychographic reconstructions (Wakonig, 2015).

[7] For example, Ptypy uses this error metric

2.5. Conclusion

The common model for ptychography (2.183) assumes a coherent forward scattering geometry in the far-field,

$$I_\mathrm{j}(\mathbf{q}) = \left| \underset{\mathbf{x},\mathbf{q}}{\mathcal{F}} \{p(\mathbf{x}) o(\mathbf{x} - \mathbf{y}_\mathrm{j})\} \right|^2 . \tag{2.217}$$

Important developments of ptychography hinge on the introduction of a new level of complexity compared to the model (2.217). Based on the previous theory, this section summarises how the model is incrementally adapted.

1. **Propagation distance**
   Like some imaging methods in the holographic regime, ptychography is not limited to far-field diffraction but may use any propagator that matches the experimental conditions. A refined model is written as

   $$I_\mathrm{j}(\mathbf{s}) = |\, \mathcal{P}_\mathrm{z}\{p(\mathbf{x})\, o(\mathbf{x} - \mathbf{x}_\mathrm{j})\} \,|^2 \tag{2.218}$$

   where $\mathcal{P}_\mathrm{z}$ is the operator that would propagate an electromagnetic wave from the source plane $\mathbf{x}$ to the detection plane $\mathbf{s}$ at a distance z. In the case of scalar diffraction theory, the angular spectrum represention is most general. It describes the propagation as a linear space-invariant filter (Goodman, 2005)

   $$\psi(\mathbf{s}) = \mathcal{P}_\mathrm{z}\{\psi(\mathbf{x})\} = \underset{\mathbf{v},\mathbf{s}}{\mathcal{F}^*} \left\{ \underset{\mathbf{x},\mathbf{v}}{\mathcal{F}} \{\psi(\mathbf{x})\} \cdot \exp\left(2\pi \mathrm{i}\, \mathrm{z} \sqrt{\lambda^{-2} - \mathbf{v}^2}\right) \right\} \tag{2.219}$$

   where $H(\mathbf{v}) = \exp(2\pi \mathrm{i}\, \mathrm{z} \sqrt{\lambda^{-2} - \mathbf{v}^2})$ is the associated transfer function for free-space propagation of a distance z for the wavelength λ. We discussed possible approximations to the angular spectrum propagator in sec. 2.1.2. Applicability of such a modification was proven for imaging regimes of high Fresnel numbers (Stockmar et al., 2013, 2015b).

2. **Sharing of diffraction data**
   As Dierolf (2015) discussed in great detail, it is sometimes advantageous to combine the reconstruction of two or more ptychographic scans if they share information. Prominent examples are a common probe for a series of scans in tomographic ptychography (Stockmar et al., 2015a) or a common object for measurements with and without beam stop or large segmented scans (Guizar-Sicairos et al., 2014). In general, the diffraction patterns I_j may be the result of a set of various combinations of probe and object entities (p_c and o_d). For example, the object index d may change, when the sample is rotated, and the probe index c may change because of optics or sample drifts. These changes may even occur within the same scan. We imply such changes with

the assignments $c = c(\mathrm{j})$ and $d = d(\mathrm{j})$ while the scan point index j spans over multiple scans. Additionally, the geometry may vary between scans or scan points (Robisch and Salditt, 2013) and, consequently, we assume $\mathrm{z} = \mathrm{z}(\mathrm{j})$. Hence, an improved model with respect to (2.218) can be written as

$$I_{\mathrm{j}}(\mathbf{s}) = \left|\mathcal{P}_{\mathrm{z(j)}}\left\{p_{c(\mathrm{j})}(\mathbf{x}) \cdot o_{d(\mathrm{j})}(\mathbf{x} - \mathbf{x}_j)\right\}\right|^2 . \tag{2.220}$$

Except for ambiguous cases we will omit the scan point index j in the expressions $d(\mathrm{j})$, $c(\mathrm{j})$ and $\mathrm{z}(\mathrm{j})$.

3. **State Mixtures**
It is rarely accurate to assume full coherence of the wave field along the imaging pathway. Apart from the obvious partial coherence in the source, effects like signal spread in the detector or sample vibrations also exhibit the same common signature of reduced speckle or fringe visibility. Ptychography can cope with all those experimental realities by including a mode decomposition for probe and/or object. This development has found immediate use and delivered high-quality reconstructions e.g. for setups with a partially coherent source (Thibault and Menzel, 2013) (also in combination with signal spread in the detector (Enders et al., 2014)), or for rapid sample movement (Clark et al., 2014; Pelz et al., 2014). According to (2.143), we extend the model in the following way:

$$I_{\mathrm{j}}(\mathbf{s}) = \sum_{m,n} \left|\mathcal{P}_{\mathrm{z}}\left\{p_{c,m}(\mathbf{x}) \cdot o_{d,n}(\mathbf{x} - \mathbf{x}_j)\right\}\right|^2 \tag{2.221}$$

Probe and object are composed of a finite number of coherent (pure) but mutually incoherent states. If the physical reality of the setup does not deviate far from a pure state, the number of modes needed for convergence will be small. We have found that in practice allowing for a minimum of 2 or 3 modes in the probe always helps to reduce incoherence-related artefacts in the object (Stockmar et al., 2013).

4. **Polychromatic illumination**
Traditionally, CDI experiments at synchrotrons facilities are carried out with monochromatic sources, which requires a high degree of spectral filtering. For a single undulator harmonic of bandwidth $\Lambda \approx 10^{-2}$ a common double crystal monochromator with a bandwidth of $\Lambda \approx 10^{-4}$ filters roughly 99% of the available photons. While it was already demonstrated that a monochromatic ptychographic model works almost unimpaired for bandwidths of up to 2% (Enders et al., 2014), higher bandwidths may require a model that takes the

polychromaticity of the source into account:

$$I_{\mathsf{j}}(\mathbf{s}) = \int \left|\mathcal{P}_{\mathsf{z}}(\lambda)\left\{p(\lambda,\mathbf{x})\cdot o(\lambda,\mathbf{x}-\mathbf{x}_j)\right\}\right|^2 \mathrm{d}\lambda \tag{2.222}$$

Alongside with the natural spectrum-dependent propagator, the object has a spectrally dependent response $o(\lambda,\mathbf{x})$ and the source spectrum can be included in the probe $p(\lambda,\mathbf{x})$. If the detector response is flat, eq. (2.222) thus represents the polychromatic model where all spectral contributions form an integral. If there is a small sensitivity threshold for energy / wavelength or the source spectrum is comb-like, a discrete sum is also appropriate. In combination with eq. (2.221), we arrive at

$$I_{\mathsf{j}}(\mathbf{s}) = \sum_{\lambda}\sum_{m,n}\left|\mathcal{P}_{\lambda,\mathsf{z}}\left\{p_{c,m,\lambda}(\mathbf{x})\cdot o_{d,n,\lambda}(\mathbf{x}-\mathbf{x}_j)\right\}\right|^2 \ . \tag{2.223}$$

5. **Masking of diffraction data**
   Some pixels in the detector can yield bad data or no data at all if they are either over- or under-responsive, part of a module gap or even blocked by a central beam stop. Hence, a mask m is needed to disregard invalid pixels in the reconstruction algorithms. In general, this mask can be different for each diffraction pattern, for instance when pixels accidentally overexpose. The model thus becomes,

$$\mathsf{m}_{\mathsf{j}}(\mathbf{s})\, I_{\mathsf{j}}(\mathbf{s}) = \mathsf{m}_{\mathsf{j}}(\mathbf{s}) \sum_{m,n}\sum_{\lambda}\left|\mathcal{P}_{\lambda,\mathsf{z}}\left\{p_{c,m,\lambda}(\mathbf{x})\cdot o_{d,n,\lambda}(\mathbf{x}-\mathbf{x}_j)\right\}\right|^2 \tag{2.224}$$

$$= \mathsf{m}_{\mathsf{j}}(\mathbf{s}) \sum_{\lambda,m,n}\left|\mathcal{P}_{\lambda,\mathsf{z}}\left\{\psi_{\mathsf{j},\lambda,m,n}(\mathbf{x})\right\}\right|^2 . \tag{2.225}$$

In this last step we have introduced the *elementary exit wave* $\psi_{\mathsf{j},\lambda,m,n}(\mathbf{x})$, which inherits all indices from p and o, and as such represents a coherent monochromatic wave after interacting with a single object mode.

Certainly, higher-dimensional forward scattering models entail complexer reconstruction engines as well. The next chapter presents a novel reconstruction framework designed to handle these models while still allowing to implement reconstruction engines in a concise, readable manner.

3. A computational framework for ptychography

Reconstruction techniques are still evolving at a rapid pace. A survey of the literature shows that several independent implementations of these algorithms exist, most of them including additional ad hoc capabilities depending on specific experimental conditions (Wilke et al., 2013; Robisch and Salditt, 2013; Zhang et al., 2013; Clark et al., 2014). Up to now only a few groups have made their code available to the wider scientific community:

- A set MATLAB scripts implementing the *Difference Map* and *ePIE* and developed at the cSAXS beamline of the Swiss Light Source in the years 2007-2009 (Dierolf et al., 2010a; Thibault, 2008; Dierolf et al., 2010b) is available for the users of the beamline.

- The Sharp Camera package (Sharp Camera Team, 2014) is developed by the *Center for Applied Mathematics for Energy Research Applications, CAMERA* and hosted at `http://www.camera.lbl.gov/`. It features parallel reconstruction on multiple graphics processor units (GPUs) in far-field geometry using RAAR as reconstruction engine (Shapiro et al., 2014). The package uses the proprietary CUDA parallel computing toolkit for nVIDIA GPUs.

- The software mentioned in (Nashed et al., 2014) was briefly available online but is currently unavailable (Deng et al., 2015a,b). It provides an implementation of *ePIE* and was implemented for parallel reconstructions on multiple GPUs again for nVIDIA's CUDA framework only.

The availability of stable reconstruction packages has clear benefits for the development of a technique:

- It allows to *compare* data of different setups, and to *validate* results after *disclosure* of data and reconstruction parameters.

- It furthers the *development of standards* which is helpful both for a *unified formalism* and *quality control*.

- It lowers the *entry barrier* for new groups resulting in an *expansion of user base* leading to a *wider acceptance* for the method.

Considering that many high-resolution nano-probe beamlines are planning to support ptychography as a standard, and that ptychographic algorithms have reached a sufficient level of maturity, it has become clear that the community needs broader access to state-of-the-art reconstruction software.

This chapter introduces and describes Ptypy, an open-source software framework for ptychography written in Python. An essential design feature of Ptypy is the clear separation between representations of physical experiments, the *models*, and the algorithmic solutions to the inverse scattering problem, the *engines*. Beyond basic design principles, this chapter provides an overview of the supported models and shows how the abstraction from the physical model is achieved. An in-depth walk-through of a basic implementation of *Difference Map* demonstrates how such abstraction and the associated programming interface of Ptypy results in concise, readable and intuitive implementations of reconstruction engines despite possible model complexity.

3.1. Design principles

It is apparent from (2.225) that developing reconstruction software for all models can be cumbersome. The contributions of various modes, probes, objects and propagators have to be considered in each algorithmic implementation either by design or when an upgrade of existing code becomes necessary. Building a software from a simple model to a more complex one may lead to cluttered, unreadable code of monolithic structure with hidden switches, global parameters etc. Such software is hard to maintain and debug and even harder to read for a novice.

Alleviating these programming difficulties was one of the main reasons to create Ptypy as open-source ptychography software in the Python programming language. Its development is guided by the following principles.

1. **Wide model support**
   A core design of Ptypy is the separation between the *models* and the *engines* in order to facilitate small and light-weight engine implementations that can be extended easily. As different experimental setups result in different models, Ptypy aims to support as many models as possible. Currently, Ptypy supports all models mentioned in Sec. 2.5 and is ready to be adapted for new developments.

2. **Wide algorithm support**
   While, in principle, any ptychographic algorithm is designed to reach a solution, the convergence properties of each the algorithm can vary (Wen et al., 2012). In practice, algorithms may converge at local minima (ePIE, ML) or explore the solution space without a clear stopping criterion (DM). With Ptypy, one can easily chain sequentially different engines to reach a specific objective – be it reaching the solution in the shortest computation time or pushing for the highest quality of the reconstruction.

3. **Modularity and documentation**
   Providing a modular object-oriented implementation helps to extend and to reuse parts of it. This is a crucial feature if the code is understood as a continually developing project. For contributors, Ptypy is made of roughly 11,000 lines of python code with 7,000 lines of comments or documentation strings and 4,000 are blank lines, ensuring readability and a high degree of in-line documentation. For the users, Ptypy is hosted online as open-source project at `http://ptycho.github.io/ptypy` and provides tutorials and explanation along with documentation of all its classes and helper functions. In addition to core algorithmic functionality Ptypy features a rich set of utilities from plotting to parallel processing to common mathematical operations.

4. **Speed**
   Naturally, the ideal implementation finds the global solution in a minimum time. For many algorithms, computation speed comes directly at the cost of flexible modular implementations. Fast implementations may require to build and compile monolithic kernels in a low level-language. It may imply to fix the ptychographic model and the engine algorithm to fit a specific problem while loosing wide applicability in combination with an added effort to write the code. As a compromise, Ptypy relies on *NumPy* (Van Der Walt et al., 2011; Oliphant, 2007) for its calculations and thus benefits from optimised C-code for most numerically-intensive operations. Accelerated engines can also be embedded in the same framework to keep the benefits of organisation, storage and plotting purposes. A GPU-based DM-engine is currently being developed.

5. **Wide end-user support**
   Preparing and organising the data prior to reconstruction marks a major part of the implementation. The absence of standards means that the data are in general organised differently for each end-user application. Up to now, no data format for ptychographic datasets is widely used, the hdf5-based (The HDF Group, 1997) CXI-database (Maia, 2012) format being the closest current

standard. Ptypy stores and loads data from hdf5 files with a custom internal tree structures. It also provides an abstract base class for loading and preparation from raw datasets, which can be easily adopted to fit to a new application or data format.

6. **Scalability, interactivity and online capacity**
As for any other scanning methods, ptychographic data are acquired as a stream. Depending on the instrument, the total acquisition time for a complete scan typically ranges from a few seconds to several minutes. Hence it may be useful for online feedback to initiate a reconstruction while data are streaming. The user should be able to oversee the reconstruction process in order to take immediate measures: abort the scan, adjust the sample position, etc. Although already achievable with current hardware, such dynamic ptychographic scans have not been used until now, mostly because of the complete absence of software to support these streams. Ptypy has been designed to provide such support: all internal storage containers can adapt to a data inflow, and the reconstruction process is ready to start as soon as the first diffraction pattern is recorded.
Ptypy is designed to run in parallel on many processing units, on a single computer or in a cluster. Larger data quantities may thus be analysed without compromise on processing time. The cluster of nodes running a reconstruction can be polled asynchronously and an interactive control of the reconstruction process following a client-server design is currently under development. Figure Fig. 3.1 gives a schematic of the implementation of distributed computing in Ptypy .

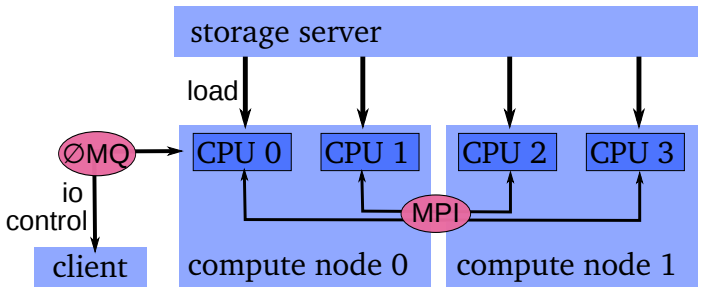


Figure 3.1. – Ptypy relies on the *Message Passaging Interface, MPI* (Dalcín et al., 2008) protocol for parallel computation. Specifically, Ptypy distributes memory intensive data buffers, e.g. for the diffraction data and for the exit waves. Data can be loaded from the storage server in parallel if needed. The master node (here "CPU 0") handles asynchronous communication with clients, (e.g. for plotting) running a server based on *ZeroMQ* (Granger and Ragan-Kelley, 2015; Hintjens, 2013) in a shared-memory thread.

A detailed description of the implementation of all these principles is beyond the

scope of this chapter. The following sections concentrate on the core design of Ptypy, corresponding to elements 1. and 2. in the above list of design principles. For this purpose, Ptypy introduces a small set of connected python objects which mediate between the physical world and its numerical representation, and organise the memory access for intuitive algorithm implementations.

Appendix B, or alternatively, the online documentation in the web page address elements 3., 5. and 6. of the above list.

3.2. Primer to Ptypy

This section describes Ptypy's internal representation of the ptychographic model and the purpose of the main python objects present in the framework. The description is illustrated with the implementation of a simple reconstruction engine.

3.2.1. Storage abstraction: the POD object

Any algorithmic implementation of ptychography starts with the problem of representing physical quantities as discrete memory buffers on the computer. The simplest solution is the use of array classes such as the *numpy* nd-array (Van Der Walt et al., 2011), that map physical quantities to multidimensional arrays, where each index represents one parametric dimension. For example, the array representing the probe may be 5-dimensional with axes $\{c, m, \lambda, x_{\mathrm{I}}, x_{\mathrm{II}}\}$, where $(x_{\mathrm{I}}, x_{\mathrm{II}})$ is a suitable sampling of the spatial coordinate $\mathbf{x}$. For the implementation of the model, the programmer has to find effective means to loop through these indices and to select or disregard the value of an index, depending on its physical meaning. One objective of Ptypy as a framework is to avoid such loops over memory buffer indices, as the same memory access pattern usually persists for all iterations of the engine. Instead, the engine is meant to loop over a single list of objects in which access rules for all memory buffers are stored.

The exit wave introduced in eq. (2.225) can be seen to be the most elementary element to loop over. In Ptypy the information associated with a given exit wave is stored in a light-weight object called `POD` (for **Probe-Object-Diffraction**). Each `pod` instance represents a unique combination of $(\mathrm{j}, \lambda, m, n)$, and thus stores access rules (e.g. memory addresses or slicing coefficients) to the corresponding memory buffers.

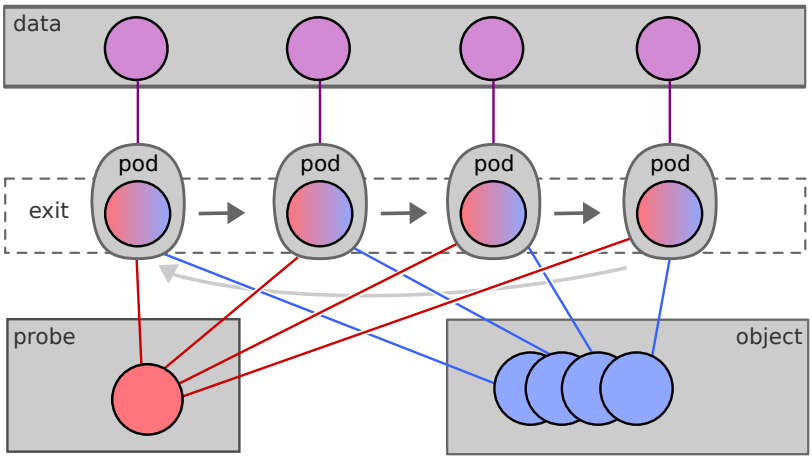


Figure 3.2. – Schematic of Ptypy's design principle. A ptychographic algorithm iterates over an assortment of objects called PODs, which are used as an abstraction layer between engine algorithms and the access to diffraction, probe and object array access.

In addition to the exit wave buffer `pod.exit`, a `pod` instance keeps attributes to access probe and object buffers through `pod.probe` and `pod.object`. It also carries access to diffraction data buffers (`pod.diff`) and the optional detector mask (`pod.mask`). The reference to the forward and backward propagator is held in two methods, `pod.fw()` and `pod.bw()` respectively.

POD attribute	field representative
`pod.probe`	$p_{c(\mathrm{j}),m,\lambda}(\mathbf{x})$
`pod.object`	$o_{d(\mathrm{j}),n,\lambda}(\mathbf{x}-\mathbf{y}_j)$
`pod.exit`	$\psi_{\mathrm{j},\lambda,m,n}(\mathbf{x})$
`pod.diff`	$I_{\mathrm{j}}(\mathbf{s})$
`pod.mask`	$\mathsf{m}_{\mathrm{j}}(\mathbf{s})$
`pod.fw`	$\mathcal{P}_{\lambda,\mathrm{z(j)}}$
`pod.bw`	$\hat{\mathcal{P}}_{\lambda,\mathrm{z(j)}}$

In essence, a `pod` is equipped with everything to perform *one coherent propagation* of the ptychographic model as illustrated by Fig. 3.3. By design, a collection of `pods` alone suffices to calculate the forward model. The "Fourier modulus constraint", imposed by the measured diffraction data, can thus be implemented using only `pod` instances as we can see in the following example.

Example: Fourier update in Difference Map

We recall steps 2.-4. ((2.201) to (2.203)) from Sec. 2.4.5 and the Fourier projector

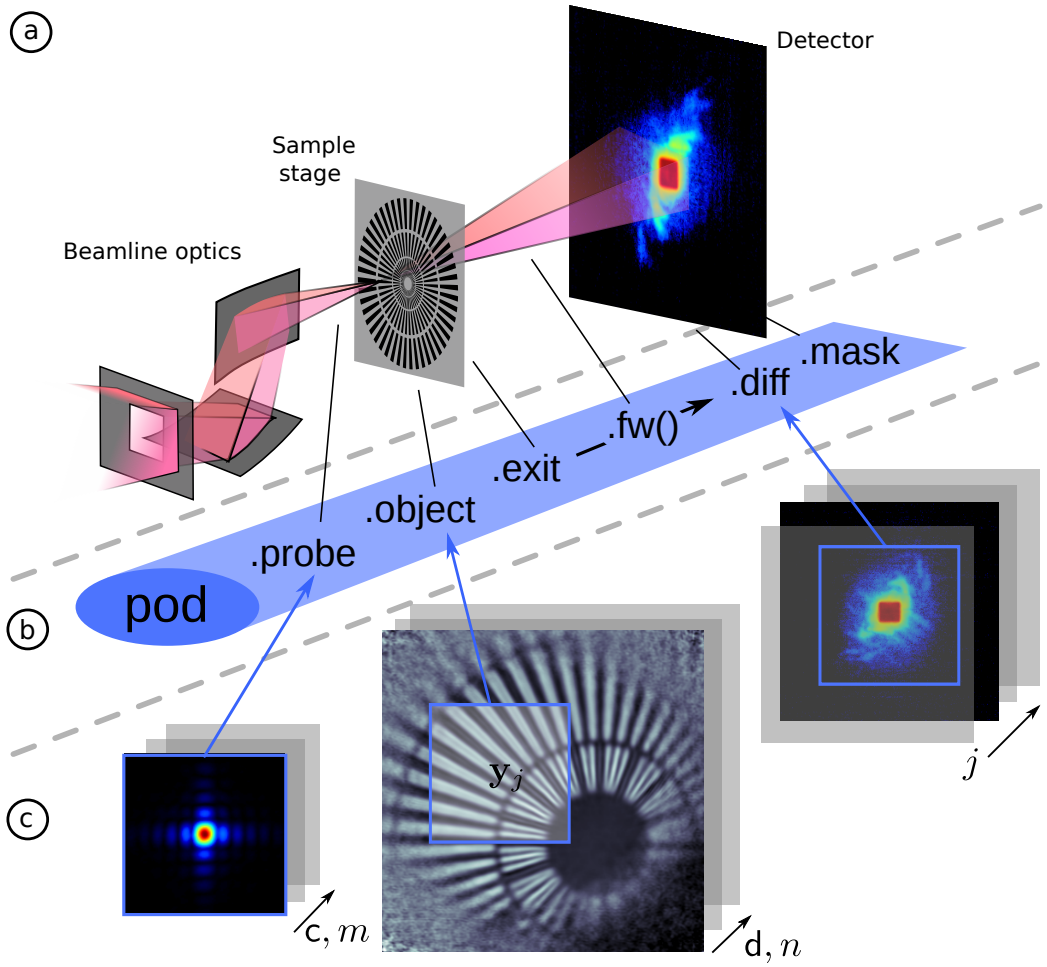


Figure 3.3. – Main attributes of the pod object. **a** Elements of a typical diffraction setup. **b** Corresponding attribute accesses of the Python object along the path of propagation. **c** Numerical arrays storing the data (`Storages`). `View` access to the data is indicated by blue squares of equal extent.

from Sec. 2.2.3. Given that probe and object have been recently updated, the auxiliary functions are build according to

$$\upsilon_{\mathrm{j}}^{\mathrm{it}}(\mathbf{x}) = 2\,p^{\mathrm{it}}(\mathbf{x})\,o^{\mathrm{it}}(\mathbf{x}-\mathbf{x}_{\mathrm{j}}) - \psi_{\mathrm{j}}^{\mathrm{it}}(\mathbf{x}) \tag{3.1}$$

Under coherent conditions and for far-field diffraction, the Fourier modulus projection on each auxiliary wave field is written as,

$$\upsilon_{\mathrm{j}}^{\mathrm{it}+1}(\mathbf{x}) = \mathcal{F}^{*}_{\mathbf{q}}\left\{\mathcal{F}_{\mathbf{x}}\left\{\upsilon_{\mathrm{j}}^{\mathrm{it}}(\mathbf{x})\right\}\frac{\sqrt{\mathsf{M}_{\mathrm{j}}(\mathbf{q})}}{\left|\mathcal{F}_{\mathbf{x}}\left\{\upsilon_{\mathrm{j}}^{\mathrm{it}}(\mathbf{x})\right\}\right|}\right\} = \mathcal{F}^{*}_{\mathbf{q}}\left\{\mathcal{F}\upsilon^{\mathrm{it}}(\mathbf{q})\cdot\Upsilon(\mathbf{q})\right\}, \tag{3.2}$$

where $\Upsilon(\mathbf{q})$ is the correction factor for the Fourier amplitude to comply with the

measured intensity data. The updated exit waves are constructed from

$$\psi_{\mathsf{j}}^{\mathsf{it}+1}(\mathbf{x}) = \psi_{\mathsf{j}}^{\mathsf{it}}(\mathbf{x}) + \upsilon_{\mathsf{j}}^{\mathsf{it}+1}(\mathbf{x}) - p^{\mathsf{it}}(\mathbf{x})\, o^{\mathsf{it}}(\mathbf{x} - \mathbf{x}_{\mathsf{j}}) \tag{3.3}$$

For the most complex model (2.220) the input to the Fourier modulus constraint becomes

$$\upsilon_{\mathsf{j},m,n,\lambda}^{\mathsf{it}}(\mathbf{x}) = 2\, p_{c,m,\lambda}^{\mathsf{it}}(\mathbf{x}) \cdot o_{d,n,\lambda}^{\mathsf{it}}(\mathbf{x} - \mathbf{x}_{\mathsf{j}}) - \psi_{\mathsf{j},m,n,\lambda}^{\mathsf{it}}(\mathbf{x}) \tag{3.4}$$

Now, we can formulate the "Fourier update" in a similar fashion to Thibault and Guizar-Sicairos (2012) where we sum over all (coherent) exit waves that contribute to the same (partially coherent) signal in order to find the correction factor,

$$\Upsilon_{\mathsf{j}}(\mathbf{s}) = 1 - \mathsf{m}_{\mathsf{j}}(\mathbf{s}) + \frac{\mathsf{m}_{\mathsf{j}}(\mathbf{s})\sqrt{\mathsf{M}_{\mathsf{j}}(\mathbf{s})}}{\sqrt{\sum_{\tilde{\lambda},\tilde{m},\tilde{n}} \left|\mathcal{P}_{\lambda,\mathsf{z}}\left\{\upsilon_{\mathsf{j},\tilde{\lambda},\tilde{m},\tilde{n}}^{\mathsf{it}}(\mathbf{x})\right\}\right|^2}} \tag{3.5}$$

Finally, we replace the Fourier transform with an arbitrary propagator (2.25).

$$\upsilon_{\mathsf{j},m,n,\lambda}^{\mathsf{it}+1}(\mathbf{x}) = \hat{\mathcal{P}}_{\lambda,\mathsf{z}}\left\{\mathcal{P}_{\lambda,\mathsf{z}}\left\{\upsilon_{\mathsf{j},m,n,\lambda}^{\mathsf{it}}(\mathbf{x})\right\} \cdot \Upsilon(\mathbf{s})\right\} \tag{3.6}$$

$$\psi_{\mathsf{j},m,n,\lambda}^{\mathsf{it}+1}(\mathbf{x}) = \psi_{\mathsf{j},m,n,\lambda}^{\mathsf{it}}(\mathbf{x}) + \upsilon_{\mathsf{j},m,n,\lambda}^{\mathsf{it}+1}(\mathbf{x}) - p_{\mathsf{j},m,\lambda}^{\mathsf{it}}(\mathbf{x}) \cdot o_{\mathsf{j},n,\lambda}^{\mathsf{it}}(\mathbf{x} - \mathbf{x}_{\mathsf{j}}) \tag{3.7}$$

In the following we show a possible implementation of the "Fourier" modulus constraint using the `Pod` class of Ptypy. The `fourier_update` takes a dictionary of `pod` instances as input that belong to the same scan point index (j). All these `pod`s represent the mixed states and spectral compositions of the experiment. The associated exit waves $\psi_{\mathsf{j},\lambda,m,n}^{\mathsf{it}}$ are updated simultaneously as the feedback factor $\Upsilon(\mathbf{s})$ from the Fourier update is the same for one diffraction pattern.

```
def fourier_update(pods):
    pod = pods.values()[0]
    # Get Magnitude and Mask.
    mask = pod.mask
    modulus = np.sqrt(np.abs(pod.diff))
    # Create temporary buffers.
    Imodel = np.zeros_like(pod.diff)
    err = 0.
    Dphi = {}
    # Propagate the exit waves.
    for gamma, pod in pods.iteritems():
        Dphi[gamma] = pod.fw(2.*pod.probe*pod.object - 1.*pod.exit)
        Imodel += Dphi[gamma] * Dphi[gamma].conj()
    # Calculate common correction factor.
    factor = (1-mask) + mask * modulus / (np.sqrt(Imodel) + 1e-10)
```

```
16      # Apply correction and propagate back.
17      for gamma, pod in pods.iteritems():
18          df = pod.bw(factor*Dphi[gamma]) - pod.probe*pod.object
19          pod.exit += df
20          err += np.mean(np.abs(df*df.conj()))
21      # Return difference map error on exit waves.
22      return err
```

3.2.2. Data constructs and access

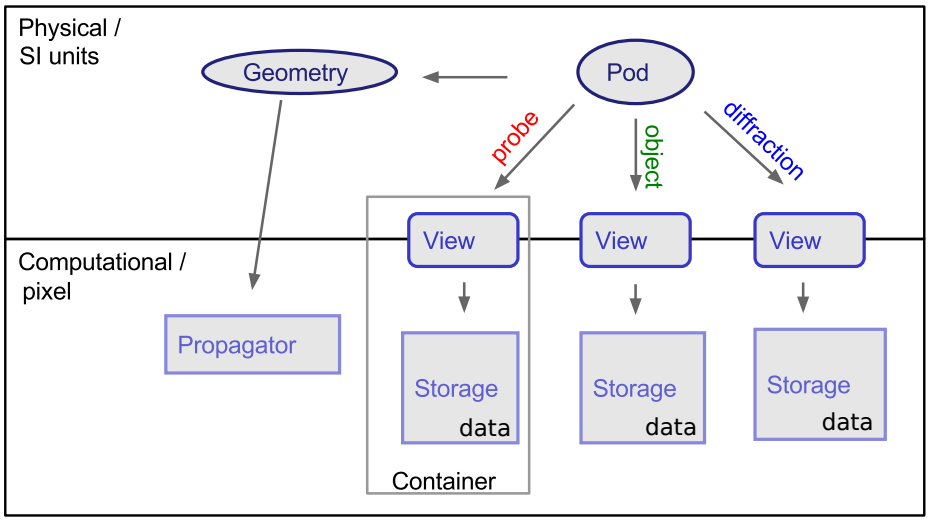


Figure 3.4. – Overview of the most important classes in Ptypy and how they relate to each other. A `Pod` is the highest level object. In order to be able to create the forward model, it contains a (data)-`View` for each entity (three are shown) and a reference to the scan's geometry in order to use the appropriate propagator. A `View` mediates between "physical" wave fields and "numerical" data buffers, and, when applied to a Storage, it yields the two dimensional data representing the region of interest of the view. A `Container` manages views and storages and there is one `Container` for each entity. The Geometry class contains the scan geometry, i.e. distance from detector to object and pixel size, resolution, etc. It provides the numerical propagator for coherent forward and backward propagation.

The `Pod` class is convenient when accessing data from the buffer arrays but these arrays need to be created, managed and contained by other objects. Ptypy provides three other basic classes for data storage and access. For a compact overview see Fig. 3.4.

View In numerical computations, we cannot work with infinitely extended wave-fields, but rather with compact rectangular regions. A `View` instance stands for such a region and is characterised by its extent and the physical position of its center

relative to the wave field's coordinate origin. In the course of computation, the `View` also caches the numerical representations of its physical counterparts. A `Pod` therefore has a view for each of the five entities *probe*, *object*, *exit* wave, *diff*raction data and *mask*. The number of views scales with the number of scan points.

Storage A `Storage` instance is a numerical array associated with a physical coordinate system. It represents a set of wave fields that share the same coordinate system. It can adapt its internal array size dynamically depending on those parts of the wave field that are requested by `view`s. The number of storages scales roughly with the number of scans that are being organized by Ptypy. Applying a `View` to a `Storage` yields the two-dimensional array which is sliced from the storages' data array and represents the numerical data for the view's physical extent. `Storage`[`View`] = `data` (2d)

Container A `Container` instance holds all `view`s and `storage`s that belong to the same entity in ptychography. In contrast to Views and Storages, there are only 5 base Containers in Ptypy, one for each entity. The number of Storages depends on the number of scans and their sharing behaviour. Applying a view to its container is the same as applying the view to its associated storage in the container. `Container`[`View`] = `data` (2d)

A container is also capable of creating copies of itself or perform basic mathematics which it will relay to all its Storages' internal data buffers. The ability of the Container to clone itself is important for algorithms to provide temporary data buffers that behave like one of the entities. In addition to basic in-place math operations, the `Container` is also capable to communicate its data among processes in the case where the same ptypy reconstruction runs in parallel on many nodes, (see Fig. 3.1.

Example: Overlap Projection in Difference Map
In its simplest form (Thibault et al., 2009), probe and overlap update can be written as

$$o^{\mathsf{it}}(\mathbf{x}) = \frac{\sum_{\mathsf{j}}[p^{\mathsf{it}-1}(\mathbf{x}+\mathbf{x}_{\mathsf{j}})]^* \cdot \psi^{\mathsf{it}}(\mathbf{x}+\mathbf{x}_{\mathsf{j}})}{\sum_{\mathsf{j}} |p^{\mathsf{it}-1}(\mathbf{x}+\mathbf{x}_{\mathsf{j}})|^2} \tag{3.8}$$

$$p^{\mathsf{it}}(\mathbf{x}) = \frac{\sum_{\mathsf{j}}[o^{\mathsf{it}-1}(\mathbf{x}-\mathbf{x}_j)]^* \cdot \psi^{\mathsf{it}}(\mathbf{x})}{\sum_j |o^{\mathsf{it}-1}(\mathbf{x}-\mathbf{x}_j)|^2} \tag{3.9}$$

If we include the model mentioned in 2.5, we have to sum also over shared data and other modes of probe and object. For one of the objects o_d we have to restrict the

update to those scan point indices i that contribute to that object, i.e. $\mathsf{i} \in \{\mathsf{j} \,|\, d(\mathsf{j}) = d\}$:

$$o^{\mathsf{it}}_{d,n,\lambda}(\mathbf{x}) = \frac{\sum_{\mathsf{i},m} \left[p^{\mathsf{it}-1}_{c(\mathsf{i}),m,\lambda}(\mathbf{x}+\mathbf{x}_\mathsf{i})\right]^* \cdot \psi^{\mathsf{it}}_{\mathsf{i},m,n,\lambda}(\mathbf{x}+\mathbf{x}_\mathsf{i})}{\sum_{\mathsf{i},m} \left|p^{\mathsf{it}-1}_{c(\mathsf{i}),m,\lambda}(\mathbf{x}+\mathbf{x}_\mathsf{i})\right|^2} \tag{3.10}$$

The same holds true for the probes. For probe c we pick the indices $\mathsf{i} \in \{\mathsf{j} \,|\, c(\mathsf{j}) = c\}$ and arrive at a similar expression:

$$p^{\mathsf{it}}_{c,m,\lambda}(\mathbf{x}) = \frac{\sum_{\mathsf{i},n} \left[o^{\mathsf{it}-1}_{d(\mathsf{i}),n,\lambda}(\mathbf{x}-\mathbf{x}_\mathsf{i})\right]^* \cdot \psi^{\mathsf{it}}_{\mathsf{i},m,n,\lambda}(\mathbf{x})}{\sum_{\mathsf{i},n} \left|o^{\mathsf{it}-1}_{d(\mathsf{i}),n,\lambda}(\mathbf{x}-\mathbf{x}_\mathsf{i})\right|^2} \tag{3.11}$$

Other than the Fourier update (code 3.1), probe and object update require a copy of the same kind as probe and object in order to calculate the denominator in the equations above. Despite the complexity in (3.10) and (3.11), the algorithmic implementation in Ptypy resembles very much (3.8) and (3.9) regarding its simplicity.

```
24 def probe_update(probe, norm, pods, fill=0.):
25     probe *= fill
26     norm << fill + 1e-10
27     for name, pod in pods.iteritems():
28         if not pod.active: continue
29         probe[pod.pr_view] += pod.object.conj() * pod.exit
30         norm[pod.pr_view] += pod.object * pod.object.conj()
31     probe.allreduce()    # MPI call
32     norm.allreduce()     # MPI call
33     probe /= norm
```

Code 3.2 – Basic DM probe update in Ptypy's syntax.

In addition to the `probe` container, the update functions require a container of similar shape which is called `norm` here. Access to the data of that container happens via the probe-*View* as `norm` was initially a copy of `probe`. As announced before, we note the in-place operation "`*=`","`«`" and "`/=`" in line 25,26 and 88, respectively, and the parallel reduction calls in line 86 and 87 which perform a sum (if no argument is given).
The object update is of similar kind and benefits in its simplicity additionally from the implicit shift of the object views.

```
35 def object_update(obj, norm, pods, fill=0.):
36     obj *= fill
37     norm << fill + 1e-10
38     for pod in pods.itervalues():
39         if not pod.active: continue
```

```
40          pod.object += pod.probe.conj() * pod.exit
41          norm[pod.ob_view] += pod.probe * pod.probe.conj()
42      obj.allreduce()       # MPI call
43      norm.allreduce()      # MPI call
44      obj /= norm
```

Code 3.3 – Basic DM object update in Ptypy's syntax.

We observe, that although the model can be very complex, the algorithmic core stays compact and simple. There is "boilerplate code" from nested loops and no indexing of memory buffers. The functions are also compatible to parallel execution with MPI.

3.2.3. A cross-referenced network

All instances of the basic classes defined in Ptypy are accessed through multiple cross-references. The highest-level entry point is the object `ptycho` which gives direct access to all pods (`ptycho.pods`), the five `Container` instances for the five entities (`ptycho.probe`, `ptycho.obj`, etc) and the scan geometries. Other objects are accessed through a hierarchy of attributes. For instance, `ptycho.diff.views` provides access to all diffraction data frames, while `view.owner` refers to the `container` that holds the view.

Example: A simple Difference Map algorithm
Provided with a `Ptycho` instance and the functions mentioned before, we are able to write a simple reconstruction engine.

```
46 def iterate(Ptycho, num):
47     # generate container copies
48     obj_norm = P.obj.copy(fill=0.)
49     probe_norm = P.probe.copy(fill=0.)
50     errors = []
51     for i in range(num):
52         err = 0
53         # fourier update
54         for di_view in Ptycho.diff.V.itervalues():
55             if not di_view.active: continue
56             err += fourier_update(di_view.pods)
57         # object update
58         object_update(Ptycho.obj, obj_norm, Ptycho.pods,0.1)
59         # probe update
60         probe_update(Ptycho.probe, probe_norm, Ptycho.pods,0.0)
61         # print error
62         errors.append(err)
63         if i % 3==0: print i,err
```

```
64
65      # cleanup
66      P.obj.delete_copy([obj_norm.ID])
67      P.probe.delete_copy([probe_norm.ID])
68      #return error
69      return errors, obj_avg
```

Code 3.4 – Basic DM engine in Ptypy's syntax.

Certainly, the example engine `iterate` requires a Python object instance `ptycho` to actually work. Depending on an extensive input parameter tree, Ptypy provides that `ptycho` instance along with the `pod`s and initialised `Storages` for probe and object. More information on this initialisation procedure can be found in the source code or the tutorials provided in the online documentation.

3.3. Examples

3.3.1. Visible light ptychography of a standard resolution target

As a first experimental demonstration of the package, we present a reconstruction from a simple laser diffraction experiment using a setup similar to one previously reported (Bunk et al., 2008). In this experiment, an aperture and a sample are placed along the optical path between a LED laser (650 nm wavelength) and a CCD. The aperture, a hole pierced in a cardboard piece with a fine needle, produces at the sample plane 11.5 mm downstream an illumination with high angular diversity, as shown by its reconstruction in Fig. 3.5a. The back-propagated illumination at the aperture plane is shown in Fig. 3.5b and a diameter of roughly 300 µm can be observed.

The detector, a Fingerlakes P1001 monochrome CCD camera is placed 145mm behind the sample plane. An additional neutral density filter before the camera is used to match the laser intensity with the minimal CCD exposure time to avoid overexposure. The diffraction images are binned by 3×3, creating effective detector pixels of $72\,\mu m \times 72\,\mu m$ area. 201 diffraction images were acquired on a non-periodic grid in a square area of 2 mm side length. An average photon count of roughly 1.18×10^7 per diffraction pattern, results in about 10^4 photons per reconstruction pixel. After acquisition, the diffraction data was processed into Ptypy's hdf data format as described in the documentation.
The ptychographic reconstruction of a USAF optical resolution target is shown in Fig. 3.5c. This reconstruction employed all the code snippets 3.1–3.4 presented in the

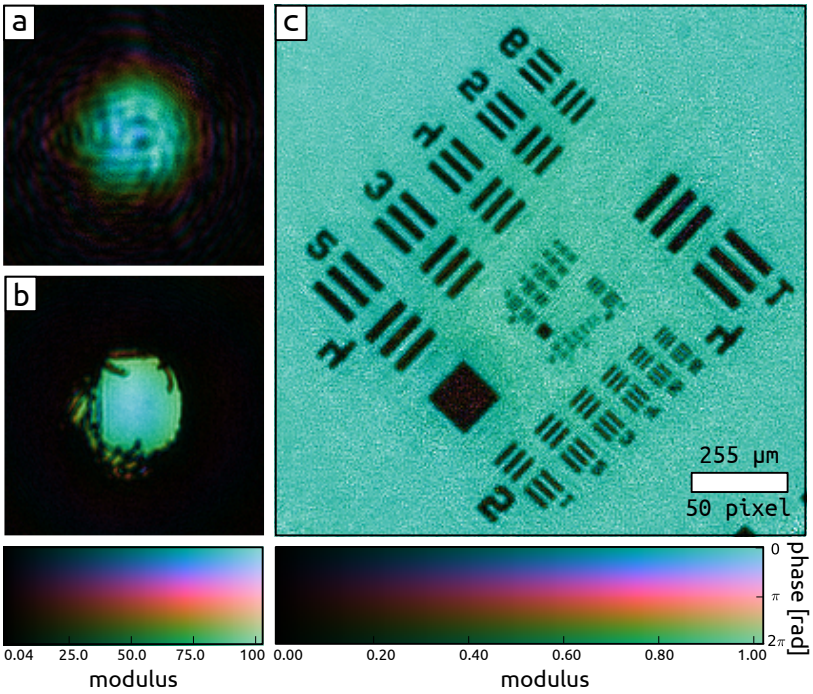


Figure 3.5. – Ptychography experiment with visible light in a laboratory. Complex values are color-coded such that the phase maps to hue and the modulus maps to the luminance of the image. **a** Recovered complex wave-front of the illumination in the sample showing a high degree of diversity. **b** Illumination in the aperture plane and was obtained from a numerical propagation of **a**. **c** Image of the recovered object transmission function of the sample. The noise in the reconstruction is caused by a low signal-to-noise ratio at high angular frequencies which, in turn, is a result of the isotropic readout noise of the CCD.

previous section. The remaining part of the reconstruction script is given below in code snippet 3.5.

```
71 ## Reconstruction script starts here ##
72 import numpy as np
73 from ptypy import utils as u
74 from ptypy.core import Ptycho
75
76 # Create input parameter tree.
77 p = u.Param()
78 p.verbose_level = 3
79 p.data_type = "single"
80 p.scan = u.Param()
81 p.scans = u.Param()
82
83 # Describe data source.
84 p.scans.usaf = u.Param()
85 p.scans.usaf.data = u.Param()
86 p.scans.usaf.data.source = 'file'
```

```
p.scans.usaf.data.dfile = 'test_pattern_roi.ptyd' # same directory

# Describe initial guess for illumination
p.scans.MF.illumination = u.Param()
p.scans.MF.illumination.aperture = u.Param()
p.scans.MF.illumination.aperture.size = 300e-6

# Pass input parameters to Ptycho instance, level=2 will create the pods.
P = Ptycho(p, level=2)

# Run reconstruction.
errors = iterate(P,120)

# Save reconstruction.
P.save_run('test_pattern_DM%d.ptyr' %i)
```

Code 3.5 – Python script to reconstruct the visible light dataset (see Fig. 3.5).

The noise visible in the object reconstruction originates from the absence of any corrective measures in the algorithm such as a Fourier relaxation threshold (Giewekemeyer et al., 2010) or gradient regularization (Thibault and Guizar-Sicairos, 2012). However, noise correction was left out on purpose to not misdirect the reader from the core principles by filling the presented code with non-essential instructions. The full capabilities of the mature reconstruction engines are demonstrated in the next section.

3.3.2. X-ray ptychography of a zone plate at a synchrotron facility

This data was originally published in (Thibault, 2008) and we refer to that publication for specifics. In brief, a ptychographic dataset of a zone plate was acquired on a rectangular raster grid pattern with 100 nm step size at a photon energy of 6.8 keV. The probe was a cone beam formed from a pinhole aperture in front of a focussing zone plate optic. A photon counting detector of 172 µm pixel size recorded diffraction images 2.19 m downstream behind the zone plate specimen.

Here, a 7 µm × 7 µm region of the original data is reconstructed with Ptypy using 300 iterations of DM with a subsequent ML refinement of 300 iterations. Five probe modes are allowed to account for possible decoherence effects.

We notice that the object transmission can be recovered in fine detail as shown by Fig. 3.6a and Fig. 3.6b. More specifically, the reconstruction quality is higher than the original as emphasized by panels i–n which compare selected regions with the original

and the new reconstruction. It is a surprising result as only 25 % (i.e 35×35 of 70×70 scan points) of the available diffraction patterns are used in this reconstruction. One reason that may have prevented the original algorithm to resolve the specimen with fine detail can be found in the mode decomposition (Fig. 3.6c–e) of the probe: A significant amount of energy (35 %) is accumulated in secondary modes indicating that the illumination onto the pinhole was not entirely coherent.

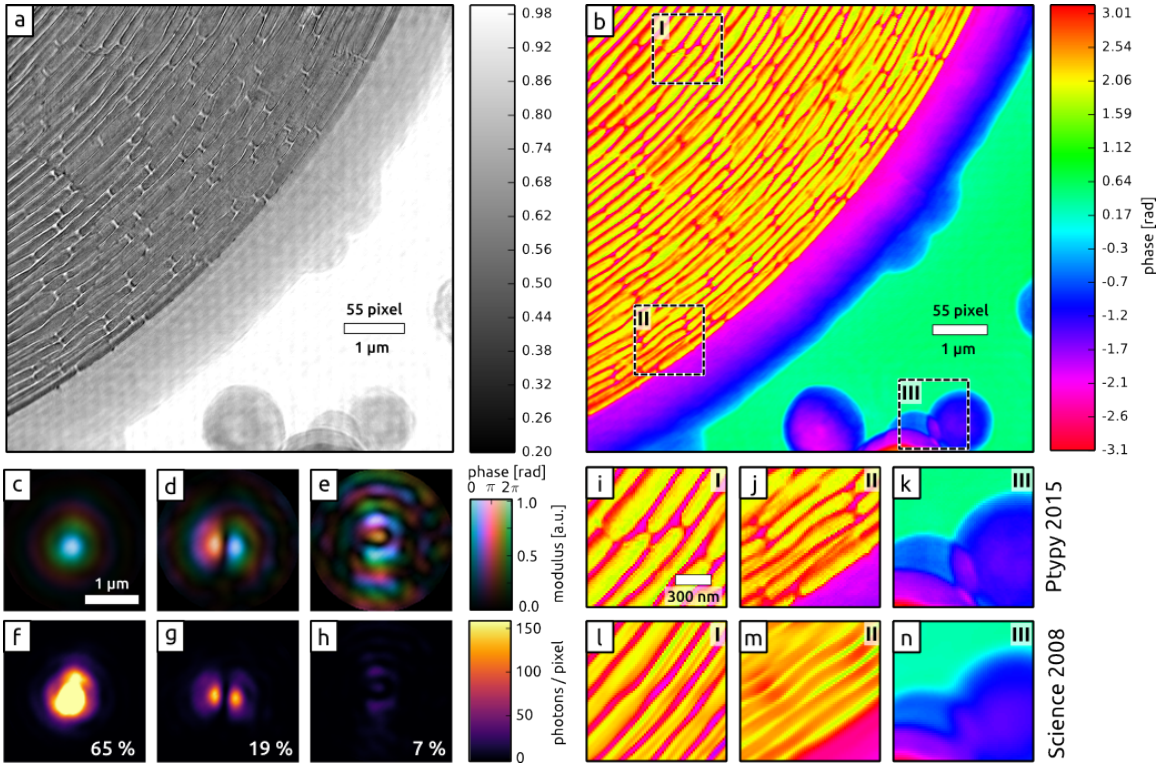


Figure 3.6. – X-ray ptychography at a small angle scattering beamline of a synchrotron. **a**, **b** Recovered modulus and phase of the outer region of a zone plate, serving here as a sample. **c** – **e** Wave fields of three recovered probe modes that represent 91 % the illumination in the sample plane. The probe modes were orthogonalized with the main mode being the mode to the left (**c**). **f** – **h** Same probe modes but displayed is their intensity distribution on linear scale. Please note that the main mode reaches values up to 700 photons in its center. The relative power of the mode is inscribed in lower right corner.**i** – **k** Selected regions of the recovered phase **b** displayed alongside with the same regions (**l** – **n**) from the original phase reconstruction (Thibault, 2008) of the sample. An improvement in reconstruction quality is apparent: Small bridges for stabilizing the ring structure are visible in **i** but not in **l**. Deviations of the outer zones from a circular path can be observed in **j** but can only be guessed from **m**.

3.4. Conclusion

This chapter presented how many different models of ptychography may be unified into a single model. A small set of abstract Python classes can be used to capture this model in a cross-referencing network. The references (API) exposed by the network were used to demonstrate how algorithms may be written in a way that is agnostic of the underlying physical geometry – an important trait for decoupling the algorithmic implementation from experimental specifications. We notice that many of today's applications are included in the formalism presented here. Other future problems of ptychography may be rephrased in a similar manner to extend the scope of this framework,

The software, Ptypy, is available for academic research in the X-ray regime and for numerical simulations. The documentation to all functions and parameters, as well as tutorials and further details about data management, are available on

`https://ptycho.github.io/ptypy`.

An excerpt of the documentation is included in appendix B.1. The official module reference is included in B.2.

Today, Ptypy features parallel computation of distributed data, an abstract loading class to accommodate different instruments and storage environments, and also a defined structure for raw and reconstructed data. As a framework, it exposes all its functions and classes to the user for free adaptation and provides a rich set of utility functions of practical use for many aspects of ptychography. The future roadmap includes GPU accelerated algorithms, on-the-fly adaptation and control of engine parameters as well as additional plug-ins for beamlines or other instruments that wish to use Ptypy for ptychographic reconstructions.

4. Ptychography with partial coherence

While in principle CDI promises diffraction-limited resolutions, the effective limit is set by the maximum solid angle that still carries sufficient photon signal. For most natural samples, it is well known that resolution d^{-1} scales like the fourth root of incident photon density Φ (Howells et al., 2009),

$$d^{-1} \propto \Phi^{\frac{1}{4}}\,, \tag{4.1}$$

thus making higher fluxes very desirable to push the resolving power of CDI techniques or speed up the acquisition. This need for higher flux to reach higher resolutions conflicts with the requirement to strongly filter the incident beam to satisfy the tight coherence prerequisites of CDI.
Section 4.1 demonstrates that ptychography can be conducted in conditions that were up to now considered insufficient, using a broad-bandwidth X-ray beam and an integrating scintillator-based detector.

Sometimes, highest resolution is not the main objective of a diffraction experiment but rather the amount or volume of specimen measured. Having a high-flux beam allows for shorter exposures with the same photon count speeding up the scanning procedure.[1] However, a shorter acquisition time has little impact if the settling time of the scanning piezo stages remains the bottleneck. Pushing for higher speeds may then introduce vibrations or positioning errors. Ultimately, scanning stages never halt and data is acquired on-the-fly (Deng et al., 2015a; Pelz et al., 2014). In cases where the vibrations are isotropic or the axis of continuous acquisition is the same throughout the scanned field of view, the resulting partial coherence can be modelled with a convolution of the shift coordinate with the offset density function ρ (ODF, see eq.(2.174)):

$$I(\mathbf{y},\mathbf{q}) = \rho(\mathbf{y}) \otimes_{\mathbf{y}} I_{\mathrm{coh}}(\mathbf{y},\mathbf{q})\,. \tag{4.2}$$

Section 4.2 is dedicated to a synchrotron experiment where decoherence was introduced by an oscillatory movement of the specimen stage. Probe and object mode

[1] Under the premise that the detector dead time is not the limit

reconstructions are compared and a general procedure to deconvolve the ODF from the mode decomposition is presented.

Strong stage oscillations can induce secondary in-plane oscillations in non-rigid specimen. These oscillations present a unique use-case for the mode expansion of the object transmission. However, at the small length scales of the experiment in Sec. 4.2, all tested specimen proved to be too rigid to exhibit secondary oscillations. Therefore, as a proof of principle, Section 4.3 takes a different approach and discusses a visible-light ptychography experiment where an air stream induces a random in-plane oscillations in a specimen.

4.1. Decoherence from high-flux conditions

Carrying out high-flux CDI experiments remains challenging, even for radiation-hard specimens which are not limited by strong dose restrictions. The main difficulties stem from the overall degradation of experimental conditions occurring when higher photon densities are brought onto a sample. First, increasing the flux in a synchrotron end station typically means reducing the amount of spatial and spectral filtering, thus deteriorating the coherence properties of the incident beam. Second, higher intensity beams produce proportionally stronger parasitic scattering from various optical elements in their path, contributing to an unwanted background. And third, reaching higher fluxes typically forces the transition from counting to integrating detectors, which often have less favourable properties such as non-linear signal conversion, inhomogeneous sensitivity or a large point-spread function (Ponchut, 2006). Under these conditions, the need for "clean" diffraction data for image reconstruction is rarely met.

Recalling eq. (2.183), the coherent model for far-field ptychography can be written as

$$I_{\mathrm{j,coh}}(\mathbf{q}) = \left| \underset{\mathbf{x},\mathbf{q}}{\mathcal{F}} \{p(\mathbf{x}) o(\mathbf{x} - \mathbf{y}_\mathrm{j})\} \right|^2 . \tag{4.3}$$

It represents ideal experimental conditions resulting in high-contrast far-field speckle patterns. In reality, background noise and spurious scattering add a smooth offset to the signal. Additionally, speckle visibility can be decreased substantially by various factors including partial transverse and longitudinal coherence of the illumination, motion of or within the sample, and detector point-spread. This latter case produces blurred diffraction patterns I_j which can be expressed by a convolution of the intensity signal with a point spread function,

$$I_\mathrm{j}(\mathbf{q}) = \mathrm{psf}(\mathbf{q}) \otimes I_{\mathrm{j,coh}}(\mathbf{q}). \tag{4.4}$$

We have seen this kind of convolution already in Sec. 2.3.8, where a blurring kernel $\mathcal{F}^{*}\tilde{\gamma}^{*}(\mathbf{q})$ is one possible manifestation of partial transverse coherence in the illumination. Partial coherence and cross-talk in the detector may thus cause the same deteriorating effect. Regardless of the actual cause, such blurring may leave algorithms based on an entirely coherent model struggle for convergence as eq. (4.3) no longer models accurately the recorded diffraction data.

It is possible to modify existing iterative algorithms by introducing a blurring kernel in a blind deconvolution step (Clark and Peele, 2011). As detailed in Sec. 2.3.4, a more general approach consists in generalizing eq. (4.3) to allow for multiple probe modes, p_m, requiring only the incoherent sum of their contributions to be compliant with the recorded diffraction data,

$$I_{\mathrm{j}}(\mathbf{q}) = \sum_m \left| \underset{\mathbf{x},\mathbf{q}}{\mathcal{F}} \{p_m(\mathbf{x}) o(\mathbf{x} - \mathbf{y}_{\mathrm{j}})\} \right|^2 . \tag{4.5}$$

Thibault and Menzel (2013) showed that the redundancy in a ptychographic dataset can be made high enough to solve for this mixture of modes without additional a-priori knowledge. The application of eq. (4.5) to a system with large detector point-spread motivates its general use for ptychographic imaging at setups where detector deficiencies or other similar effects reduce the diffraction image quality.

4.1.1. Experiment at ID22NI, ESRF

An experiment was performed at the nano-imaging endstation of the ID22 beamline (Martínez-Criado et al., 2012) at the European Synchrotron Radiation Facility (ESRF) in Grenoble, France, to demonstrate the robustness of ptychography in a configuration that allows for high-flux experiments.

The setup is sketched in Fig. 4.1(a). A pair of crossed multilayer-coated Kirkpatrick-Baez (KB) mirrors (Kirkpatrick and Baez, 1948; Hignette et al., 2005) focuses a portion of the X-ray beam attenuated by 3 mm of aluminum and selected by a pair of slits (35 µm (H) × 70 µm (V)) to a focal spot of about 150 nm (H) × 170 nm (V) (FWHM) with an energy bandwidth of $\Delta E/E \simeq 1.5 \times 10^{-2}$ (i.e $\ell_{\mathrm{coh}} \approx 3.2$ nm, see eq. (2.90)) at a peak energy of 16.95 keV.

The diffracted x-rays are recorded about 2.89 m downstream from the sample with a FreLoN 4320 T detector, which consists of a Gadolinium oxide scintillator coupled by tapered fiber-optics to a charge-coupled device (CCD) resulting in 51 µm effective pixel side length. A dark frame from the CCD is subtracted from the raw diffraction images before they are processed by the reconstruction algorithm.

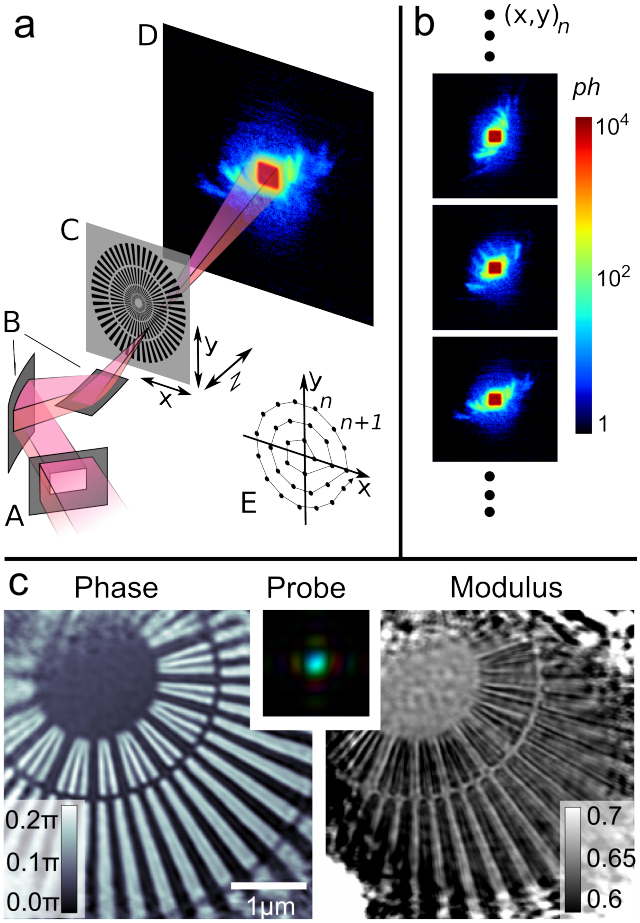


Figure 4.1. – **a** Schematic of the experimental setup (not to scale) used for ptychography at the former ID22NI endstation at ESRF. The X-ray beam enters through an aperture (A) and is focussed by two multilayer Kirkpatrick-Baez mirrors (B). The sample (C) is placed close to the focus and the diffraction patterns are recorded downstream by a CCD coupled to a scintillator (D). To avoid artifacts the sample is scanned along a circular pattern (E). **b** Three typical diffraction patterns taken from the 930 diffraction patterns measured for one ptychographic scan. **c** Phase (left) and amplitude (right) reconstruction of the object transmission. Probe wave field (middle) in complex color encoding placed as inset on the same length scale as the images of the object transmission.

In order to suppress raster-grid pathology (Thibault et al., 2009), the sample is scanned on a circular, 5-fold symmetric scan pattern (Dierolf et al., 2010b) with a shell spacing of half the focal size (75 nm). As a result, each scan consists of 930 diffraction patterns (50 ms exposure time, Fig. 4.1b) and covers a circular area of the sample with 2.7 µm in diameter. The diffraction patterns exhibit a significant halo around the zeroth order (blueish in the chosen color map) which can be attributed mostly to detector PSF (Ponchut, 2006) but also, to a lesser extent, to air scattering and partial coherence.

Ptychographic scans were taken at two different distances from focus, 0.35 mm and 2.1 mm along the beam propagation direction. As mentioned above, the larger distance is expected to yield datasets more prone to reconstruction deficiencies as the increased size of the illumination yields smaller speckles.

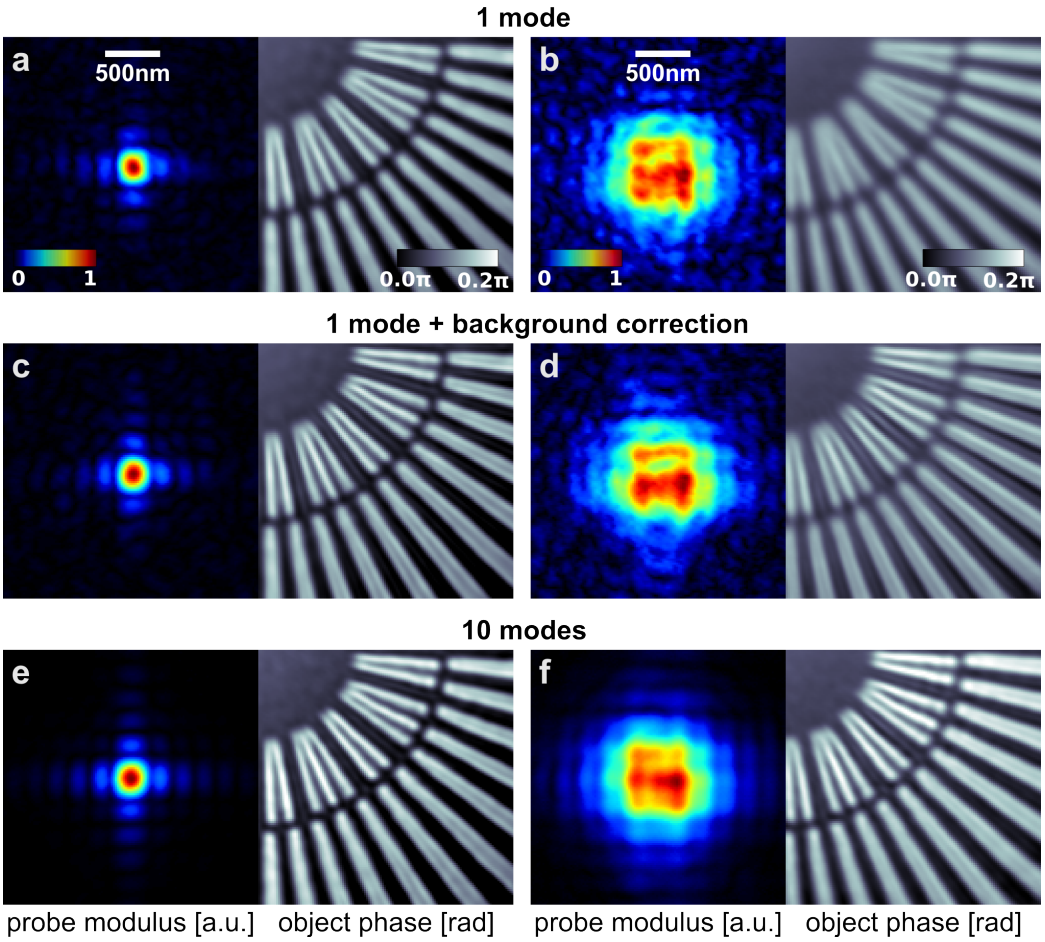


Figure 4.2. – Comparison between three reconstruction strategies: conventional reconstruction (**a** & **d**), reconstruction after background correction on diffraction data (**b** & **e**), and mixed-state reconstruction using 10 modes (**c** & **f**). For each frame, the image on the top displays the modulus of the retrieved probe and the image on the bottom is the phase of the reconstructed object. **a–c** (resp. **d–e**) Reconstructions from the scan at a distance of $+0.35\,\mathrm{mm}$ (resp. $+2.1\,\mathrm{mm}$) from the focus of the illumination. The color scale for all panels is the same as in **a**.

4.1.2. Reconstructions

To verify the effect of the data degradation caused by the detector PSF and, to a lesser extent, by the broad bandwidth, the acquired data was analysed in three ways. First a conventional single-mode ptychographic reconstruction (assuming full coherence) was performed, to form the basis for comparison. The results of this reconstruction are shown in Fig. 4.2a and Fig. 4.2d. As expected, the quality of the reconstruction for the defocussed probe is significantly reduced as compared to the one closer to focus.

A second approach attempted to increase the diffraction image quality by removing the background, possibly induced from the central beam. This task was complicated by the need to model accurately this background signal. Here, we have used the fact that a scan consists of nearly a thousand diffraction patterns to extract collective information from it. The background was approximated by taking the minimum signal value at every pixel over all diffraction patterns. This procedure gave good results except for the central zeroth order. In this area the background was modelled as a two-dimensional Gaussian whose amplitude and width was adjusted to blend seamlessly into the outer part obtained with the minimum operation. This modelled background was subtracted from each diffraction pattern prior to the reconstruction, which again was a conventional single-mode reconstruction. The results, shown in Fig. 4.2b and Fig. 4.2e, show an overall improvement, especially in the object reconstruction with larger probe (Fig. 4.2e), but the retrieved probe still exhibits ripples that do not match with the expected profile for the given experimental conditions.

The third reconstruction used the mixed-states approach (mode decomposition of the probe, see eq. (4.5)). Fig. 4.2c and Fig. 4.2f show a 10-mode ptychographic reconstruction using the uncorrected diffraction images, as in the first approach. After orthogonalization of the modes recovered by the reconstruction algorithm, the first mode (or *main mode*) is the one that comprises the largest number of coherent photons. As can be seen in Fig. 4.2, the resulting reconstruction quality is notably improved, especially in the case of the more extended probe. The sharpness and contrast of features in the reconstruction is seen to be very similar for the small and large illumination cases, which is the expected result since both scans used the same scanning parameters and total dose. The main probe mode is also consistent with the illumination expected for a square pupil.

All ptychographic reconstructions used 300 iterations of a difference map algorithm (DM) and 150 iterations of a maximum-likelihood algorithm (ML) with probe retrieval.

4.1.3. Discussion

The mixed-states approach is manifestly beneficial to alleviate difficulties originating from the detector PSF, partial coherence, or other incoherent scattering. As expected, this approach becomes essential when sampling of the intensity signal gets closer to the Nyquist limit, which occurs for larger probes.

Looking at the orthogonalized modes can provide a better insight into the main sources of data degradation and the way a mixed-states reconstruction can solve the resulting problems. In the current case, Fig. 4.3 reveals that the mixed-states recon-

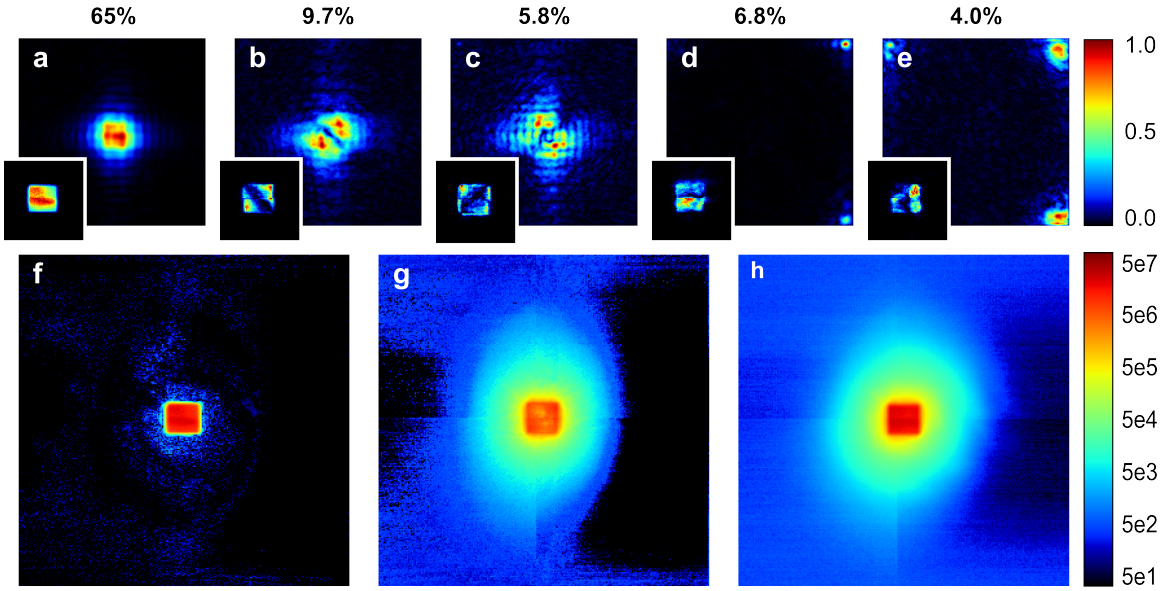


Figure 4.3. – Orthogonalized probe modes recovered from the defocused scan. **a–f** 5 modes out of the 10 with an intensity greater than 2 % of the accumulated intensity of the probe (insets: corresponding Fourier transform magnitudes). In addition to the main mode **a**, two modes **b–c** are associated with the detector point-spread (or partial coherence), and two others **d–e)** are related to the inhomogeneous response of the four quadrants of the detector. **f** Total intensity contribution of the main mode. **g** Total contribution of all other modes. **h** Sum over all measured diffraction patterns.

struction solves two different problems at once. First, the reconstruction addresses the blurring caused by the detector PSF (and possibly partial coherence of the beam – both effects are indiscernible in the present case), as shown by the first three modes. Fig. 4.3a–c. The main mode, Fig. 4.3a, which gathered the most power, has a single lobe. Its propagated intensity, which produces an image of the pupil opening, is consistent with a uniform intensity distribution of the incoming beam on the aperture. The two next modes (b and c) are the "p-waves" expected from the expansion of the convolution kernel of the detector (or partial coherence).

The second problem is of a different nature, since all the remaining modes (of which two are shown in Fig. 4.3d and Fig. 4.3e) have no intensity in the center of the reconstruction frame. The significance of this unphysical result can be understood when one realizes that the outer part of the probe array affects the periphery of the object image, where it is most easy to violate the overlap constraint. Thus by transferring intensity to these areas, the modes can correct for static imperfections in the detector without having to correlate with features in the object transmission function. As can be seen in Fig. 4.3a, which is the sum of all measured diffraction patterns, the four quadrants forming the sensor of the detector have a different response. This discrepancy in the detector response is absorbed in these "outer" modes, whose inho-

mogeneity is especially visible in the diffraction plane, within the high intensity area shaped by the aperture (inset of Fig. 4.3d & e). The propagated main mode yields a sharp image of the beam defining aperture Fig. 4.3f, whereas the contribution from all the other modes Fig. 4.3g accounts for background scattering and signal spread.

While a broadband beam was used for this experiment, attenuating filters were required to avoid saturation of the CCD camera due to hardware limits for the minimum exposure time. As a result of the limited detector dynamic range, the scattered signal was confined within a relatively small solid angle.

Nevertheless, the influence of the fractional bandwidth $\Lambda = \Delta\lambda/\lambda$ can be mitigated by reducing the size of the illumination. More precisely, according to condition (2.92) the spectral blurring of the diffraction patterns is much smaller than the speckle size as long as the extent D of the probe satisfies

$$D < \frac{\ell_{\mathrm{coh}}}{\sin\alpha}\,, \tag{4.6}$$

where ℓ_{coh} is the (longitudinal) coherence length and α the maximum solid angle still carrying signal.

Using (4.6) and assuming that the scattering signal extends up to a scattering angle $\alpha = 2.3 \times 10^{-3}$ (256 pixels), one arrives at $D \lesssim 1.6\,\mu\mathrm{m}$. Even for the larger defocus distance of 2.1 mm, the probe is only about 1 µm wide, indicating that the datasets were mostly free from spectral blurring and justifies the use of monochromatic propagation. Ptychographic reconstructions using multiple wavelengths (polychromatic case) were performed but did not show improvements over the monochromatic case and are therefore not shown here. We will later see in Sec. 5.1 that spectra up to 2 % geometry do not significantly impair a monochromatic reconstruction model.

In order to exploit the full potential of high flux sources in future applications, it will be necessary to use faster integrating detectors with adaptive gain (Henrich et al., 2010), or possibly semi-transparent beamstops (Wilke et al., 2013). For an acquisition scheme where a significant amount of photons is scattered at high scattering angles, it is expected that multi-wavelength propagation will be needed. This new development, as well as other algorithmic improvements, will make use of the increased flexibility of the reconstruction procedure demonstrated in this article to optimize further the speed, resolution and the reliability of ptychography at high-flux synchrotron end-stations.

4.2. Decoherence from stage vibrations

Some experiments were carried out at the cSAXS beam line of the Swiss Light Source to investigate decoherence that arises from small sample movements. If these movements differed locally in the sample plane, a ptychographic scan would contain state mixtures which could only be accurately represented with object modes (see (2.134)). The idea to use vibrations as source for object modes was taken precedence over fabricating an intrinsic statistical sample like the Ising model in (Thibault and Menzel, 2013). For the purpose of artificial vibrations, the sample stage was brought into a controlled mechanical oscillation using a piezo-element driven by an amplified signal generator.

Unfortunately, the rigidity of the specimen at this small lengths scales was underestimated. The mechanical oscillations of the frame only introduced the expected collective vibrations between sample a frame and illumination and no secondary oscillations within the samples. As a consequence, all acquired data fits the model (2.174) which means that either probe or object modes (or any mixture thereof) are theoretically valid models for a reconstruction algorithm.

Collective vibrations are interesting nonetheless as they deteriorate the phase-space measurement in a direction that is orthogonal to partially coherent illumination (see Sec. 2.4.2). In order to see how this alternate source of data degradation is handled by multi-mode reconstructions we take a look at two datasets from that beam time - one with the sample at rest and the other one with an oscillating sample. For the oscillating sample, a object-modes-only model is compared with the common model of probe-modes reconstruction. Apart from discussions about these modes this section demonstrates how to deconvolve the offset density function ρ from both sets of reconstructed modes.

4.2.1. Experiment at cSAXS, SLS

Setup geometry
The distances from detector to sample and from sample to the optics were 7.21 m and 50 mm, respectively. Diffraction data was acquired at an energy of 6.2 keV ($\lambda = 0.20$ nm) with a single PILATUS module limiting the detector ROI to a square array of 192×192 pixel. As each PILATUS pixel has a side length of 172 µm resulting in a pixel size of 43.7 nm in the sample plan which corresponds to a resolution of $11.4\,\mu\text{m}^{-1}$

Probe and Object

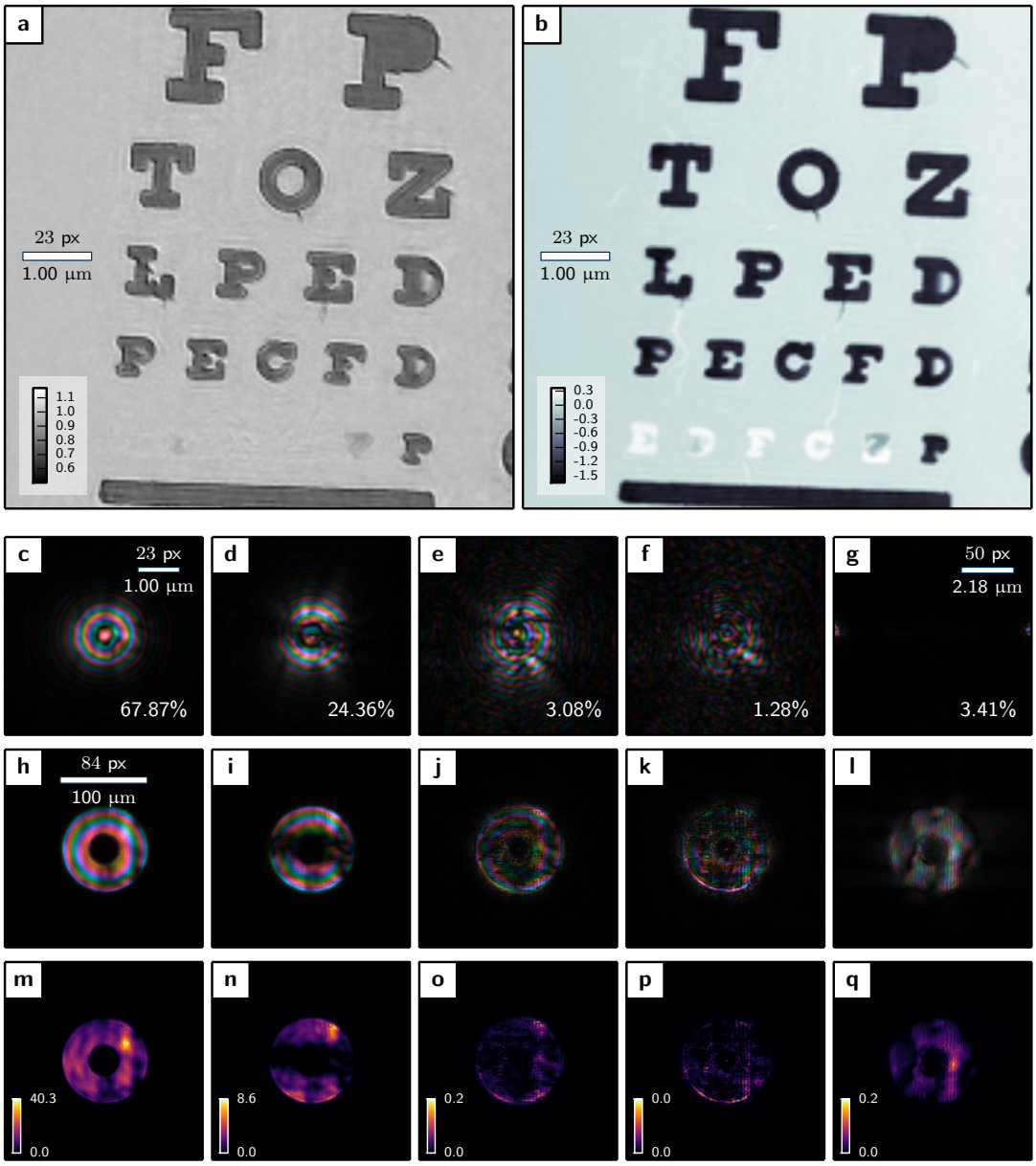
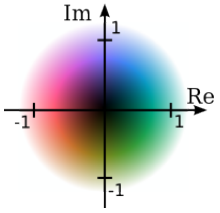


Figure 4.4. – Ptychographic reconstruction of test pattern and defocussed probe from a zone plate. Five modes were allowed to contribute to the probe. **a**, **b** Amplitude image and phase image, respectively, of the test pattern. **c**-**f** Wave field images of probe modes after orthogonalization in descending order. The percentage inscribed in the lower-right corner denotes their power. **g** Irregular mode, with intensity located at the computational boundaries of the probe field of view. **h**-**l** Wave field images of the same modes as in **c**-**g** but propagated to the plane of the zone plate. The respective intensities are displayed in **m**-**q** where the color is scaled in units of $[1000\,\mathrm{ph/s/\mu m^2}]$. All wave fields are renormalized to their maximum value. The color code of the complex plane is displayed to the right.



A Fresnel zone plate with 100µm outer diameter and 50mm focal length was used. Fig. 4.4h displays a wave field image of the probe wave function shortly behind the zone plate. The sample was a test pattern with roughly 20 % absorption and a strong phase shift of 0.5π. Absorption and phase image are depicted Fig. 4.4a & b, respectively.

Scan pattern
From all data taken, two ptychographic datasets are compared here: One dataset with the sample at rest and one with the sample oscillating sinusoidally at 10Hz frequency and 250 nm amplitude in horizontal[2] direction (with respect to the image orientation in Fig. 4.4). Both scans applied a round pattern of equidistant shells where scan positions were discarded that lay outside a square field of view of 5 µm side length. The shell width was 300 nm and a total number of 221 scan points were acquired per scan. For each scan point the sample was exposed for 0.5 s collecting between 7.13 and 8.27 and 7.71×10^8 photons per exposure.

4.2.2. Reconstructions

Sample at rest
The sample at rest was reconstructed using 400 iterations of DM and additional 150 iterations of ML refinement. Fig. 4.4 displays an overview of the reconstruction with phase and amplitude reconstructions in the upper panels. Even though the sample was at rest, the reconstructions required a few additional modes in the probe to achieve best result. This indicates that the illumination was partially coherent. The reconstruction started with 5 modes which were modelled according to the description above. Modes no. 2–5 were only given 10 % of the main mode's power and had noise added to them to diversify the initial starting conditions.

The lower three rows of Fig. 4.4 show the final five orthogonalized modes. The main mode (Fig. 4.4c) accumulated roughly 68 % of the total power suggesting that more than 30 % of photon energy would not be considered in an entirely coherent reconstruction model.[3] The second mode (Fig. 4.4d) exhibits a zero-crossing in vertical direction (corresponds to horizontal direction at the beamline), which is a p-type

[2] The horizontal direction in the images of this section corresponds to the vertical direction in the laboratory coordinate system. All images are rotated by 90° as the human eye is used to read letters upright.

[3] This may be barely noticeable to the unaware observer. A ptychographic algorithm strives to return the best coherent fit to the data within its constraint set. This means that the algorithm may converge at a higher level of discrepancy between modelled and measured diffraction data to be consistent with the overlap projection. The result may, however, strike the user as an excellent reconstruction even though not all photons have been accounted for.

mode like the ones in Fig. 4.3b. However in this case, it is not caused by a potential point spread in the detector but consequence of reduced coherence in the horizontal direction controlled by primary vertical slit of the beamline. Modes three and four (Fig. 4.4e,f) contribute very few photons to the probe and can also not identified easily as a mode of the upstream illumination. As the back-propagated wave-fields reveal, these modes are mainly sourced from the rim of the FZP and may thus be caused from additional scattering structures there. The fifth mode (Fig. 4.4g) is an irregular mode with the power localized in the less constraint edges of the numerical reconstruction frame. Just like the modes in Fig. 4.3d it cannot be an eigenvector of the probe's mutual intensity. As we can also not attribute this mode to different detector modules, incoherent air scattering in the (far-field) image of the FZP may be a plausible cause.

Oscillating sample

For the oscillating sample, two different reconstruction strategies were applied The first strategy enabled 10 probe modes and a single (coherent) object mode in the mixed-state model (2.221),

$$I_{\mathrm{j}}(\mathbf{q}) = \sum_{m=0}^{9} \left| \underset{\mathbf{x},\mathbf{q}}{\mathcal{F}} \left\{ p_m(\mathbf{x})\, o(\mathbf{x} - \mathbf{y}_j) \right\} \right|^2 , \tag{4.7}$$

while the second enabled six object modes and a single (coherent) probe mode,

$$I_{\mathrm{j}}(\mathbf{q}) = \sum_{n=0}^{5} \left| \underset{\mathbf{x},\mathbf{q}}{\mathcal{F}} \left\{ p(\mathbf{x})\, o_n(\mathbf{x} - \mathbf{y}_{\mathrm{j}}) \right\} \right|^2 , \tag{4.8}$$

Both strategies used 1000 iterations of DM and additional 150 iterations of ML refinement as reconstruction engine. The comparatively large number of iterations stems from the probe-modes strategy which took roughly 700 of the 1000 iterations in DM to get on a path towards convergence. The object-modes strategy converged quickly but was iterated the same number in order for the results to be comparable.

The orthogonalized probe-modes p_m from model (4.7) are depicted in Fig. 4.5b–h next to the coherent probe p in Fig. 4.5a. The main probe mode (Fig. 4.5b) does neither resemble the coherent probe nor the main probe mode of the sample at rest (Fig. 4.4c). The oscillation, which corresponds to an amplitude of roughly 5 pixel in horizontal direction, stretches the probe modes horizontally by twice the number of pixels. Characteristic for that stretch are the horizontal zero-crossings that appear in modes Fig. 4.5b,c & e. The vertical zero crossing ("p-lobe") from partial transverse coherence is noticeable in modes Fig. 4.5f,g & h. For example, the mode e could be the partial transverse coherent complement to b as it looks very similar besides the p-lobe. It also has also one third of the power of b, just like the mode d in Fig. 4.4 contains roughly on third of the power of c.

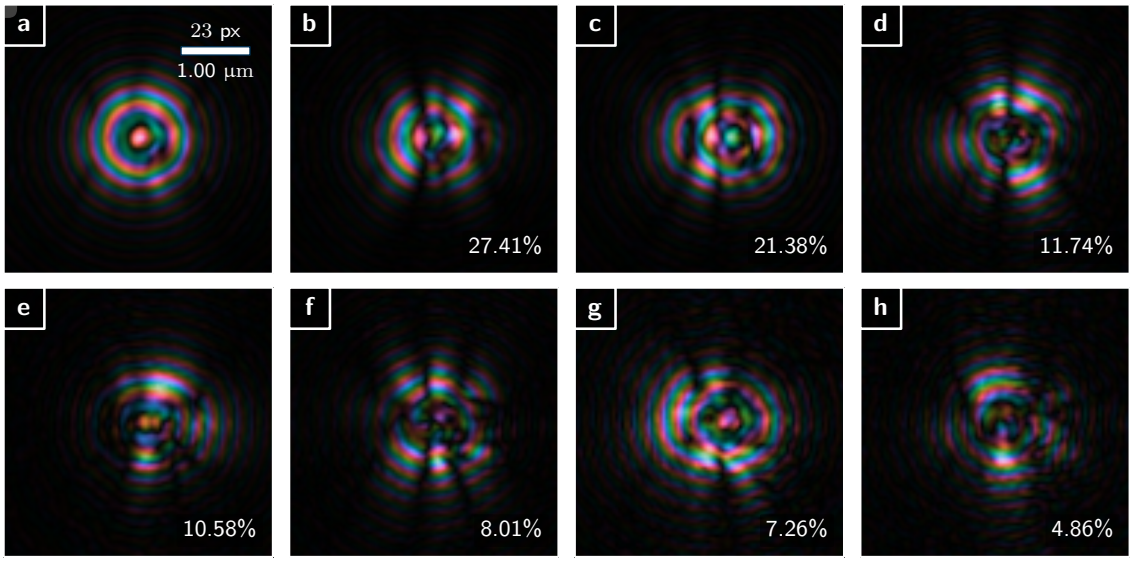


Figure 4.5. – Ptychographic reconstruction of a test object with sinusoidal vibration of $250\,\mathrm{nm}$ amplitude. One object mode and 10 probe modes were allowed of which 7 are displayed in **b-h** as wave-field images in descending order with respect to their relative occupancy. They accumulate $91\,\%$ of the total photon count. The remaining modes were irregular (see Fig. 4.4g for comparison) and are thus not displayed here. For comparison, **a** shows a wave-field image of the coherent probe of the object-mode-only model (4.7) which is almost identical to main mode of the non-vibrating measurement. The scale is the same in all images.

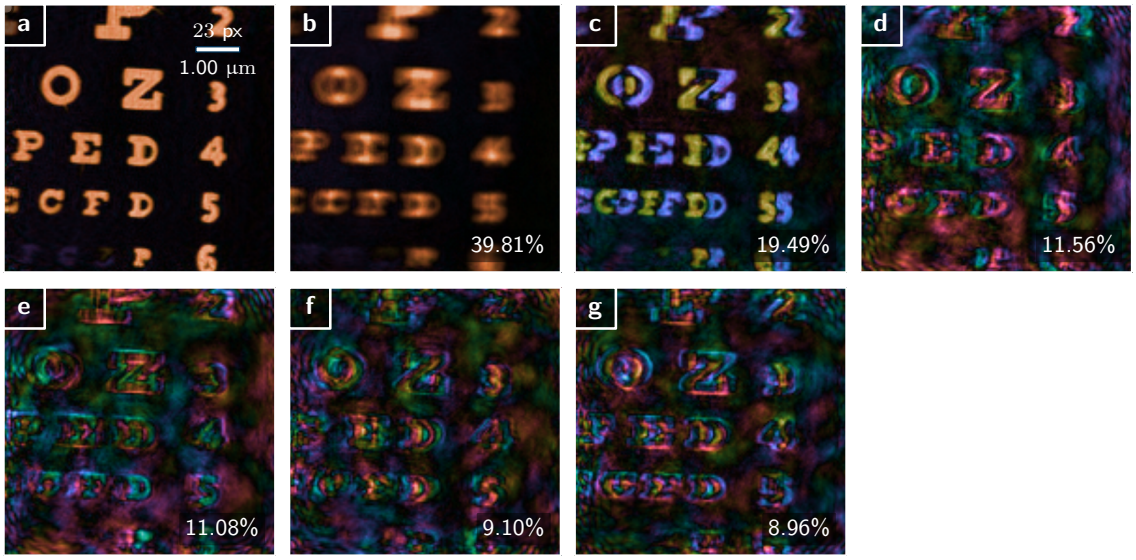


Figure 4.6. – Ptychographic reconstruction of a test object with sinusoidal vibration of $250\,\mathrm{nm}$ amplitude. One probe mode and six object modes were allowed of which the six object modes are displayed in **b-g** as wave-field images in descending order with respect to their relative occupancy. For comparison, **a** shows a wave-field image of the coherent object of the object-mode-only model (4.8) which corresponds to the object of the non-vibrating measurement. The substrate transmittance was subtracted from the main object modes (**a** and **b**). The scale is the same in all images

The orthogonalized object-modes o_n from model (4.8) are shown in Fig. 4.6b–h next to the coherent probe o from model (4.7) in Fig. 4.6a . The substrate transmittance was subtracted from both the coherent object (a) and the main mode of the object-mode reconstruction (b). After subtraction the main mode contains the averaged scattering density $o_0(\mathbf{x}) - 1 = \langle \tilde{o}(\mathbf{x}) \rangle$ (see eq.(2.140)). When compared with Fig. 2.3h, the sinusoidal amplitude of the oscillation is observable in the pronounced contrast at reversal points of oscillation. The second strongest object mode (Fig. 4.6c) is apparently a mode that belongs solely to states at reversal point of the oscillation. It corresponds well with Fig. 2.3i.

The remaining object modes elude a direct interpretation. Not only do they seem to have different characteristics for each region but they also exhibit an unexpected low-frequency background. To some extent the background may be explained as the response of the model to partial transverse coherence. On the other hand, the "coherent" background is given the same degrees of freedom as the scattering density $\tilde{o}$ which actively contribute to the diffraction signal of the object. As a consequence we might argue that the low-frequency background is less constraint by the dataset and the additional freedom in the object modes invokes unphysical results in the background.

Whether object modes or probe modes, we should not forget that modes are merely intermediate constructs to build the correlation function of probe and object. In a statistical sense, only the entire set bears a meaning. Meticulously seeking a cause for each mode might therefore sometimes be a fruitless endeavour - especially for those modes with lower power.

The next section is therefore dedicated to extract the object density function ρ as a collective meaning of these two sets of modes.

4.2.3. Coherent modes as vibration kernel

Allowing a modal expansion for probe or object factors in a diverse set of decoherence sources. As only the entire set of orthogonal modes represents a single probe (or object), it remains sometimes difficult to attribute a specific meaning to an individual mode. Hence, if the modes arise from "simple" decoherence source like vibration or partial spatial coherence we may want to retrieve directly interpretable quantities like the ODF or the PSF instead of visually inspecting a set of orthogonalized modes. This section presents a numerical procedure for that case.

If we compare eq.(2.176) with eq.(2.189) and assume that only the object is affected by decoherence, the ambiguity synthesis of the probe has to fulfil the following con-

dition:

$$\sum_n \mathcal{A}_{o_n}(\mathbf{w},\mathbf{x}) = \mathcal{F}\rho(\mathbf{w})\,\gamma(\mathbf{x})\,\mathcal{A}_{O_{\text{coh}}}(\mathbf{w},\mathbf{x})\,. \tag{4.9}$$

As the right side of eq.(4.9) consists essentially of three unknown quantities, we reinterpret the equation as the solution of a least square minimization with respect to γ, ρ and o of the cost function

$$\mathcal{L} = \iint \left| \sum_n \mathcal{A}_{o_n}(\mathbf{w},\mathbf{x}) - \mathcal{F}\rho(\mathbf{w})\,\gamma(\mathbf{x})\,\mathcal{A}_{O_{\text{coh}}}(\mathbf{w},\mathbf{x}) \right|^2 \mathrm{d}\mathbf{w}\,\mathrm{d}\mathbf{x}\,. \tag{4.10}$$

If the cause for decoherence is either vibration or partial spatial coherence but not both, further simplifications are possible. In absence of vibrations ($\mathcal{F}\rho = 1$) an inverse transform with respect to $\mathbf{w}$ yields

$$\mathcal{L} = \iint \left| \sum_n o_n^*\left(\mathbf{y} - \frac{\mathbf{x}}{2}\right) o_n\left(\mathbf{y} + \frac{\mathbf{x}}{2}\right) - \gamma(\mathbf{x})\, o^*\left(\mathbf{y} - \frac{\mathbf{x}}{2}\right) o\left(\mathbf{y} + \frac{\mathbf{x}}{2}\right) \right|^2 \mathrm{d}\mathbf{y}\,\mathrm{d}\mathbf{x} \tag{4.11}$$

$$= \iint \left| \sum_n o_n^*(\mathbf{x}_1) o_n(\mathbf{x}_2) - \gamma(\mathbf{x}_1 - \mathbf{x}_2)\, o^*(\mathbf{x}_1) o(\mathbf{x}_2) \right|^2 \mathrm{d}\mathbf{x}_1\,\mathrm{d}\mathbf{x}_2\,. \tag{4.12}$$

An equivalent cost function can be found for full spatial coherence ($\gamma = 1$). A forward transform with respect to $\mathbf{x}$ and subsequent shift of integration variables results in

$$\mathcal{L} = \iint \left| \sum_n \mathcal{F}^* o_n^*(\mathbf{v}_1) \mathcal{F} o_n(\mathbf{v}_2) - \mathcal{F}\rho(\mathbf{v}_1 - \mathbf{v}_2)\, \mathcal{F}^* o^*(\mathbf{v}_1) \mathcal{F} o(\mathbf{v}_2) \right|^2 \mathrm{d}\mathbf{v}_1\,\mathrm{d}\mathbf{v}_2\,, \tag{4.13}$$

where we used an equivalent formulation of the coherent ambiguity function,

$$\mathcal{A}_{O_{\text{coh}}}(\mathbf{v},\mathbf{y}) = \int \mathcal{F}^* o^*\left(\mathbf{w} - \frac{\mathbf{v}}{2}\right) \mathcal{F} o\left(\mathbf{w} + \frac{\mathbf{v}}{2}\right) e^{2\pi \mathrm{i}\,\mathbf{w}\mathbf{y}}\,. \tag{4.14}$$

Both cost functions (4.12) and (4.13) present the same signature and may thus be minimized by the same algorithm. One possible algorithm is derived in the appendix A.2.

Needless to say, the mechanism to extract ρ or γ works exactly the same when the probe ambiguity $\mathcal{A}_P$ is assumed to be affected by decoherence. Therefore both sets of modes from the previous section will be used to extract the ODF.

Figure 4.7 shows the result of the deconvolution procedure for probe and object modes. We observe that the outcome is almost identical to the coherent object (Fig. 4.5a) and probe (Fig. 4.6a) reconstruction which emphasizes that the minimization procedure very well determines the coherent wave fields p and o. However, the ODFs ρ as displayed in Fig. 4.7c and Fig. 4.7e differ. The theoretical ODF (Fig. 4.7c) shows

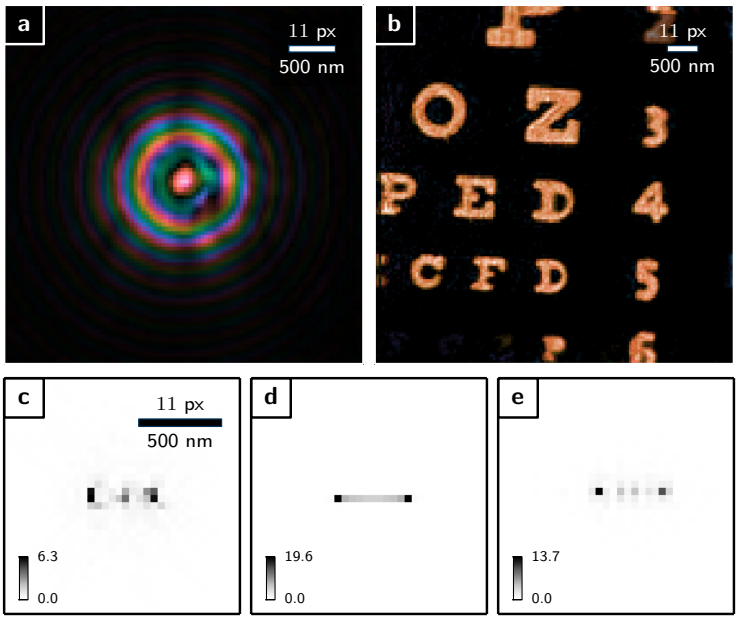


Figure 4.7. – Deconvolution from modes using least squares. **a** Probe deconvolved with kernel (**c**) using the probe modes depicted in Fig. 4.5. **b** Object deconvolved with kernel (**e**) using the object modes depicted in Fig. 4.6. **d** Theoretical kernel for a sinusoidal oscillation as comparison.
Convolution kernels are scaled in percent. The complex wave fields in panel **a** and **b** have been renormalized to their maximum value. For the color code of the complex plane, see Fig. 4.4.

that the probability density should be confined to a single line, a result shared with the object ODF in e. The result for the probe ODF spreads a few pixel in vertical direction which suggests, that the partial spatial coherence from the upstream optics, which is inevitably present in the modes (Fig. 4.5), feeds back into the deconvolution process. However, that partial spatial coherence should have also been encoded in the object modes (Fig. 4.5) in one way or the other and produce the same ODF.

The slight difference in probability between from the experimental to the theoretical result could arise from an actual non-ideal sinusoidal movement of the sample stage. Despite this small discrepancy, the retrieval procedure works exceptionally well. The deconvolution algorithm was also tested on simulated vibrations where it converged reliably with 10 modes.

4.3. Decoherence from object fluctuations

Thibault and Menzel (2013) reported about a simulation where a mode expansion of the object (see Sec. 2.3.7) was employed to identify ferromagnetic and antiferromagnetic coupling in a simplified Ising model. However, an experimental proof for the necessity of object modes is still pending.

This section details a selection of proof-of-concept experiments where random specimen movement was stimulated by an air stream from a common computer fan. Like flags waving in the wind, it is well known that Bernoulli's principle causes flexible objects to oscillate when aligned parallel to an air stream and anchored at the wind-facing side. Here, the experiments focussed on light objects like insect wings, a dandelion seed or a human hair which are known to be susceptible to even small air streams. For example, a hair exposed to a parallel aligned air stream is expected to oscillate in a similar way as depicted in Fig. 2.3. If these oscillations occur at comparable length scales as what the diffraction setup can resolve, ptychographic reconstruction would require an object mode approach due to the oscillations being non-isotropic within the scanned region.

The experimental setup was the same as described in Sec. 3.3.1. Diffraction images were acquired 145 mm downstream of the specimen and binned by 3×3 pixels, creating effective detector pixels of 72 µm × 72 µm area. Other than in Sec. 3.3.1, images of high dynamic range were stitched together from a series of exposures. Starting from the longest exposure, each pixel counting more than 50000 events (80% fill) was replaced subsequently by a scaled value of the next shorter exposure. The exposure series (in seconds) was $[0.05, 0.2, 0.8, 3.2, (12.8)]$, yielding up to 10^9 photons per scan point. Exemplary diffraction patterns are shown in Fig. 4.8. The laser wavelength of 650 nm results in a reconstructed pixel size of 5.1 µm.

4.3.1. Dandelion seed

The pappus of a seed from a member of the Taraxacum species was scanned a total of three times on a circular scanning pattern in a rectangular scan field of 2.6 mm × 2 mm side length. The first scan comprised 283 diffraction patterns at a scan shell width of 80 µm. As the fan was turned off, it served as still-air reference scan. Fig. 4.8a displays the single-mode reconstruction using 150 iterations of DM and 150 iterations of ML refinement. The reconstructed probe is inset in the lower left and resembles Fig. 3.5a. The object transmission is mainly air interspersed with the feathery bristles of the pappus. We observe that not all bristles are sharp, indicating, that the pappus was in parts out of focus. A few positions are highlighted and the respective diffraction

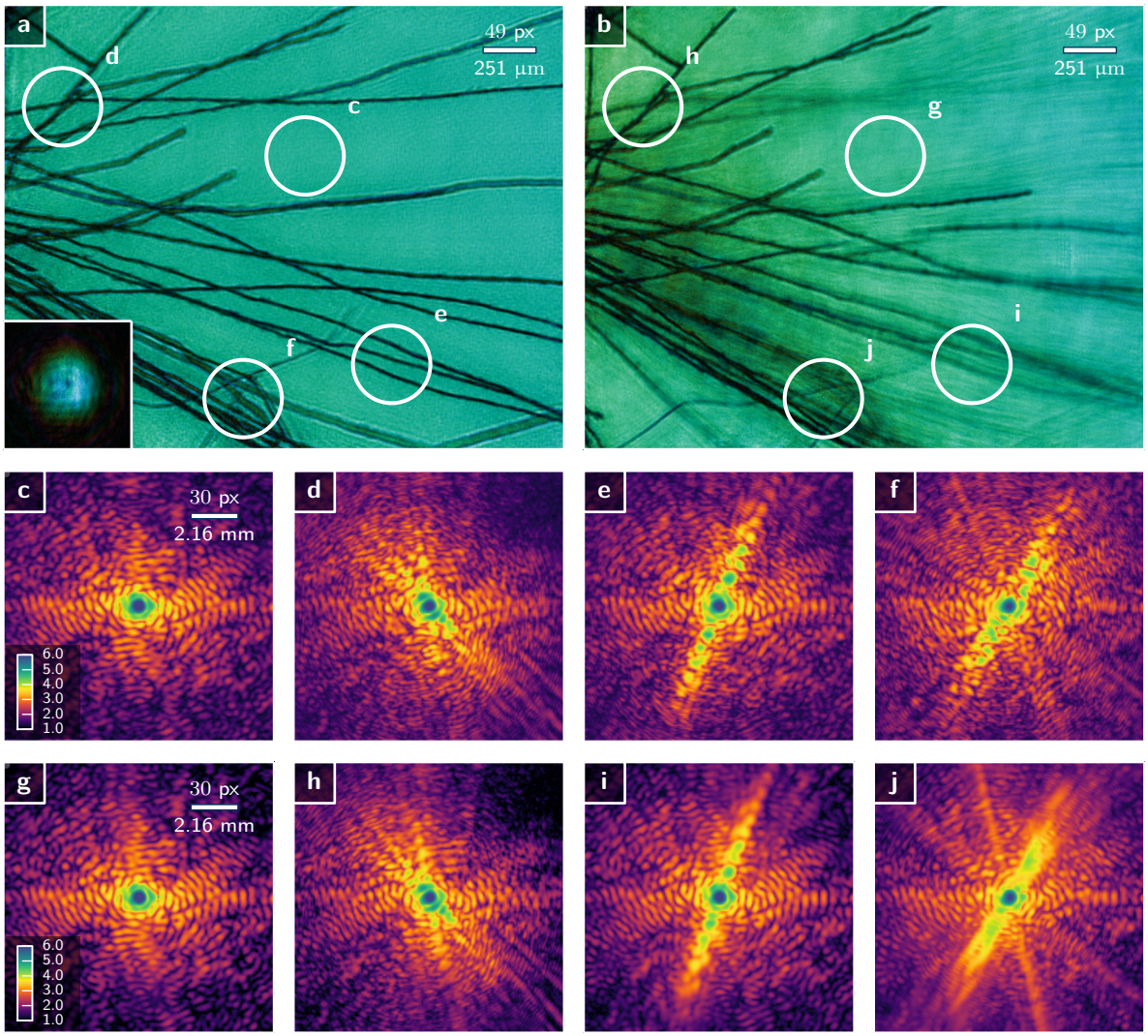


Figure 4.8. – Ptychographic reconstruction of the pappus of a seed from a member of the *Taraxacum* species, commonly known as dandelion. The hair-like, feathery bristles of the pappus act as parachute to help the seeds disperse through air. Each bristle holds many barbs which are the reason for the rough surface of the bristles and should not be mistaken for a reconstruction artifact. Beak and seed are outside the field of view to the left. **a** Wave field image of the sample tranmission in still air. An image of the probe is located in an inset to the lower left (same scale). Four white circles with the probe's diameter highlight examplary positions of the scan pattern. The diffraction images (still air) to each of these position are displayed in panels **c**-**f**. The diffraction images in **g**-**h** were taken at the the exact same scanning positions but the pappus was agitated with a fan. The resulting oscillation in the bristles cause the diffraction speckles of the bristles to blurr out. For illustration purposes, the wave-field image of the main mode of a recontruction with many object modes is shown in **b**. The main object mode usually appears as time average for oscillatory movements and thus exhibits a motion blurr in various regions. A visible reconstruction artifact is the small phase ramp in horizontal direction which can be observerd in the hue channel of the image.

patterns shown in Fig. 4.8c–f.

The diffraction images at these points are compared with those of a second scan that used the same scan pattern and acquisition scheme but with an activated air stream. The respective diffraction patterns are shown in Fig. 4.8g–j. We notice, that not all regions of the object exhibit a blurring of speckles. Positions c (and g) are unaffected as they contain diffraction from the probe only and positions d (and h) contain diffraction from bristles that stood apparently still even when the fan was activated. Positions e,f (and i,j) lie in regions that are both susceptible to speckle blurring from bristle movement with differing severity. We note that only the diffraction speckles from the bristles are blurred while some speckles from the probe diffraction stay sharp. Fig. 4.8b shows the main object mode of a 3-mode reconstruction of the second scan. As the main object mode displays the averaged object transmission, we readily verify the observation from the diffraction patterns. The four emphasized scan positions underline that a probe-modes-only approach cannot work here as otherwise either the speckle blurring had to be isotropic for the entire diffraction image (partial spatial coherence) or at least for the speckle contributions from scattering at the bristles.

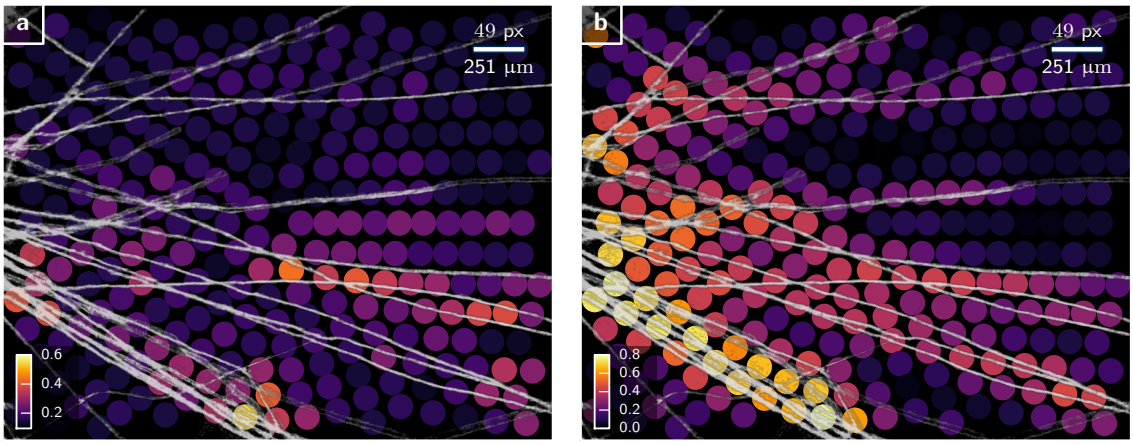


Figure 4.9. – **a** Map of the decoherence signal (4.16), plotted at shift positions $\mathbf{y}_\mathsf{j}$, reveals where the oscillation of the seed's bristles was strongest. An semitransparent image of the bristles at rest ($|o|$) is overlaid in white for comparison. **b** Map of the scattering signal, (4.15), plotted at shift positions $\mathbf{y}_\mathsf{j}$, reveals the distribution of the scattering density. It correlates well with the image of the bristles at rest.

Fig. 4.9 comprises the same analysis but for all points of the scan. Fig. 4.9b displays the scattering contrast which is the normalised euklidean distance to the empty diffraction pattern (scan point index $\mathsf{j} = 27$),

$$\text{scattering}(\mathbf{y}_\mathsf{j}) = \frac{\|I_{\mathsf{j},\text{coh}}(\mathbf{q}) - I_{27,\text{coh}}(\mathbf{q})\|_\mathbf{q}}{\|I_{27,\text{coh}}(\mathbf{q})\|_\mathbf{q}} \tag{4.15}$$

This signal corresponds very well with the absorption of the specimen in still air that is overladen in semitransparent white.
Fig. 4.9a shows the decoherence signal which is the normalised euklidean distance to the diffraction image of the resting scan,

$$\text{decoherence}(\mathbf{y}_\text{j}) = \frac{\|I_{\text{j,coh}}(\mathbf{q}) - I_\text{j}(\mathbf{q})\|_\mathbf{q}}{\|I_{\text{j,coh}}(\mathbf{q})\|_\mathbf{q}} \tag{4.16}$$

For decoherence caused by isotropic vibrations, we would expect that the decoherence signal is mostly proportional to the scattering signal, which is apparently not the case here.

Scattering and diffraction signal unveil the bottom right region to have the highest ratio of decoherence signal compared to the scattering signal and thus being most interesting for mode analysis.

In order to probe the phase space dense enough to allow for many object modes, the dandelion pappus was scanned a third time in the same rectangular region but with a denser pattern of 80 µm shell distance resulting in 644 diffraction images. Additionally, this scan featured an extra exposure of 12.8 s to further increase the dynamic range. This third scan was reconstructed enabling 5 object modes and using 300 iterations of DM and 400 iterations of ML refinement. The object modes had the same uniform transmission as starting guess with a little noise added on top to diversify starting conditions.

Fig. 4.10b–f display these 5 object modes in the chosen region. The inscribed percentages reflect their power contribution to the transmission signal. Fig. 4.10a shows the object reconstruction from the first scan in still air for comparison. We observe to some extent the dislocation of the bristles from oscillatory movement already in these modes. Additionally we note an uneven low-frequency background spanning almost the full absorption range and a phase range of roughly $\frac{\pi}{2}$

Fig. 4.11b–f display the object modes after orthogonalization. The inscribed percentages reflect their power contribution to the transmission signal. Fig. 4.11a is the same as Fig. 4.10a but with the transmission of air subtracted in order to show the scattering density. The same operation was applied to the main object mode in b. Just like for the vibrating test pattern in Fig. 4.6b the mean scattering density picks up roughly 40% of the total scattering power. All other modes in c–f have comparable scattering power and there is no real "secondary mode" like in Fig. 4.6c. Additionally, these modes do not show a uniform characteristic over the field of view, which is expected due to independent statistics of each bristle. Looking closely at individual bristles we can at least spot a certain resemblance to modes i & j of the oscillating wire in Fig. 2.3. Mode i seems to be the predominant mode for the bristle oscillation.

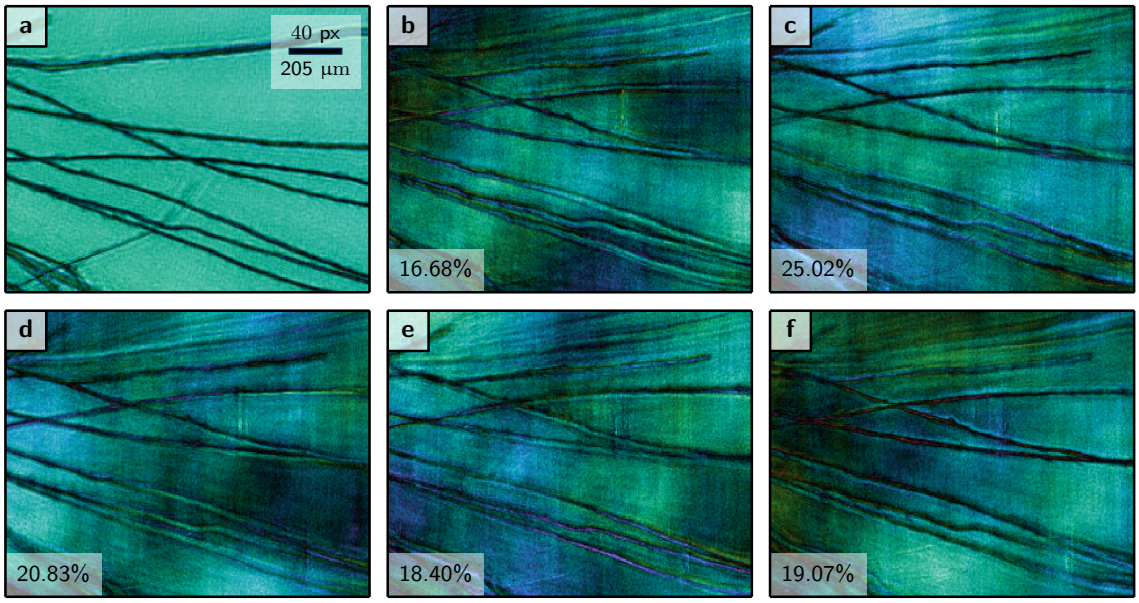


Figure 4.10. – **a** Wave field image of reconstructed object transmission $o(\mathbf{x})$ for the pappus in still air. The region of interest is in the lower right quadrant of Fig. 4.8a. **b–f** Wave field images of the 5 object modes enabled for the reconstruction of the wind-stimulated pappus. The bristle displacements are readily observable

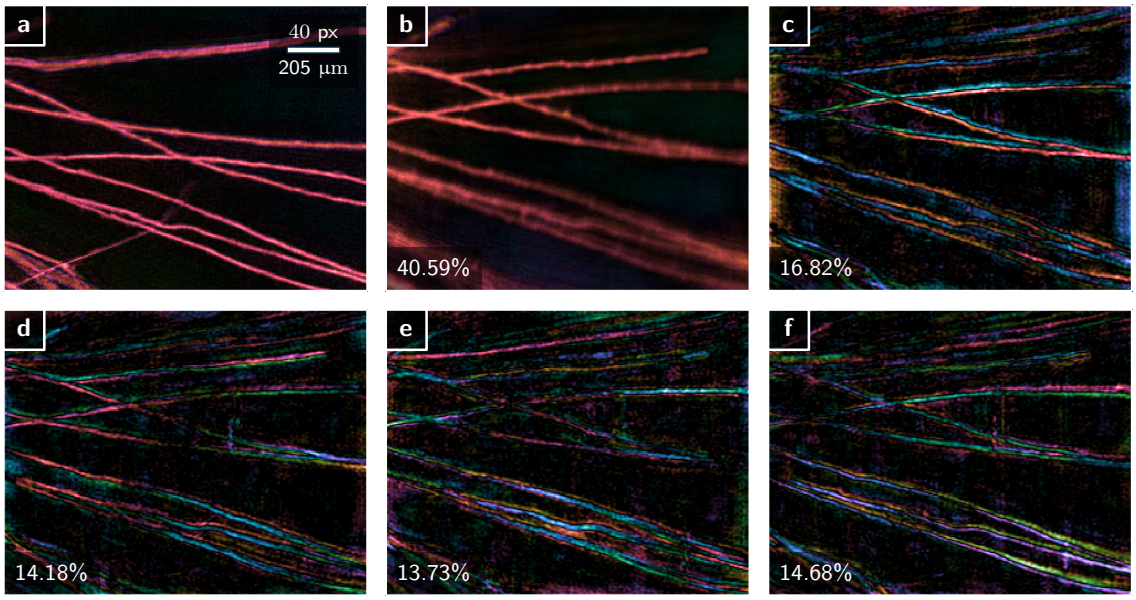


Figure 4.11. – **a** Wave field image of reconstructed object scattering potential $o(\mathbf{x}) - 1$ for the pappus at rest. The ROI is the same as in Fig. 4.10a. **b** Main mode, i.e. averaged object transmission $o_0(\mathbf{x}) = \langle o(\mathbf{x}) \rangle - 1$ of the 5 orthogonalized object modes of the wind-stimulated pappus. **c–f** Wave field images of the remaining 4 orthogonalized object modes. Fields are high-pass filtered in order to suppress low-frequency background variations.

Other than in Fig. 2.3i–k, the modes do not always extend to the full length of the bristle but instead parts of the same bristle appear in separate modes with the same mode characteristic.

4.3.2. Single hair

As a simpler specimen to study object modes, a single hair was scanned with and without air stream using a circular pattern within a rectangular scan field of 2.6 mm × 2 mm. Each scan comprised 419 diffraction patterns at a scan shell distance of 100 µm. The scan in still air was reconstructed with a single object mode and 300 iterations of DM and 150 iterations of ML refinement. The retrieved object transmission, $o(\mathbf{x})$, is shown in Fig. 4.12a. The scan with airflow used the same reconstruction engine but the model allowed for three object modes. Fig. 4.12b–d display the central region of the three retrieved object modes. Fig. 4.12f–g contain the modes after orthogonalization with respect to the diplayed frame. Even though modes b–d appear less prone to a low-frequency background, a high-pass filter was applied to the g & h in order to highlight the modes along the hair boundaries.

The three mode reconstruction allows direct interpretation. The main mode in f is the averaged scattering density $\langle o(\mathbf{x})\rangle - 1$ while the mode in g can be attributed to a vertical translation of the entire hair. Mode h corresponds well with Fig. 2.3i which is the secondary mode of the oscillating wire. Other than for the bristles of the dandelion, the modes of the hair do extend along its entire length.

4.3.3. Discussion

The apparent bright spot in Fig. 4.12c arises from a malfunctioning shutter. The shortest exposure was corrupted from the shutter not closing and thus overexposing the frame. In fact many scan points in the scans analysed here were corrupted by a malfunctioning shutter and the shutter had to be replaced after the experiments. To what extend the corrupted data has influenced the reconstructions should be analysed further. The characteristic vertical structure of the spot anomaly in c appears to have radiated to other regions of the field of view. Such behavior is not entirely uncommon for ptychography as anomalies that are imprinted onto the probe will eventually be noticeable in the background and reflect the scanning pattern. This effect, known as raster-grid pathology (Thibault et al., 2009) is usually penalized in coherent datasets with large overlap and non-periodic scan pattern. Here, the additional degrees of freedom from the object modes may have relaxed the overlap constraint to

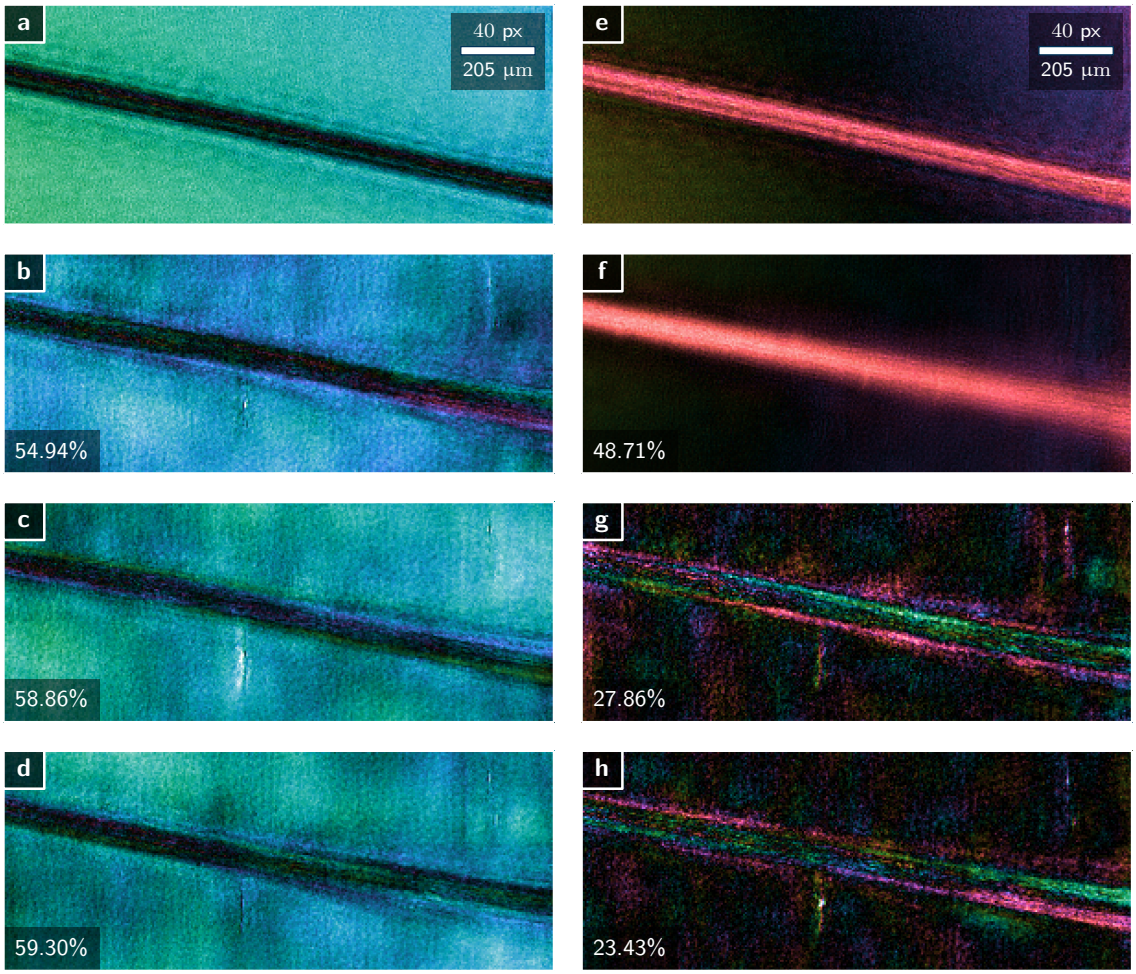
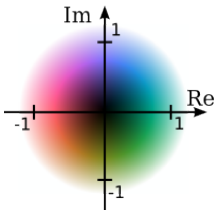


Figure 4.12. – Reconstructed object transmission of a hair with and without air stream. **a** Wave field image of reconstructed object transmission $o(\mathbf{x})$ for the hair in still air. **e** scattering potenzial $o(\mathbf{x})-1$.**b–d** Object modes as obtained directly after engine has finished. **f–h** Orthogonalized modes $o_n(\mathbf{x})$. The transmission of air was subtracted from the main mode in **f**. All wave fields share the same length scale and have been renormalized such that the transmittance of air lies on the unit circle in the complex plane. The color code of the complex plane is displayed to the right.



allow the shutter artefact to spread in the background. A certain tendency to a vertical stripe pattern is also noticeable in Fig. 4.11b–f.

As data corruption cannot be ruled out entirely, the analysis presented here must be considered preliminary. Besides data integrity it raises some other open questions.

- The origin of the low-frequency background remains unknown. If not caused by shutter artifacts, the low frequencies in the object transmission could generally be very little contraint in a model that allows object modes. For example, as the blurring in diffraction patterns Fig. 4.8g–j does not spread across the complete frame, the volume of deteriorated phase-space may be small, maybe too small to make use of the additional degrees of freedom offered from object modes. Against this hypothesis stands the observation that it usually takes a larger number of iterations for the engines to form the object modes. A loosely constrained problem with too many degrees of freedom usually converges fast.

- While the modes of the reconstructed hair extend along the entire length of the hair within the regarded field of-view, the same statement cannot be made for the bristles. This may, in fact, relate to the statistics of the oscillations. For a system as complex as the pappus, the requirement for a statistically stationary process might get infringed from the scan taking too much time. The other question is whether the mode decomposition of the object does extend beyond the probe size at all.

- Additional measures ensured that the hair oscillated only transverse to the beam. The same restriction could not be imposed on the seed, which may have also swung longitudinally into and out-of focus during exposure. Of course, these type of oscillations are not covered by the wire model in Fig. 2.3.

Despite these open questions, the general resemblance of the object modes of hair and pappus to the theoretical modes in Fig. 2.3 is striking, especially for the hair specimen. It confirms the validity for the object-mode model to handle this kind of specimen and measurement conditions.

4.4. Conclusion

The mode expansion for the probe has become a ubiquitous tool to address multiple source of decoherence ranging from partial transverse coherence (Thibault and Menzel, 2013; Enders et al., 2014; Stockmar et al., 2015b) to vibrations (sec.4.2) and other dynamics (Clark et al., 2014). The modal approach is

useful even when the setup provides coherent conditions, but an incoherent on-the-fly scanning mode (Deng et al., 2015a; Pelz et al., 2014) promises significant acceleration of the measurement.

The probe-mode-only approach also assumes a coherent object which is often the desired result when ptychographic phase retrieval serves as microscopy technique. Moreover, it is possible to attribute some probe modes to specific properties of the setup as we have seen in sec. 4.2 and sec. 4.1.

For far-field ptychography with a localised probe, the phase space of the probe is strongly constraint by the measurement. As a consequence the additional degrees of freedom in the mode expansion of the probe are unlikely to lead an engine to converge to a non-physical solution. Hence, there is no obvious reason to not use a few modes instead of a single coherent probe. Sometimes, allowing a few modes even though the data was considered coherent can lead to surprising insights (see Fig. 3.6)

The argument in favour of a mode expansion for the object is not as straight forward. First of all, the reconstruction will deliver a coherent probe along with the correlation of the object transmission, which should be intentional. The meaning of some individual object modes may elude any rational interpretation and the orthogonalization procedure depends on the actual cropped field of view if the object is not isolated. As a consequence, the power series of the object modes may not exhibit a steady decline as it usually does for the mode expansion of the probe. If the mode with the lowest power does not fall below a few percent, there remains a hint of uncertainty if enough modes were enabled. On the other hand, enabling more object modes just to be on the safe-side can introduce a disproportionate amount of degrees of freedom which exhaust the overlap redundancy and may cause the algorithm to convergence to non-physical solutions. The remaining question is if the orthogonalization only bears meaning when applied to a ROI within the limits of the probe size as was done by Thibault and Menzel (2013).

However, there is a silver lining for the application of object modes. In near-field geometry, the illumination may well be much greater than the object (Stockmar et al., 2013, 2015b). In this case, the phase space of the sample is stronger constraint than the phase space of the probe. Object modes may then be the logical choice. That is also the case if the statistics of dynamic processes, possibly within a buried sample, become a field of intense research in ptychography. At time of publication of this thesis, sec. 4.3 signifies the first and single reported use of object modes for experimental data.

For well known problems such as sample stage vibrations or partial transverse coherence, the procedure outlined in sec. 4.2.3 describes a valuable post-processing tool to deconvolve either the PSF $\tilde{\gamma}$ or the ODF ρ from the reconstructed set of modes.

5. Ptychography in broad-bandwidth conditions

As reported by Thibault and Menzel (2013), opening the upstream slits of a beamline increases the photon flux but reduces partial spatial coherence in the illumination. This conflict between signal strength and speckle contrast of the diffracted light is characteristic for light sources employing the van Cittert-Zernike theorem (see Sec. 2.3.8). However, higher fluxes are desirable for ptychographic imaging at higher resolution or at higher speeds. Furthermore, coherent diffraction imaging usually requires a quasi-monochromatic spectrum to reach temporal coherence and to ensure speckle contrast off the optical axis (see Sec. 2.3.2). Again, photon quality conflicts with photon quantity when temporal coherence is achieved via strong spectral filtering of the primary light source.

This chapter details the consequences when the spectral filtering of the incoming beam is reduced, or in other words, when ptychographic imaging is attempted with a polychromatic illumination of broad bandwidth.

The case of partial temporal coherence, as produced by a broad-bandwidth x-ray source, requires a different model than partial spatial coherence. As outlined in the conclusion of the theory chapter (Sec. 2.5) the polychromatic model for ptychography is written as

$$I_\mathrm{j}(\mathbf{s}) = \int_{-\infty}^{\infty} \left|\mathcal{P}(\lambda)\left\{p(\lambda,\mathbf{x})\cdot o(\lambda,\mathbf{x}-\mathbf{x}_j)\right\}\right|^2 \mathrm{d}\lambda\,, \tag{5.1}$$

which explicitly assumes that probe, object and the propagator (free space) can be dispersive, i.e. wavelength dependent. The numerical forward model bins the integral to a sum,

$$I_\mathrm{j}(\mathbf{s}) = \sum_\lambda \left|\mathcal{P}_\lambda\left\{p_\lambda(\mathbf{x})\cdot o_\lambda(\mathbf{x}-\mathbf{x}_j)\right\}\right|^2\,, \tag{5.2}$$

Though spectral shares are independent in a statistical sense, we will denote o_λ and p_λ as *chromatic* or *spectral modes* here. We have to keep in mind that these modes do not compose a set of eigenvectors for a second order correlation function.

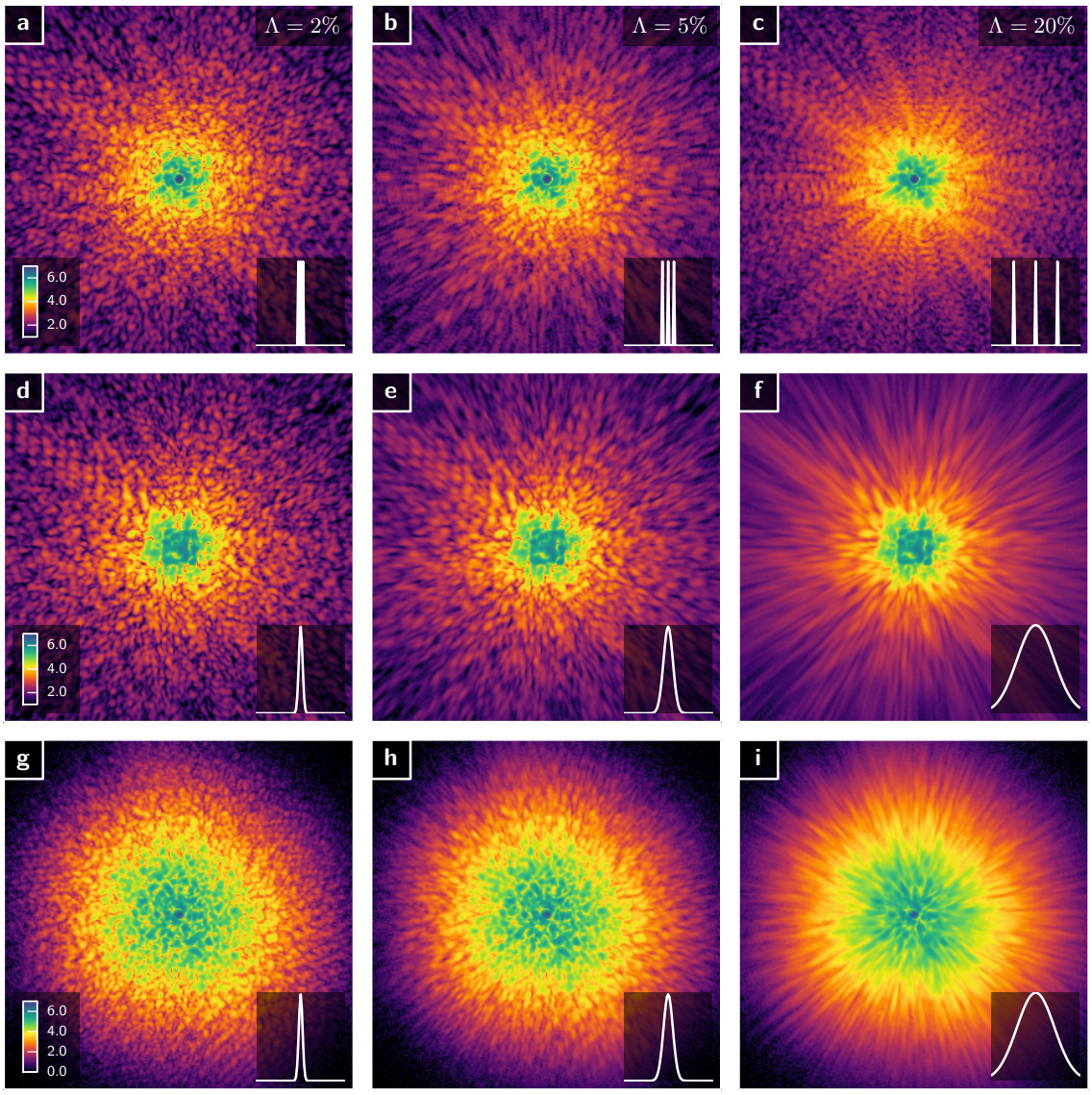


Figure 5.1. – Simulated diffraction images for three combinations of probe, object and spectrum. The columns represent the relative bandwidths $\Lambda = 2, 5$ and 20% used in the simulation, and the spectrum is displayed schematically in the lower right corner of each image. In the first row, **a-c**, a pinhole probe (see Fig. 5.3) is used to illuminate a binary object (see Fig. 5.7) with a spectrum consisting of three equidistant lines. The second row **d-f**, simulates a defocussed KB-mirror probe (see Fig. 5.4) and a Gaussian spectrum. The last row uses the same probe type as the first row, but with a Gaussian spectrum and a smoothed random phase sample (see Fig. 5.5). All images show the same field of view in reciprocal space and intensities are scaled with the decade logarithm.

For comparison of the spectral characteristics of an experiment, we recall the *relative bandwidth* (see eq.(2.91)),

$$\Lambda = \frac{\Delta\lambda}{\bar{\lambda}} = \frac{\Delta\nu}{\bar{\nu}}\,, \tag{5.3}$$

where $\Delta\lambda$ and $\Delta\nu$ are wavelength and frequency bandwidth (here FWHM) of the spectrum set in relation to $\bar{\lambda}$ and $\bar{\nu}$ which are center wavelength and center frequency respectively. Common values (Als-Nielsen and McMorrow, 2011) for relative bandwidths are

$\Lambda \approx 1.4 \cdot 10^{-4}$ for Si(111) double crystal monochromator,
$\Lambda = \frac{1}{nN} \approx 10^{-2}$ for the nth harmonic of a N-segment undulator, and
$\Lambda \approx 10^{-1}$ for single-color LEDs.

Apparently, two orders of magnitude in photon flux can be gained by omitting the monochromator behind the undulator. Figure 5.1 compares the far-field diffraction patterns for narrowband and broadband illumination of various types of probe and object. It shows that, in the far-field, mostly high angle diffraction is subject to deterioration from a broadband spectrum of the source.

The previous section 4.1 about ptychography without monochromator already demonstrated that quasi-monochromatic illumination is not needed for successful reconstruction (Enders et al., 2014). The question is to what extent and under what conditions the relative bandwidth Λ can be increased while still being able to recover good quality images of object and probe.

Section 5.1 derives a *polychromatic error* to quantify the deterioration from different types of broad spectrum. Furthermore, it analyses which of the three dispersion relations in (5.1) is dominant, i.e. contributes the most to the error, and which dispersion relation may be neglected.

Section 5.3 discusses algorithmic models for broadband ptychography. Different models are tested on a selection of dispersive and less dispersive probe and objects.

Eventually, section 5.4 reports about a selection of experiments as proof of concept. The algorithms are tested with experimental data from light sources of various bandwidth, from a narrow band laser to an ultra-broadband white LED.

5.1. Deterioration of diffraction contrast

Figure 5.1 shows that the speckle contrast in the diffraction image is reduced when using broadband source. This section attempts to quantify this deterioration of data quality for light sources with *symmetric* spectra.

A symmetric spectrum $\xi(\lambda)$ centred around wavelength $\bar{\lambda}$ has the properties

$$\int \xi(\lambda)\mathrm{d}\lambda = \mathsf{N}_{\mathrm{ph}} \qquad \text{and} \qquad \xi(\lambda - \bar{\lambda}) = \xi(\bar{\lambda} - \lambda) \tag{5.4}$$

where N_{ph} denotes the total number of photons. A natural candidate for such a spectrum is a Gaussian

$$\xi_g(\lambda) = \frac{\mathsf{N}_{\mathrm{ph}}}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(\lambda-\bar{\lambda})^2}{2\sigma^2}\right] \tag{5.5}$$

$$= \frac{2\mathsf{N}_{\mathrm{ph}}}{\Delta\lambda}\sqrt{\frac{\ln 2}{\pi}} \exp\left[-4\ln 2\frac{(\lambda-\bar{\lambda})^2}{\Delta\lambda^2}\right] \tag{5.6}$$

where $\Delta\lambda$ is the FWHM of the spectrum. Another interesting candidate is the multi-line spectrum of R spectral lines containing $\mathsf{N}_{\mathrm{ph}}/\mathsf{R}$ photons

$$\xi_{\mathsf{R}}(\lambda) = \frac{\mathsf{N}_{\mathrm{ph}}}{\mathsf{R}} \sum_{\mathsf{r}=0}^{\mathsf{R}-1} \delta\left[\lambda - \bar{\lambda} + \Delta\lambda\left(\frac{\mathsf{r}}{\mathsf{R}-1} - \frac{1}{2}\right)\right] \tag{5.7}$$

where the individual lines are equally spaced by $\frac{\Delta\lambda}{\mathsf{R}-1}$ and δ denotes the *delta*-function. The two-line spectrum $\xi_2(\lambda)$ and the box spectrum $\xi_\infty(\lambda)$ are the two extreme cases:

$$\xi_2(\lambda) = \frac{\mathsf{N}_{\mathrm{ph}}}{2}\,\delta\left[\lambda - \bar{\lambda} - \frac{\Delta\lambda}{2}\right] + \frac{\mathsf{N}_{\mathrm{ph}}}{2}\,\delta\left[\lambda - \bar{\lambda} + \frac{\Delta\lambda}{2}\right] \tag{5.8}$$

$$\xi_\infty(\lambda) = \frac{\mathsf{N}_{\mathrm{ph}}}{\Delta\lambda}\,\mathrm{rect}\left[\frac{\lambda-\bar{\lambda}}{\Delta\lambda}\right]. \tag{5.9}$$

In order to isolate the effect of the spectrum, we extract the spectral power distribution of the source $\xi(\lambda) = \|p(\lambda,\mathbf{x})\|^2_{\mathbf{x}}$ from the probe and rewrite (5.1) in the following way:

$$I_{\mathrm{j}}(\mathbf{s}) = \int_{-\infty}^{\infty} \xi(\lambda) I_{\mathrm{j}}(\lambda,\mathbf{s})\mathrm{d}\lambda \tag{5.10}$$

where

$$I_{\mathrm{j}}(\lambda,\mathbf{s}) = |F_{\mathrm{j}}(\lambda,\mathbf{s})|^2 = \left|\mathcal{P}(\lambda)\left\{p(\lambda,\mathbf{x})\, o(\lambda,\mathbf{x}-\mathbf{x}_j)\right\}(\mathbf{s})\right|^2 \tag{5.11}$$

is the spectrum-specific diffraction contrast of the system. For a monochromatic system, the spectrum is a single line, $\xi(\lambda) = \mathsf{N}_{\mathrm{ph}}\delta(\lambda - \bar{\lambda})$, and the monochromatic diffraction contrast is therefore given by

$$I_{\mathrm{j}}^{\mathrm{mono}}(\mathbf{s}) = \int_{-\infty}^{\infty} \mathsf{N}_{\mathrm{ph}}\delta(\lambda - \bar{\lambda}) I_{\mathrm{j}}(\lambda, \mathbf{s}) \mathrm{d}\lambda = \mathsf{N}_{\mathrm{ph}} I_{\mathrm{j}}(\bar{\lambda}, \mathbf{s})\,. \tag{5.12}$$

With the L^2-norm, we define a relative *"polychromatic" error* as the difference of the polychromatic diffraction contrast to the monochromatic contrast, relative to the norm of the monochromatic contrast:

$$\epsilon_{\mathrm{poly}} \equiv \frac{\|I_{\mathrm{j}}^{\mathrm{mono}}(\mathbf{s}) - I_{\mathrm{j}}(\mathbf{s})\|_{\mathrm{j},\mathbf{s}}}{\|I_{\mathrm{j}}^{\mathrm{mono}}(\mathbf{s})\|_{\mathrm{j},\mathbf{s}}} \tag{5.13}$$

where the norm is taken over all $(j, \mathbf{s})$. For any reasonable diffraction setup, the relative error should be within the interval $[0, 1]$. In general, the effect of partial temporal coherence should resemble a blurring of speckles as depicted in Fig. 5.1. When a reduction of speckle contrast is the predominant effect of a broad bandwidth, the polychromatic error ϵ_{poly} will be bound by 0.5.

If we assume that the spectral bandwidth $\Delta\lambda$ is small with respect to the center wavelength $\bar{\lambda}$ we can replace $I_{\mathrm{j}}(\lambda, \mathbf{s})$ by a Taylor approximation

$$I_{\mathrm{j}}(\lambda, \mathbf{s}) = I_{\mathrm{j}}(\bar{\lambda}, \mathbf{s}) + (\lambda - \bar{\lambda})\frac{\mathrm{d}I_{\mathrm{j}}}{\mathrm{d}\lambda}(\bar{\lambda}, \mathbf{s}) + \frac{(\lambda - \bar{\lambda})^2}{2}\frac{\mathrm{d}^2 I_{\mathrm{j}}}{\mathrm{d}\lambda^2}(\bar{\lambda}, \mathbf{s}) + \mathcal{O}(\lambda^3) \tag{5.14}$$

and insert this approximation in the integral in (5.10).

$$I_{\mathrm{j}}(\mathbf{s}) = I_{\mathrm{j}}(\bar{\lambda}, \mathbf{s}) \underbrace{\int_{-\infty}^{\infty} \xi(\lambda)\mathrm{d}\lambda}_{=\mathsf{N}_{\mathrm{ph}}} + \frac{\mathrm{d}^2 I_{\mathrm{j}}}{\mathrm{d}\lambda^2}(\bar{\lambda}, \mathbf{s}) \underbrace{\int_{-\infty}^{\infty} \frac{(\lambda - \bar{\lambda})^2}{2}\xi(\lambda)\mathrm{d}\lambda}_{\equiv 0.5 S(\xi)} + \mathcal{O}((\lambda - \bar{\lambda})\lambda^4)\,. \tag{5.15}$$

The odd powers of the Tailor expansion cannot contribute due to symmetry of the spectrum. Up to second order, the polychromatic error may thus be written as

$$\epsilon_{\mathrm{poly}} \approx \frac{S(\Delta\lambda)}{2\|I_{\mathrm{j}}^{\mathrm{mono}}(\mathbf{s})\|_{\mathrm{j},\mathbf{s}}} \left\| \frac{\mathrm{d}^2 I_{\mathrm{j}}}{\mathrm{d}\lambda^2}(\bar{\lambda}, \mathbf{s}) \right\|_{\mathrm{j},\mathbf{s}} \tag{5.16}$$

with S being the variance of ξ. While the variance of a Gaussian distributed spectrum

Spectrum ξ	Variance $S(\xi)/P$
Gaussian	$\Delta\lambda^2/(8\ln 2)$
Box	$\Delta\lambda^2/12$
2-line	$\Delta\lambda^2/4$
3-line	$\Delta\lambda^2/6$

Table 5.1. – Variances of various spectra with the same FWHM $\Delta\lambda$

is well-known, the variance of a poly-line spectrum is a little more evolved:

$$S(\xi_{\mathsf{R}}) = \frac{\mathsf{N}_{\mathrm{ph}}}{\mathsf{R}} \int_{-\infty}^{\infty} \sum_{\mathsf{r}=0}^{\mathsf{R}-1} \delta\left[\lambda - \bar{\lambda} + \Delta\lambda\left(\frac{\mathsf{r}}{\mathsf{R}-1} - \frac{1}{2}\right)\right](\lambda-\bar{\lambda})^2\,\mathrm{d}\lambda \tag{5.17}$$

$$= \mathsf{N}_{\mathrm{ph}}\frac{\Delta\lambda^2}{\mathsf{R}} \sum_{\mathsf{r}=0}^{\mathsf{R}-1}\left(\frac{\mathsf{r}^2}{(\mathsf{R}-1)^2} - \frac{\mathsf{r}}{\mathsf{R}-1} + \frac{1}{4}\right) \tag{5.18}$$

$$= \mathsf{N}_{\mathrm{ph}}\frac{\Delta\lambda^2}{\mathsf{R}}\left(\frac{\mathsf{R}(\mathsf{R}-1)(\mathsf{R}-2)}{6}\frac{1}{(\mathsf{R}-1)^2} - \frac{1}{\mathsf{R}-1}\frac{\mathsf{R}(\mathsf{R}-1)}{2} + \frac{\mathsf{R}}{4}\right) \tag{5.19}$$

$$= \mathsf{N}_{\mathrm{ph}}\frac{\Delta\lambda^2}{12}\frac{\mathsf{R}+1}{\mathsf{R}-1}\,. \tag{5.20}$$

The table 5.1 summarizes the results. With $\sigma_\xi^2 \equiv S(\xi)/P/\Delta\lambda^2$ we obtain a final expression for ϵ_{poly}:

$$\epsilon_{\mathrm{poly}} \approx \frac{\sigma_\xi^2\,\Lambda^2}{2\|I_{\mathsf{j}}(\bar{\lambda},\mathbf{s})\|_{\mathsf{j},\mathbf{s}}}\left\|\bar{\lambda}^2\frac{\mathrm{d}^2 I_{\mathsf{j}}}{\mathrm{d}\lambda^2}(\bar{\lambda},\mathbf{s})\right\|_{\mathsf{j},\mathbf{s}}\,. \tag{5.21}$$

Eq. (5.21) would take exactly the same shape if the frequency ν was considered instead of wavelength *and* if the spectrum $\xi(\nu)$ were symmetric in frequency (or energy).

Further analysis requires knowledge of the actual diffraction geometry as well as characteristics about sample and object. However, since I_{j} is an intensity we can express $\frac{\mathrm{d}^2 I_{\mathsf{j}}}{\mathrm{d}\lambda^2}$ by the propagated scalar wave field F_{j} ,

$$\frac{1}{2}\frac{\mathrm{d}^2 I_{\mathsf{j}}}{\mathrm{d}\lambda^2}(\lambda,\mathbf{s}) = \mathrm{Re}\left\{F_{\mathsf{j}}^*(\lambda,\mathbf{s})\frac{\mathrm{d}^2 F_{\mathsf{j}}}{\mathrm{d}\lambda^2}(\lambda,\mathbf{s})\right\} + \left|\frac{\mathrm{d}F_{\mathsf{j}}}{\mathrm{d}\lambda}(\lambda,\mathbf{s})\right|^2\,. \tag{5.22}$$

We see that both linear and quadratic terms of the propagated complex wave fields contribute to the second order of the intensity. A special case is given when both probe and object are non-dispersive, i.e. $p(\mathbf{x}) = p(\lambda,\mathbf{x})$ and $o(\mathbf{x}) = o(\lambda,\mathbf{x})$. Consequently, the exit waves $\psi_{\mathsf{j}}(\mathbf{x}) = p(\mathbf{x})\cdot o(\mathbf{x}-\mathbf{x}_j)$ are non-dispersive, too. A deviation from a monochromatic response of the system is then introduced only through the propagator. In case of a far-field propagator, we see that

$$F_{\mathsf{j}}(\lambda,\mathbf{s}) = \mathcal{P}(\lambda)\{\psi_{\mathsf{j}}(\mathbf{x})\}(\lambda,\mathbf{s}) = \mathcal{F}\psi_{\mathsf{j}}\left(\frac{\mathbf{s}}{\lambda \mathsf{z}}\right)\,, \tag{5.23}$$

meaning that an off-centred spectral component simply produces a centrically stretched or compressed replica of the central wavelength. The derivatives are first and second order directional derivatives,

$$\left|\frac{\mathrm{d}F_\mathrm{j}}{\mathrm{d}\lambda}\right|^2 = \frac{1}{\lambda^4 z^2}\left|\mathbf{s}^T\nabla\mathcal{F}\psi_\mathrm{j}\right|^2 \tag{5.24}$$

$$\frac{\mathrm{d}^2F_\mathrm{j}}{\mathrm{d}\lambda^2} = \frac{2}{\lambda^3 z}\,\mathbf{s}^T\nabla\mathcal{F}\psi_\mathrm{j} + \frac{1}{\lambda^4 z^2}\,\mathbf{s}^T\mathcal{H}(\mathcal{F}\psi_\mathrm{j})\mathbf{s}\,, \tag{5.25}$$

weighted with the (quadratic) distance $|\mathbf{s}|$ from the optical axis. $\mathcal{H}$ is the Hessian matrix $\mathcal{H}(f)_{a,b} = \partial_a\partial_b f$, $a, b \in \{\mathrm{I}, \mathrm{II}\}$. This means that those geometries or experiments are particularly affected, that scatter to large angles *and* provide a fine speckle structure. These are imaging conditions of high space-bandwidth product DQ, as the speckle size scales inverse with D. Especially any setup that uses an illuminated pinhole to provide the probe falls into this category, as the probe provides strong contrast (= high angle scattering) at large separation (= small speckle size, large gradient).

5.1.1. Example: Two point sources in the far-field

From eq. (5.25) we conclude that calculating the second order dependence of the intensity on the spectrum requires precise knowledge on probe and object and may often not be possible analytically. However, one example where an analytic expression can be found with reasonable effort is the diffraction from two pinholes. The exit wave ψ for a uniform illumination can be written as

$$\psi(\mathbf{x}) = \frac{1}{2}\left(\delta\left[\mathbf{x} - (D/2, 0)\right] + \delta\left[\mathbf{x} + (D/2, 0)\right]\right)\,. \tag{5.26}$$

The propagated wave front is a cosine function in one direction,

$$F(\lambda, \mathbf{s}) = \mathcal{F}\psi\left(\frac{\mathbf{s}}{\lambda z}\right) = \cos\left(\frac{\pi D s_\mathrm{I}}{\lambda z}\right)\,, \tag{5.27}$$

with $\mathbf{s} = (s_\mathrm{I}, s_\mathrm{II})$. The short form $\eta(\lambda, \mathbf{s}) = \frac{\pi D s_\mathrm{I}}{\lambda z}$ simplifies calculation of the derivatives,

$$\bar{\lambda}^2\left|\frac{\mathrm{d}F}{\mathrm{d}\lambda}\right|^2(\bar{\lambda}, \mathbf{s}) = \eta^2\sin^2(\eta) \tag{5.28}$$

$$\bar{\lambda}^2 F^*\frac{\mathrm{d}^2F}{\mathrm{d}\lambda^2}(\bar{\lambda}, \mathbf{s}) = -2\eta\sin(\eta)\cos(\eta) - \eta^2\cos^2(\eta)\,, \tag{5.29}$$

which are inserted into the numerator of the polychromatic error (5.21):

$$\frac{\bar{\lambda}^4}{4}\left\|\frac{\mathrm{d}^2 I}{\mathrm{d}\lambda^2}\left(\bar{\lambda},\mathbf{s}\right)\right\|_{\mathbf{s}}^2 = \int_{\mathcal{S}} \eta^4 \left[\cos^2(2\eta) + \frac{\sin(4\eta)}{\eta} + \frac{\sin^2(2\eta)}{(\eta)^2}\right] \mathrm{d}\mathbf{s}\,, \tag{5.30}$$

where $\mathcal{S} = [-S, S] \times [-S, S]$ denotes the domain of the detection screen. We can safely assume that a few periods of interference fringes are recorded. Since η can be roughly interpreted as the number of fringes, we assume $\eta \gg 1$. In this case, the first summand, $\cos^2$, in the integrand of (5.30) is dominant. Additionally, we approximate $\cos^2 \approx 0.5$ and obtain,

$$\frac{\bar{\lambda}^4}{4}\left\|\frac{\mathrm{d}^2 I}{\mathrm{d}\lambda^2}\left(\bar{\lambda},\mathbf{s}\right)\right\|_{\mathbf{s}}^2 \gtrsim 2S \int_{-S}^{S} \left[\frac{\eta^4}{2}\right] \mathrm{d}s_{\mathrm{I}} = \frac{2}{5} S^2 \left(\frac{\pi D S}{\bar{\lambda}\mathrm{z}}\right)^4 . \tag{5.31}$$

Similarly, we find the following approximation for the denominator,

$$\left\|I(\bar{\lambda},\mathbf{s})\right\|_{\mathbf{s}}^2 = \int_{\mathcal{S}} \cos^4(\eta)\mathrm{d}\mathbf{s} = \int_{\mathcal{S}} \left[\frac{1}{4} + \frac{1}{2}\cos(2\eta(\mathbf{s})) + \frac{1}{8}\left[1 + \cos(4\eta(\mathbf{s}))\right]\right] \mathrm{d}\mathbf{s} \tag{5.32}$$

$$\approx 2S \int_{-S}^{S} \frac{3}{8} \mathrm{d}s_{\mathrm{I}} = \frac{3}{2} S^2 . \tag{5.33}$$

Concluding, the polychromatic error for two point sources may be estimated as

$$\epsilon_{\mathrm{poly}} \gtrsim \frac{2\pi^2}{\sqrt{15}} \sigma_\xi^2 \lambda^2 (D\mathrm{Q})^2 \approx 5.1\, \sigma_\xi^2\, \Lambda^2\, (D\mathrm{Q})^2 . \tag{5.34}$$

Pfeiffer et al. (2007) derived a criterion for two secondary waves separated by D and wavelength $\Delta\lambda$. Just like we did in 2.2 they compared the path difference at observation angle of maximum space frequency $Q = 1/\mathrm{d_x}$ with the longitudinal coherence length. Their choice $\ell_{\mathrm{coh}} = \bar{\lambda}/(2\Lambda)$ corresponds to the first distance of destructive interference of the two waves. Eventually they found the upper limit of tolerable relative bandwidth to be,

$$\Lambda < \frac{1}{2DQ}\,, \tag{5.35}$$

which is a bit stricter than (2.90). If we insert the coefficient $\sigma_\xi^2 = 0.25$ of a 2-line spectrum into (5.34) we obtain

$$\epsilon_{\mathrm{poly}} \approx \Lambda^2 (DQ)^2 \lesssim \frac{1}{4} \tag{5.36}$$

as overall polychromatic error for the same geometry. The result makes sense as the error assumes the mean value 0.5 in the periphery where the waves are considered completely dephased. $\epsilon_{\mathrm{poly}} \approx 0.25$ corresponds then to the mean value for linear

transition from the central region with no deterioration to the periphery. A polychromatic error of 0.25 is rather large as it means that contrast in the diffraction image is lost in an equivalent area of half the field of view in reciprocal space. In general, we should expect difficulties much earlier, which means even for smaller probe sizes. However, high spatial frequencies have a lower impact on the error as the intensity declines usually faster than $\mathbf{q}^2$ which means that high spatial frequencies take a smaller fraction of the power distribution even though they take a larger area in reciprocal space. Regardless of the value of the error, a monochromatic reconstruction from polychromatic data is expected to show erratic behaviour predominantly for the high spatial frequencies, i.e. on small length scales.

Figure 5.2 shows the ϵ_{poly} for three different symmetric spectra. The quadratic dependence of ϵ_{poly} is apparent for all spectra for $\Lambda < 10\%$. As predicted by the variance calculations from before (Tab. 5.1), the box spectrum has the smallest impact on the polychromatic error. All curves exhibit a linear phase in the region $\Lambda \approx 10\% - 20\%$ and a tendency to a saturation curve when approaching $\Lambda \approx 30\%$ – an expected behaviour as ϵ_{poly} should be bound by 0.5. Regions with $\Lambda > 30\%$ were beyond the scope of this simulations which used 76 equidistant wavelengths to cover a bandwidth of 40%. In particular, the strong decrease in slope for the Gaussian spectrum for $\Lambda > 20\%$ is due to the tails of the spectrum being excluded from the simulation.

Furthermore, Fig. 5.2d shows the distribution of ϵ_{poly} with distance from optical axis $|\mathbf{s}|$, which in turn represents the specific spatial frequency $|\mathbf{s}|/(\bar{\lambda}z)$. We can readily observe, that the error accumulates first at high scattering angles, meaning that broadband deterioration affects the resolution immediately. Other than expected, the plots do not saturate at a value of 0.5 but lean to a slightly larger value. This is due to the general property of far-field intensity to decrease rapidly with scattering angle. Due to the stretching characteristic of the propagator (see eq.(5.25)), intensity from the intense central regions of the diffraction share of the larger wavelengths is spread to larger angles, yielding a net increase in intensity when compared to the monochromatic diffraction intensity.

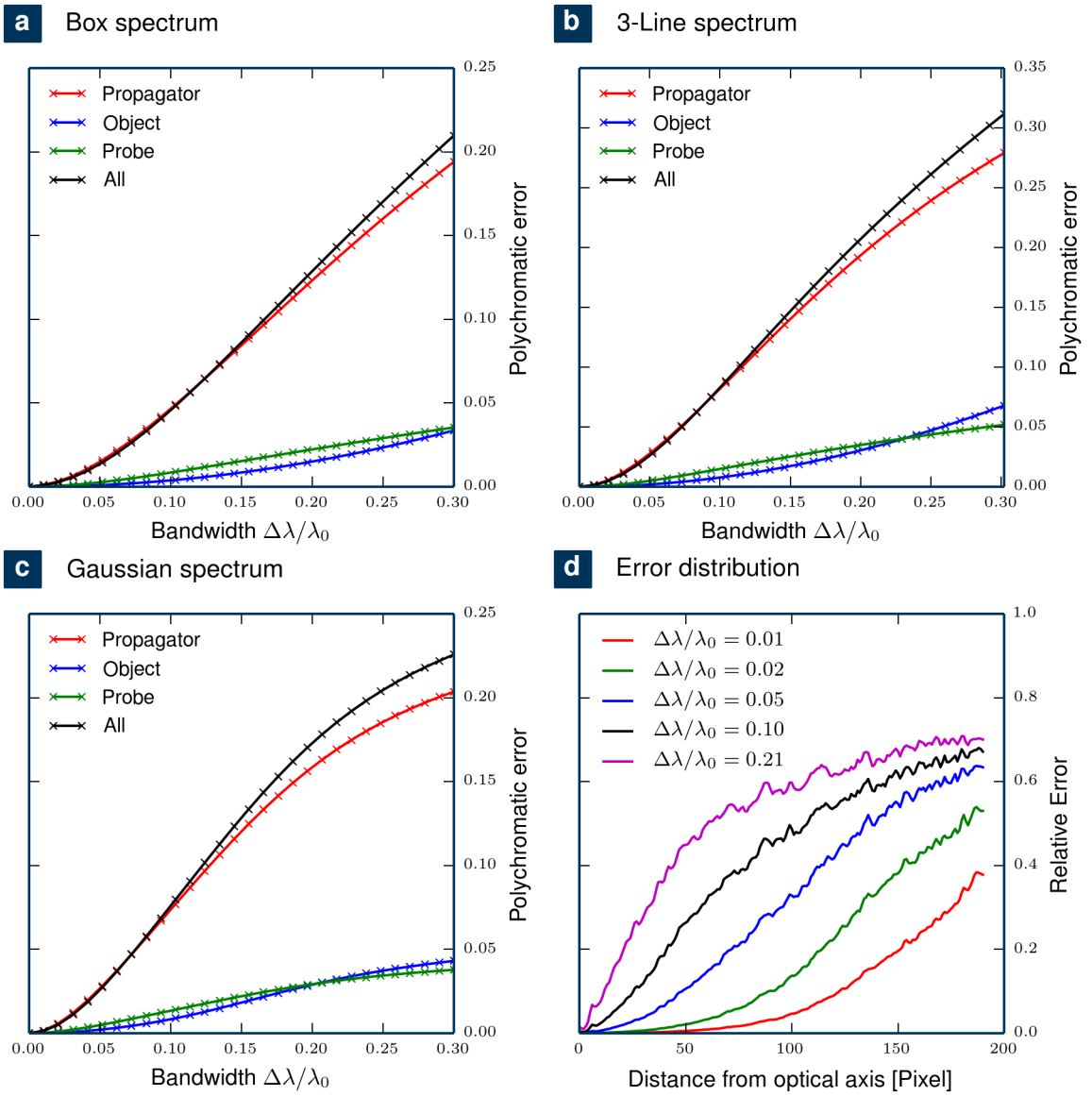


Figure 5.2. – Relation between polychromatic error and relative bandwidth for a box (**a**), a Gaussian (**b**) and a 3-line spectrum (**c**). The red, blue and green plots represent cases where, respectively, only one of propagator, object or probe is dispersive. The black curve represents the case where all components are dispersive. Panel **d** displays the distribution of the error with scattering angle. The probe was a propagated pinhole as depicted in Fig. 5.3 and the object was mainly a phase shifting variant as depicted in Fig. 5.5.

5.2. Implementation of spectral propagation

A far-field propagator implemented via a discrete Fourier transform imposes constraints (sec. 2.2.2) on the sampling in specimen plane and observation plane,

$$\triangle x \; \triangle s = \frac{\bar{\lambda}\mathsf{z}}{\mathsf{N}} \,. \tag{5.37}$$

As the detector grid spacing $\triangle s$ is fixed, imaging at a different wavelength $\bar{\lambda}+\triangle\lambda$ must be compensated by a variable number of samples $\mathsf{N}+\triangle\mathsf{N}$ in order to keep grid spacing $\triangle x$ in the sample plane constant.

$$\triangle x \, \triangle s = \frac{\bar{\lambda}\mathsf{z}}{\mathsf{N}} = \frac{\mathsf{z}(\bar{\lambda}+\triangle\lambda)}{\mathsf{N}+\triangle\mathsf{N}} \,. \tag{5.38}$$

As a consequence, the array containing the sampled exit wave for a specific $\bar{\lambda}\pm\triangle\lambda$ has to be cropped or padded by zeros prior to the propagation by $\pm\mathsf{N}\,\triangle\lambda/\bar{\lambda}$. The inverse operation has to be applied after numerical propagation.

Since the smallest, symmetric cropping or padding is one line on each boundary, it imposes the constraint $\triangle\mathsf{N} \in 2\mathbb{Z}$ which means that the minimum distance in spectral sampling of the propagator is given by

$$\triangle\lambda_{\mathrm{min}} = \frac{2\bar{\lambda}}{\mathsf{N}} \,. \tag{5.39}$$

This constraint is specific to a propagator that uses a single Fourier transform for propagation (like (2.34) and (2.36)). Near-field propagation as done by Stockmar et al. (2013) and implemented with an angular spectrum propagator (2.25) does not necessarily have a lower limit on spectral sensitivity.

5.3. Polychromatic models for ptychography

As discussed in Sec. 2.3.4, the frequency components of the cross-spectral density contribute *independently* to the diffraction signal. Discretising 5.1 in small steps $\triangle\lambda$ creates spectral "modes" p_λ, o_λ which are statistically independent. That means that the concept of limited rank for an eigenvalue decomposition of a partial spatial coherent probe does not apply here. Enabling many spectral modes may therefore result in the problem that too many degrees of freedom were added to the algorithm which, in turn, facilitate an early convergence to a non-unique solution. Therefore, a sensible algorithmic implementation of broadband ptychography has to work even for monochromatic data by not populating the additional (superfluous) spectral modes.

For most experimental setups, it is safe to assume that each spectral component of illumination passes the same set of optics and propagates the same distance. Similarly, the sample may be imaged in a regime where its dispersion relation follows a simple polynomial or another analytical expression. This means, that even though the mutual coherence puts no constraints on neighbouring frequencies in a polychromatic light field, the frequency component of probe and object may be correlated at the sample plane in a specific way. Hence, the limits

$$\lim_{\lambda\to\bar{\lambda}} p(\lambda,\mathbf{x}) = p(\bar{\lambda},\mathbf{x}) \qquad \text{and} \qquad \lim_{\lambda\to\bar{\lambda}} o(\lambda,\mathbf{x}) = o(\bar{\lambda},\mathbf{x}) \tag{5.40}$$

coincide with components of probe and object of the mean wavelength. This continuity was silently assumed in the previous section as otherwise the exchange of differentiation and integration would have been a formal error.

For the paraxial propagator and any illumination that has passed a pinhole or linear optics, condition (5.40) is true. The limit also hold for the specimen if the light consists of X-rays and the spectrum does not cover any resonant interaction regime (i.e. being far away from absorption edges). In these cases, we can further assume, that the energy dependence of sample or object results in additive terms in eq.(5.25). Upon comparison of the individual error contributions from probe, object or propagator we may find that one of the error sources is negligible. This is equivalent to either probe, object or propagator being non-dispersive and interchangeable with their monochromatic counterparts. The 'monochromatic', non-dispersive probe or object is then shared among all frequencies and and imposes a strong constraint. It forces the other frequencies to communicate with each other at the shared non-dispersive quantity, thus reducing the number of possible solutions and creating a path to a unique solution within the polychromatic model *without additional a priori information*. The forward model for polychromatic ptychography with a non-dispersive object or with an achromatic probe is written as

$$I_\mathrm{j}(\mathbf{s}) = \sum_\lambda \mathcal{P}_\lambda \left\{ p_\lambda(\mathbf{x}) o(\mathbf{x}-\mathbf{y}_\mathrm{j}) \right\} \quad \text{and} \quad I_\mathrm{j}(\mathbf{s}) = \sum_\lambda \mathcal{P}_\lambda \left\{ p(\mathbf{x}) o_\lambda(\mathbf{x}-\mathbf{y}_\mathrm{j}) \right\} , \tag{5.41}$$

respectively. Eq. (5.41) has an additional side benefit. Due to the distance-wavelength equivalence of the paraxial propagator, it can also be interpreted as an incoherent sum of different propagation distances z, i.e.

$$I_\mathrm{j}(\mathbf{s}) = \sum_\mathrm{z} \mathcal{P}_\mathrm{z} \left\{ p_\mathrm{z}(\mathbf{x}) o(\mathbf{x}-\mathbf{y}_\mathrm{j}) \right\} . \tag{5.42}$$

As a consequence, an algorithm that solves for chromatic modes is also suited to determine the free space propagation distance.

From eq.(5.41) we immediately observe a possible pitfall. If for any two neighbouring wavelengths the propagator was not dispersive enough, the forward model would

resemble exactly the model of the mode expansion for partial spatial coherence of either probe or object. As an extreme case, the forward model could take on the shape,

$$I_{\mathrm{j}}(\mathbf{s}) = \sum_{\lambda} \mathcal{P}_{\bar{\lambda}} \left\{ p_{\lambda}(\mathbf{x}) o(\mathbf{x} - \mathbf{y}_{\mathrm{j}}) \right\} \quad \text{or} \quad I_{\mathrm{j}}(\mathbf{s}) = \sum_{\lambda} \mathcal{P}_{\bar{\lambda}} \left\{ p(\mathbf{x}) o_{\lambda}(\mathbf{x} - \mathbf{y}_{\mathrm{j}}) \right\} . \tag{5.43}$$

In that case the chromatic modes will lose their former meaning and become probe or object modes of partial spatial coherence. This means, that broadband ptychography with chemical sensitivity, i.e. imaging across absorption edges is not likely to work. We may also hypothesize that the dispersive nature of the paraxial propagator is critical to success of finding chromatic modes without a priori information. The propagator's sensitivity determines the spectral resolution $\triangle\lambda$ of the retrieved dispersion relation.

5.3.1. Simulation

Broadband simulations were conducted to verify, whether the approach of a dispersive probe with a non-dispersive object and vice-versa would yield acceptable reconstructions for a diverse set of probe-types and object-types.

Probes

Three different probes of the same effective diameter D were simulated. The dispersive or chromatic relation of the probe arises mostly from the free-space propagation path between beamline optics and specimen.

Pinhole in contact
Due to the absence of free space propagation, a pinhole close-to or in contact with the specimen was taken as a candidate for a mainly non-dispersive behaviour.

Propagated pinhole
The same pinhole but propagated for a distance of 2 mm into the Fresnel regime was used as a dispersive probe. Such a probe is very common for ptychography as a pinhole is easy to manufacture and an air gap usually separates probe and object. The dispersive behaviour of this probe is illustrated in Fig. 5.3. The wave field at center frequency (Fig. 5.3a) shows the characteristic circular interference fringes. Fig. 5.3b shows how these fringes change with wavelength. Starting in the Fresnel regime the dispersion of the paraxial propagator becomes equivalent to propagation. Hence, the dispersion relation appears like a propagation effect even though the same distance

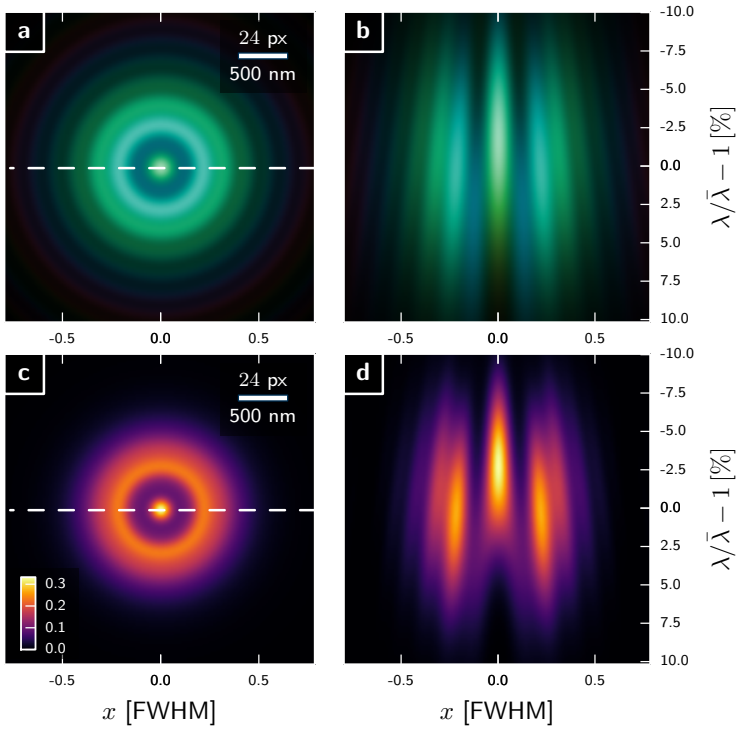


Figure 5.3. – Dispersion relation of a propagated pinhole as probe;
a Wave field image of the probe at center wavelength, $\mathrm{Img}(\mathbf{x}) = p(\mathbf{x}, \bar{\lambda})$.
The dashed line marks a cut (**b**) through the three-dimensional $(\mathbf{x}, \lambda)$-space of the probe's wave field:
$\mathrm{Img}(x, \lambda) = p((x, 0), \lambda)$.
c Probe intensity,
$\mathrm{Img}(\mathbf{x}) = \sum_\lambda \xi(\lambda)|p(\mathbf{x}, \lambda)|^2$
in 10^6 photons per pixel.
d Spectral density,
$\mathrm{Img}(x, \lambda) = \xi(\lambda)|p((x, 0), \lambda)|^2$
indicating that the photon distribution undergoes substantial changes with wavelength.

was applied for the near-field propagator. Fig. 5.3c & d display probe intensity and a cut through the spectral density (see (2.102)) respectively.

Defocussed diverging probe

This probe type is typical when focussing optics (KB-mirrors, FZP) shape the illumination. Here, a focal distance of 10 cm was assumed together with a square entrance aperture of 50 µm side length. The focussed probe was propagated 2 mm out of focus in order to provide the same FWHM as the other probes. The dispersive behaviour of this probe is illustrated in Fig. 5.4. The wave field at center frequency (Fig. 5.4a) has the characteristic quadratic phase factor for a diverging illumination. Fig. 5.4b reveals that wave front characteristics mainly rescale with wavelength or propagation distance. Fig. 5.4c and d display probe intensity and a cut through the spectral density (see (2.102)) respectively. The diverging probe appears smoother in its spectral density than the pinhole probe.

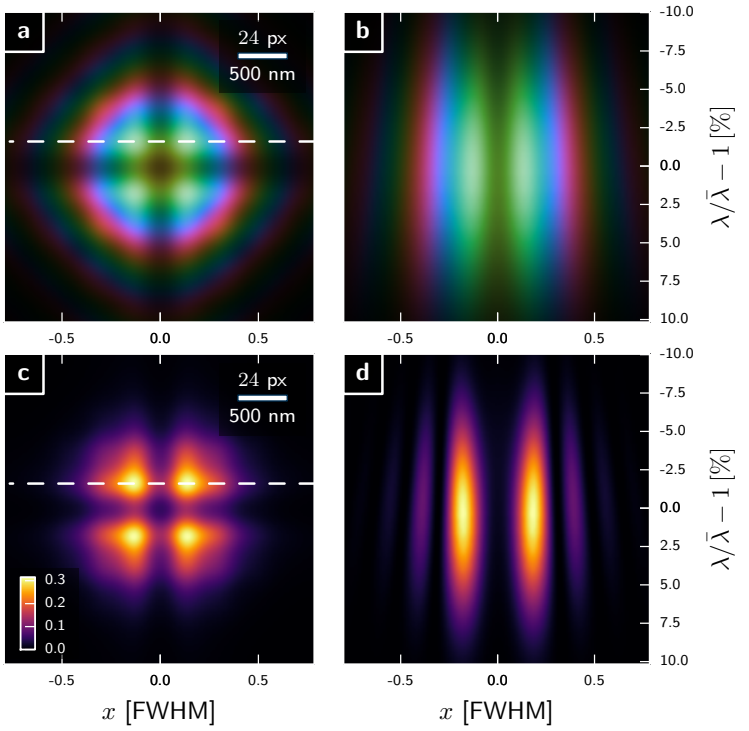


Figure 5.4. – Dispersion relation of a focussed diverging illumination as probe;
a Wave field image of the probe at center wavelength, $\mathrm{Img}(\mathbf{x}) = p(\mathbf{x}, \bar{\lambda})$. The dashed line marks a cut (**b**) through the three-dimensional $(\mathbf{x}, \lambda)$-space of the probe's wave field: $\mathrm{Img}(x, \lambda) = p((x, c), \lambda)$.
c Probe intensity, $\mathrm{Img}(\mathbf{x}) = \sum_\lambda \xi(\lambda)|p(\mathbf{x}, \lambda)|^2$ in 10^6 photons per pixel.
d Spectral density, $\mathrm{Img}(x, \lambda) = \xi(\lambda)|p((x, c), \lambda)|^2$
Aside from a resizing effect, the diverging illumination does not appear to change substantially with wavelength.

Objects

Three objects with different dispersion relations were simulated. As basis for all objects served an array of normal-distributed pseudo-random numbers. That array was filtered with a Gaussian blurring kernel to create a smoothed random surface $h(\mathbf{x})$.

Opaque object

For a mainly non-dispersive specimen, a binary, opaque transmission $b(\mathbf{x})$ was built through application of a threshold,

$$o(\mathbf{x}, \lambda) = b(\mathbf{x}) = \Theta(h(\mathbf{x}))\,. \tag{5.44}$$

Fig. 5.6 displays reconstructions of this object transmission at center wavelength.

Binary object

An object transmission of a more evolved dispersion, is a binary phase-shifting transmission. This transmission is very common for lithographically-made test objects, as used in sections 4.1 and 4.2.

$$o(\mathbf{x}, \lambda) = \exp\left[\frac{\mathrm{i}\pi}{2}\, b(\mathbf{x})\, \frac{\lambda}{\bar{\lambda}}\right]\,. \tag{5.45}$$

Fig. 5.7 displays reconstructions of this object transmission at center wavelength.

Phase object

When creating a strong continuous phase shifting object, the smallest feature in the transmission should be much smaller than the probe size in order to suppress refraction. However, at a constant phase gradient, smaller features imply smaller amplitudes in the phase profile and the amplitudes are expected to predominantly influence the dispersion relation of the transmission. Hence, a compromise has to be made. Here, the gradient of the random surface $h(\mathbf{x})$ was rescaled to an average phase gradient of 2π per 10 pixel,

$$\tilde{h}(\mathbf{x}) = h(\mathbf{x}) \frac{10 \triangle \mathbf{x}}{2\pi \langle |\nabla h(\mathbf{x})| \rangle_{\mathbf{x}}} \tag{5.46}$$

resulting in phase amplitudes of roughly $\pm\pi$ in the phase profile (see Fig. 5.5). As the specimen transmission should exhibit positive shifts in phase, the minimum value $\tilde{h}_{\min}$ of $\tilde{h}$ within the field of view served as reference for air. $\tilde{h} - \tilde{h}_{\min}$ then represents a projected density or thickness profile. The entire dispersive object was built from

$$o(\mathbf{x}, \lambda) = \exp\left[\frac{\mathrm{i}\lambda}{\bar{\lambda}} (\tilde{h}(\mathbf{x}) - \tilde{h}_{\min})\right] . \tag{5.47}$$

Fig. 5.5 summarises the dispersion relation of the random phase object.

Geometry and scan pattern

The simulations used cSAXS parameters as setup geometry. It means that the distance from detector to sample was $\mathrm{z} = 7.0\,\mathrm{m}$ and the design mean wavelength was $\bar{\lambda} = 0.20\,\mathrm{nm}$ (mean energy $\bar{E} = 6.2\,\mathrm{keV}$). Even the typical PILATUS detector pixel size of $\triangle s = 172\,\mu\mathrm{m}$ was adopted although the assumed gap free module of $\mathsf{N} \times \mathsf{N} = 384 \times 384$ pixel does not exist. As a result, the pixel size is $\triangle x = 21.2\,\mathrm{nm}$ in the sample plane which corresponds to a resolution of $23.5\,\mu\mathrm{m}^{-1}$. Shannon sampling would support probe sizes up to $D = 192 \cdot 21.2\,\mathrm{nm}$ but in view of shorter wavelengths and eventual tails of the probes, a FWHM of $D = 96 \cdot 21.2\,\mathrm{nm} \approx 2\,\mu\mathrm{m}$ was chosen for all probes. The simulation is transferable to other imaging regimes having the same Fresnel number

$$\mathrm{F} = \frac{D^2}{4\bar{\lambda}\mathrm{z}} = 0.003 \tag{5.48}$$

for as long as the total number of samples per dimension in propagated space is roughly 400 and the probe diameter is covered by 100 samples in each direction.

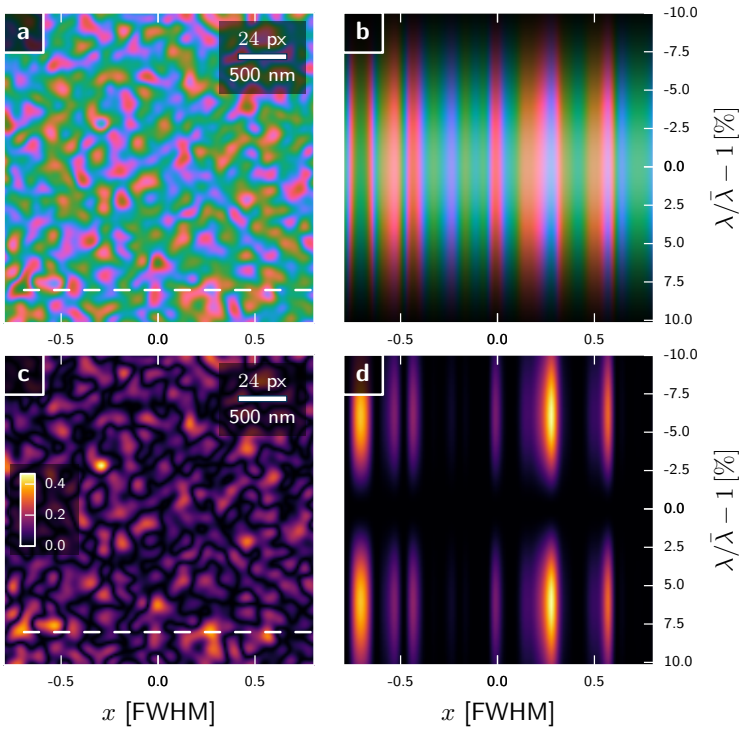


Figure 5.5. – Dispersion relation of a random phase sample as object:
a Wave field image of the object at center wavelength, $\mathrm{Img}(\mathbf{x}) = p(\mathbf{x}, \bar{\lambda})$.
The dashed line marks a cut at distance c (panel **b**) through the three-dimensional $(\mathbf{x}, \lambda)$-space of the object's wave field: $\mathrm{Img}(x, \lambda) = o(x, y_\mathrm{c}, \lambda)$.
c Averaged deviation (nrmse) of dispersion including spectral weights, $\mathrm{Img}(\mathbf{x}) = \|\sqrt{\xi(\lambda)}(o(\mathbf{x}, \lambda) - o(\mathbf{x}, \bar{\lambda}))\|_\lambda$
d Euklidean distance of spectral modes to the main wavelength, $\mathrm{Img}(x, \lambda) = \xi(\lambda)|o(x, c, \lambda) - o(x, c, \bar{\lambda})|^2$.

All simulations scanned on a round scan pattern of equidistant shells of 326 nm which corresponds to 4 % of the probe field of view and 16 % of the probe FWHM, resulting in a high overlap. To keep the reconstruction time short, only 161 scan positions in a square field of view of 4.62 µm were selected from the pattern.

At each scan point, broad-bandwidth diffraction images were generated from a range of wavelengths. According to eq.(5.39), the pitch in wavelength was prescribed by the number of detector samples N,

$$\triangle\lambda = \bar{\lambda}\frac{2}{\mathsf{N}} \approx 0.52\,\%\,\bar{\lambda} \approx 1 \times 10^{-3}\,\mathrm{nm}\,. \tag{5.49}$$

In total, 76 wavelengths spanning a bandwidth of $\Lambda = 40\,\%$ allowed to simulate Gaussian spectra up until $\Lambda = 20\,\%$. Each spectrum was populated with 10^9 photons per exposure, in order to fill the diffraction patterns to large scattering angles.

Every possible probe-object combination was simulated for four different different bandwidths $\Lambda = 2$, 5, 10 and 20% resulting in 64 datasets. A selection of diffraction patterns from these datasets are displayed in panel row two and three of figure 5.1. Datasets with other types of spectrum were not considered as the long-tailed smooth Gaussian spectrum poses the hardest challenge.

Reconstruction results

Each dataset was reconstructed using three different models. The first assumed an entirely monochromatic experiment with all propagator, probe and object being non-dispersive. The second model assumed a dispersive probe and propagator but a non-dispersive object. The third model assumed dispersive propagator and object but a non-dispersive probe. The spectral resolution for the reconstructions was a fifth of the resolution in the simulation ($\mathrm{d}_\lambda/\bar{\lambda} = 2.5\%$), in order to save memory and computation time. Each dispersive reconstruction strategy enabled 1, 3, 7 and 15 spectral modes, respectively, for the 2, 5, 10 and 20% bandwidth datasets.

The retrieved probes and objects from this simulation exceed well beyond 600 wave field arrays, each of them complex. With this volume it is simply impossible to show and discuss every result. Nevertheless, a selection of object reconstructions are shown in Figures 5.6, 5.7 and 5.8 and the results are briefly commented in each caption. One striking similarity is that the monochromatic object reconstructions (top row) exhibit high-frequency artefacts for large bandwidths. This is a consequence of the monochromatic model not being able to account for the loss of speckle visibility at large scattering angles.

In order to quantify how well each model applies for each of the nine combinations of probe and object, four different error metrics were considered.

- The relative log-likelihood error ϵ_{ll} (see (2.214)) is an indicator for how well the model has allowed the engine to approximate the measured data. The error measures convergence but cannot detect if the retrieved spectral modes are at all accurate or meaningful. It only considers the incoherent sum at each scan point. If the model is to relaxed and provides too many degrees of freedom, it is not sufficient to monitor this error only.

- The normalised root square error was calculated for each spectral mode associated with the central wavelength $\bar{\lambda}$. For probe and object the error metric is written as

$$\epsilon_p = \frac{\left\| p_{\bar{\lambda}}(\mathbf{x}) - p(\mathbf{x}, \bar{\lambda}) \right\|_{\mathbf{x}}}{\left\| p(\mathbf{x}, \bar{\lambda}) \right\|_{\mathbf{x}}} \quad \text{and} \quad \epsilon_o = \frac{\left\| o_{\bar{\lambda}}(\mathbf{x}) - o(\mathbf{x}, \bar{\lambda}) \right\|_{\mathbf{x}}}{\left\| o(\mathbf{x}, \bar{\lambda}) \right\|_{\mathbf{x}}} \tag{5.50}$$

  where the reconstructed quantity $p_{\bar{\lambda}}$ is distinguished from the simulated reference by the discrete index. Apparently, ϵ_p and ϵ_o should be the ideal indicators whether or not the reconstruction approach was successful. However, far-field ptychography withholds several reconstruction ambiguities like a complex scaling factor, linear phase and an axis-offset (see eq.(2.215)). These ambiguities were removed in an automated a posteriori step here.

- The last error metric considered the recovered spectrum. For those models with spectral probe or object modes the spectrum was calculated from

$$\xi_\lambda = \int |p_\lambda(\mathbf{x})|^2 \,\mathrm{d}\mathbf{x} \quad \text{or} \quad \xi_\lambda = \int |o_\lambda(\mathbf{x})|^2 \,\mathrm{d}\mathbf{x} \tag{5.51}$$

and after normalization compared with the original Gaussian input spectrum

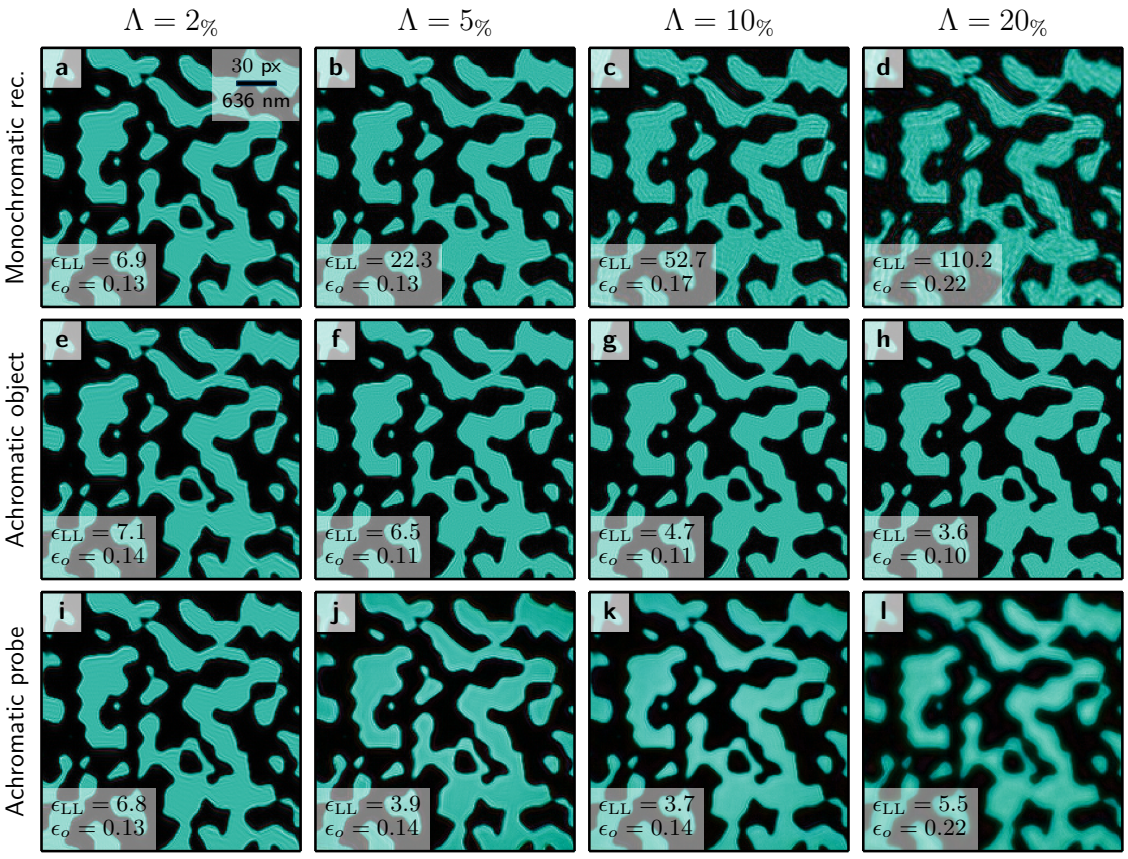
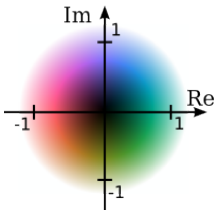


Figure 5.6. – Overview of the reconstructed object transmissions for a binary, opaque object in conjunction with a diverging illumination as shown in Fig. 5.4. Displayed is only the object mode at center wavelength, $o_{\bar{\lambda}}(\mathbf{x})$ for each reconstruction. Compared are the three reconstruction models (rows) for increasingly larger relative bandwidths Λ (columns). As the specimen is non-dispersive, the second row presents the best fit of the model (lowest ϵ_{LL}) and the lowest deviation from the simulated transmission (lowest ϵ_o) as one would expect. The color code of the complex plane is displayed to the right.



integrated over each wavelength bin,

$$\epsilon_\xi = \sum_\lambda \left| \xi_\lambda - \int_{-\Delta\lambda/2}^{\Delta\lambda/2} \xi(\lambda' - \lambda) \mathrm{d}\lambda' \right| \tag{5.52}$$

to yield a *spectrum error*. This error indicates how accurate the engine populated the spectral modes with energy.

Tables 5.2, 5.3 and 5.4 summarise the error metrics for the entire simulation. We

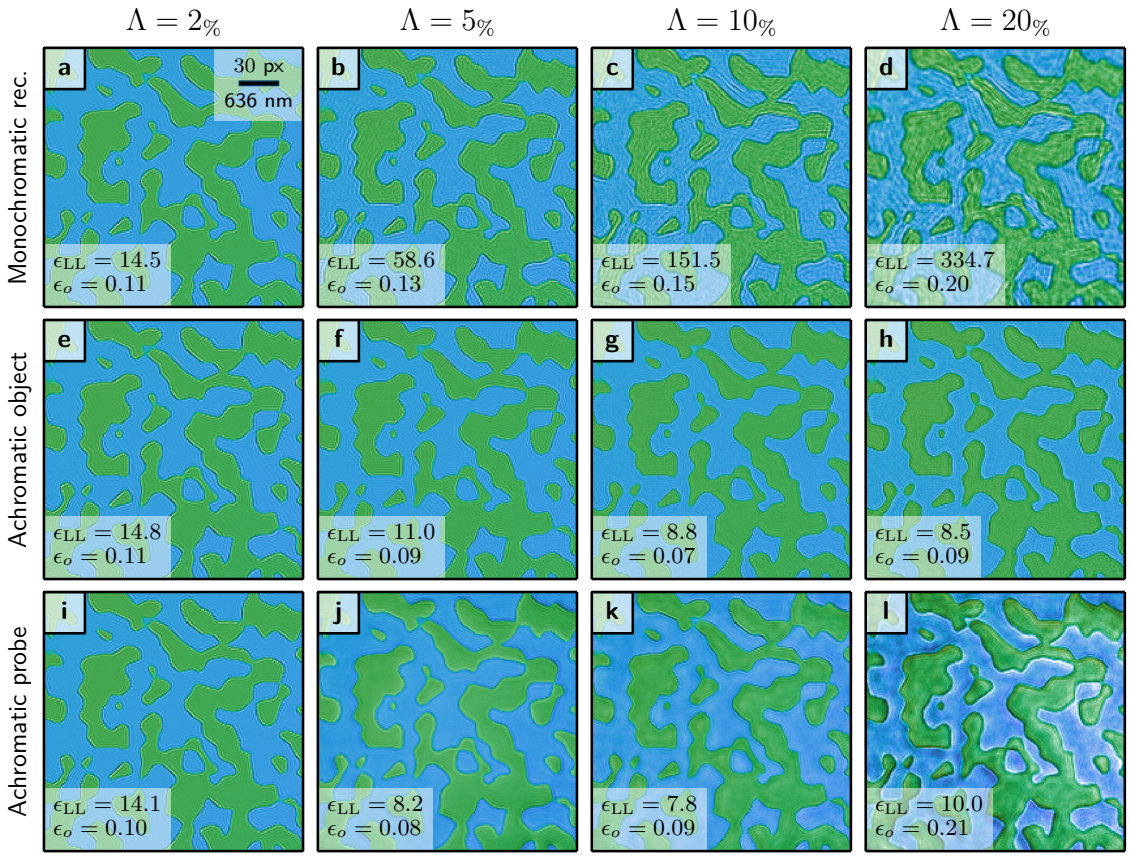
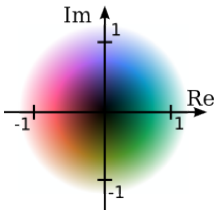


Figure 5.7. – Overview of the reconstructed object transmissions for a binary phase-shifting object in conjunction with a chromatic illumination from a propagated pinhole as shown in Fig. 5.3. Displayed is only the object mode at center wavelength, $o_{\bar{\lambda}}(\mathbf{x})$ for each reconstruction. Compared are the three reconstruction models (rows) for increasingly larger relative bandwidths Λ (columns). Here, none of the reconstruction models can be considered ideal, as both probe and specimen are dispersive. However, like in Fig. 5.6, the second row presents an overall good fit, indicating that the specimen is only slightly dispersive.



highlight the following observations:

- The monochromatic reconstruction model in Tab. 5.2 shows the expected behaviour of increasing errors for increasing bandwidth. This applies to all errors ϵ_{ll}, ϵ_p and ϵ_o. In general, the object error ϵ_o is higher than probe error ϵ_p. Surprisingly, the probe error stays low for the opaque or binary object.

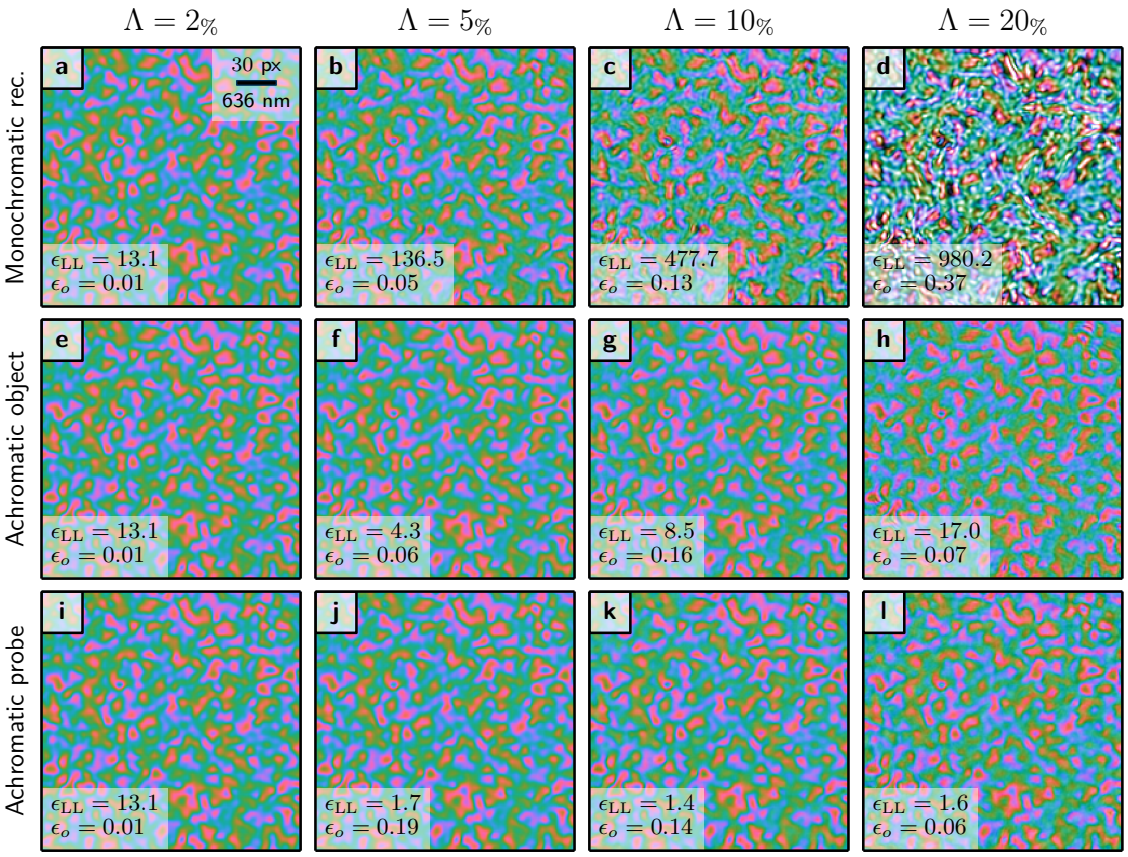
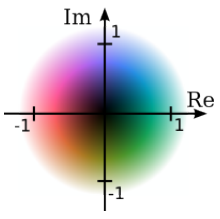


Figure 5.8. – Overview of the reconstructed object transmissions for a continuous phase-shifting object (see Fig. 5.5) in conjunction with an achromatic pinhole illumination in close proximity. Displayed is only the object mode at center wavelength, $o_{\bar{\lambda}}(\mathbf{x})$ for each reconstruction. Compared are the three reconstruction models (rows) for increasingly larger relative bandwidths Λ (columns). As a probe from a pinhole in close proximity is mainly non-dispersive, the last row presents the best fit of the model (lowest ϵ_{LL}) and the lowest deviation from the simulated transmission (lowest ϵ_o) as we would expect here. Despite a slightly higher model error, the reconstruction with a non-dispersive object seems perfectly fine even for large bandwidths. The monochromatic model fails entirely.



		contact probe				propagated probe				diverging probe			
	Λ	2%	5%	10%	20%	2%	5%	10%	20%	2%	5%	10%	20%
opaque	ϵ_{ll}	4	9	15	23	4	9	15	23	4	8	13	19
	ϵ_{o}	10	11	15	22	11	11	14	19	13	12	16	22
	ϵ_{p}	0	2	5	10	1	2	4	9	0	1	2	4
binary	ϵ_{ll}	3	8	14	22	4	9	15	22	4	8	12	18
	ϵ_{o}	5	7	13	23	10	12	15	19	15	14	16	21
	ϵ_{p}	1	2	6	11	1	2	4	9	0	1	2	4
phase	ϵ_{ll}	4	14	27	41	5	14	27	41	4	14	26	40
	ϵ_{o}	1	4	12	36	2	4	12	35	6	11	18	38
	ϵ_{p}	0	3	6	11	1	3	7	13	0	2	6	11

Table 5.2. – Error metrics ϵ (in %) for a conventional **monochromatic** model.

		contact probe				propagated probe				diverging probe			
	Λ	2%	5%	10%	20%	2%	5%	10%	20%	2%	5%	10%	20%
opaque	ϵ_{ll}	4	2	2	2	4	3	3	4	4	3	3	4
object	ϵ_{ξ}	0	7	2	1	0	6	7	6	0	5	35	53
	ϵ_{o}	9	10	11	13	10	11	12	18	12	14	14	22
	ϵ_{p}	0	1	2	3	1	2	2	3	0	1	1	2
binary	ϵ_{ll}	3	2	1	1	4	3	3	3	4	3	3	3
object	ϵ_{ξ}	0	6	3	1	0	4	6	4	0	9	37	57
	ϵ_{o}	4	4	7	8	10	8	9	20	15	13	12	23
	ϵ_{p}	1	1	3	3	1	1	1	3	0	0	1	2
phase	ϵ_{ll}	4	1	1	1	5	4	4	5	4	3	4	5
object	ϵ_{ξ}	0	4	3	1	0	6	10	7	0	0	29	23
	ϵ_{o}	1	18	13	6	2	10	7	16	5	16	16	31
	ϵ_{p}	0	8	6	3	1	2	1	3	0	0	2	8

Table 5.3. – Error metrics ϵ (in %) for a polychromatic model with a **non-dispersive probe**.

		contact probe				propagated probe				diverging probe			
	Λ	2%	5%	10%	20%	2%	5%	10%	20%	2%	5%	10%	20%
opaque	ϵ_{ll}	4	4	3	3	4	4	4	3	4	4	3	3
object	ϵ_{ξ}	0	24	7	1	0	22	5	1	0	20	4	1
	ϵ_{o}	10	10	11	12	11	9	8	9	13	11	11	10
	ϵ_{p}	0	3	4	5	1	4	4	5	0	4	4	5
binary	ϵ_{ll}	3	4	3	3	4	4	3	3	4	4	3	3
object	ϵ_{ξ}	0	21	5	2	0	18	4	2	0	16	2	1
	ϵ_{o}	5	6	6	8	10	8	7	9	15	8	10	10
	ϵ_{p}	1	4	4	5	1	4	3	4	0	3	3	5
phase	ϵ_{ll}	4	2	3	5	5	3	4	5	4	2	3	5
object	ϵ_{ξ}	0	4	1	3	0	0	2	3	0	1	1	3
	ϵ_{o}	1	5	15	7	2	4	6	9	7	2	5	13
	ϵ_{p}	0	2	7	2	1	2	3	3	0	2	2	2

Table 5.4. – Error metrics ϵ (in %) for a polychromatic model with a **non-dispersive object**.

- The object error for the binary, opaque object stays always at or around 10 % which can be attributed to ringing artefacts (aliasing) at the domain borders (see Fig. 5.6).

- The broadband reconstruction model with a non-dispersive object and a dispersive probe (Tab. 5.4) seems to perform well for all combinations. The log-likelihood error is low everywhere, suggesting a good fit of the model to the diffraction data. Application optimum for this model is the top row and, despite previously mentioned ringing artefacts, probe error ϵ_p and spectrum error ϵ_ξ are lowest in this row. In particular, the spectrum error can be as low as 1 % for $\Lambda = 20\,\%$ which indicates, that the input spectrum was retrieved exceptionally well.

- The broadband reconstruction model of a dispersive object and a non-dispersive probe (Tab. 5.3) performs not so well when compared to the model with a non-dispersive object and a dispersive probe. Application optimum for this model is the left column, and in fact, log-likelihood error ϵ_{ll} and spectrum error ϵ_ξ are lowest in this column. For probes, that exhibit a strong dispersive behaviour, this model fails in object and spectrum recovery and performs almost as bad as the monochromatic model regarding the object error ϵ_o. As the log-likelihood error stays low in every case, it stands to reason, that the additional degrees of freedom in the spectral object modes have been used for compensation while deteriorating the object reconstructions. The images of the object reconstruction (Fig. 5.6l and Fig. 5.7l) reveal that the object develops low frequency artefacts in these cases.

5.3.2. Discussion

With the observation we have not yet exhausted all the available information in tables 5.2 to 5.4. However, the simulation shows that a model with a non-dispersive probe and a dispersive object or vice versa succeeds to reconstruct broadband data in selected cases. That means that polychromatic data may be reconstructed without any additional a priori knowledge apart from the basic geometry. In particular we note, that a non-dispersive object with a dispersive probe appears to yield better results. Possible reasons are:

- Constructing a good initial guess is easier for the probe due to knowledge of beamline optics.

- The phase space of the probe is stronger constraint as it contains less degrees of freedom than the phase space of the object.

- Most of the error in ϵ_{ll} arises from the false scaling of the frequency components when the dispersion relation of the flight path is dominant. Compared with spectral density of probe and object, the spectrum only requires few extra parameters. As both spectral modes of probe and object can incorporate the spectrum as part of the cross-spectral density, it is beneficial to entrust the recovery to the model with the fewest extra degrees of freedom which is the model of a polychromatic probe.

However, for some cases the actual spectral density distribution of both probe and object may be of particular interest. Other than assuming non-dispersive behaviour for either probe or object, a quadratic approximation of their respective dispersion relation or other measures may become necessary. It is safe to assume, that ptychography will deliver appropriate solutions if additional a-priori knowledge is phrased into constraints and included in the reconstruction procedure. This assumption is backed by the multiplexing approach (Batey et al., 2014). The idea can be lifted to an extreme level, where entire different experiments are added in the diffraction data, but would also require explicit a priori knowledge on either all probes or objects.

In any case, the expected results should justify the added computational expense and programming effort in formulating a priori knowledge as additional constraints. For example, the multiplexing scheme will not benefit from the higher photon count in the diffraction image as each spectral intensity contribution is attributed to an entirely different object. That contradicts our primary motivation to utilize the higher photon flux in broadband spectra.

Concluding, the simulations demonstrated that recovering the probe's spectral density is particularly straight forward and stable. Since no a priori information is needed about the spectral composition of the source, this method may even be suitable to recover the spectrum of the source *and* the spectral wave fronts in a single measurement. Therefore, the next section is dedicated to an experiment as a proof of concept for this method.

5.4. White light ptychography

A set of experiments were carried out in order to determine whether broadband spectra and the associated cross spectral density of an illumination may be recovered from ptychographic diffraction experiments. For this purpose, spatially coherent but broadband sources were derived from LEDs as primary source.

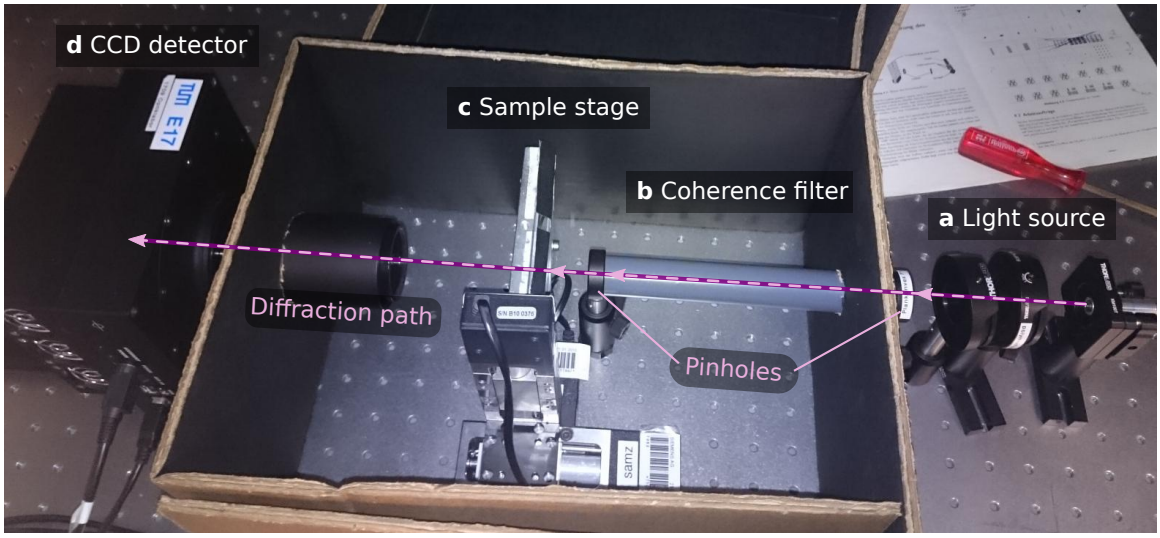


Figure 5.9. – Photograph of the experimental setup used for visible-light ptychography with ultra-broadband spectra. **P**osition for primary light source of any kind. **b** Consisting of two pinhole apertures attached to an empty tube, the coherence filter defines the optical axis. **c** Stack of sample stages to move the specimen along and transverse to the optical axis. **d** CCD detector (ProLiant PL1001 camera) after free propagation path. The tubes of camera and coherence filter penetrate the black cardboard enclosure which was used here to shield the experiment from ambient light sources.

5.4.1. Light source

The main difficulty for designing a broadband, visible-light experiment for ptychography was finding an appropriate light source of high spatial coherence and low temporal coherence. Most often, broadband sources also show poor spatial coherence.[1]

Exceptions to this rule are short pulsed laser sources as used in pump-probe experiments or non-linear optics (Herink et al., 2012). These lasers possess broad spectra as a result of their short pulses and are inherently spatial coherent being a laser after

[1] Behold the candle!

all. However as lab equipment, they also stand for a substantial financial investment and should serve other purposes beyond a proof-of-principle experiment.

As a consequence, in first experiments laser beams from economical laser diodes of different wavelengths (532 nm, green ; 632 nm, 650 nm and 670 nm, all red) were superimposed in different combinations to achieve a 2-line or narrow bandwidth source of high spatial coherence. That idea proved to be unsatisfactory as the narrowband source made from the red lasers had only 6 % maximum relative bandwidth and broadband deterioration on the reconstruction process remain negligible, when the relative bandwidth stays around or below 5 %[2] as indicated by the simulations (see sec. 5.3.1). A 2-line spectra made from green and red lasers, on the other hand, exhibits a high bandwidth, but the strictly monochromatic emission lines and the large separation may eventually separate the speckle in the diffraction pattern wide enough such that speckle visibility is preserved for each color. Such a scheme increases the information density in the diffraction pattern and the data can be used in a multiplexing approach, as was found out by Batey et al. (2014).

Much like a synchrotron beam line today, the final setup relied on the van Cittert-Zernike theorem. With two lateral filters and a free-space propagation path in-between, any light source can be made spatially coherent, even with a broad and continuous spectrum.

Fig. 5.9 contains a photograph of the opened setup. A (flight) tube of length $l = 19$cm with pinhole apertures on both ends served as a *spatial coherence filter (SCF)* here. It emits entirely spatially coherent wave fields regardless of the primary source. The pinhole facing the illumination had a diameter of $D = 100\mu$m. According to eq.(2.151) and assuming incoherent illumination, the complex degree of coherence is given by the Fourier transform of the source-facing pinhole.

$$\gamma(\Delta\mathbf{x}) = \mathcal{F}_{\mathbf{x}'}\left\{\mathrm{circ}\left(\frac{|\mathbf{x}'|}{D}\right)\right\}\left(\frac{\Delta\mathbf{x}}{\lambda l}\right) = \frac{2\lambda l}{\pi|\Delta\mathbf{x}|D}\mathcal{J}_1\left(\frac{\pi|\Delta\mathbf{x}|D}{\lambda l}\right). \tag{5.53}$$

where $\mathcal{J}_1$ is the first order Bessel function. We can use its first root as an upper bound for the transverse coherence length $\ell_{\mathbf{x}}$ at the plane of the second pinhole which had a diameter of $150\mu m$.

$$3.83 \approx \frac{\pi\ell_{\mathbf{x}}D}{\lambda l} \quad \rightarrow \quad 1.0\mathrm{mm}(\lambda = 420nm) \lesssim \ell_{\mathbf{x}} \lesssim 1.5\mathrm{mm}(\lambda = 650nm) \tag{5.54}$$

The transverse coherence length exceeds by far the diameter of the second pinhole. Hence, the field diverging from the second pinhole is entirely spatially coherent.

[2] This is true when far-field diffraction is considered and the dynamic range of the detector image or accumulated images stays below 10^6. Higher dynamic ranges and thus higher resolutions than used here may benefit from a spectral reconstruction algorithm

The disadvantage of a coherence filter is, of course, a poor photon yield when compared to those available at the primary source. Looking for broadband, "white" primary light sources, so-called "super-bright" light emitting diodes (LEDs) were selected. Today's white LEDs commonly consist of a high-efficiency blue LED core that is embedded in a phosphor. After excitation, the phosphor emits over a broad frequency band with a maximum around the yellow light. In conjunction with the blue emission line from the core, human eyes perceive this light combination as white. An alternate approach to produce white light is to superimpose light from a red, blue and green LED in the right mixture. The following three sources were eventually used in the experiments:

- A common 5mm wide *warm-white LED* with diffused head,
- a high-power (>300mA) four-chip *RGBW-LED* and
- a *green laser diode* as reference of 532nm wavelength.

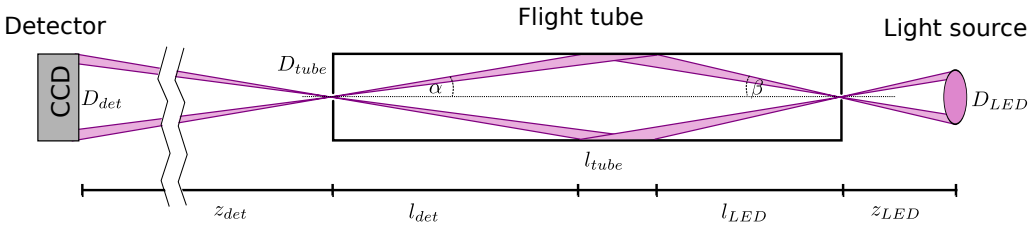
The color channels of the RGBW-LED were independently dimmable from command-line using a self-built control circuit with a USB-to-serial connection. Here, only the red (R), green (G) and blue (B) channel were powered which is why we will also refer to that source as RGB-LED.

5.4.2. Geometric considerations

While the SCF appears to be a simple optical element, sending as much light through it presents some experimental intricacies. First of all, the filter defines the optical axis as the line intersecting both pinhole apertures. Any light source must be placed on or very close to the optical axis in order to illuminate the target behind the second pinhole. Thus, at first glance, placing the light source as close as possible to the first pinhole or using a focussing lens would ensure that the primary illumination is on the optical axis. However, that turned out to be bad practice as it would cause incoherent reflections to propagate into the peripheral areas of the detector, where the large-angle scattering is recorded. This effect can be explained with Fig. 5.10a. A pinhole acts a simple imaging device, where light entering at one angle will exit the pinhole under the same angle (though mirrored at the optical axis). In this manner, the CCD in combination with the second pinhole, acts as a *camera obscura* for all regions in the flight tube with an angle smaller than

$$\alpha \approx \frac{D_{\mathrm{det}}}{2z_{\mathrm{det}}}\,. \tag{5.55}$$

a Internal tube reflections from close light source



b Source separation from first pinhole

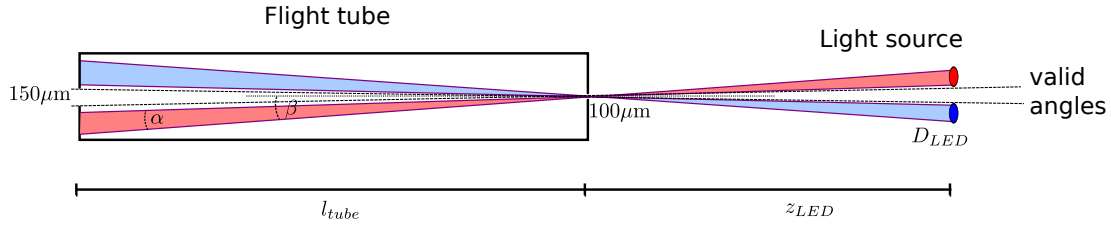


Figure 5.10. – Geometric consideration regarding the spatial coherence filter. **a** Incoherent reflections from a light source that is close to the input pinhole. **b** Separation of secondary sources at the specimen-facing pinhole as a consequence of the camera obscura principle.

which translates into a depth $l_{\mathrm{det}} = \alpha^{-1} D_{\mathrm{tube}}/2$ on the flight tube wall. All light scattered into the first part $l_{\mathrm{tube}} - l_{\mathrm{det}}$ of the tube is imaged *incoherently* onto the detector. Thus the projection depth l_{LED} of the light source must be greater than $l_{\mathrm{tube}} - l_{\mathrm{det}}$ leading to the inequality

$$z_{\mathrm{LED}} > D_{\mathrm{LED}} \left(\frac{l_{\mathrm{tube}}}{D_{\mathrm{tube}}} - \frac{z_{\mathrm{det}}}{D_{\mathrm{det}}} \right) \approx D_{\mathrm{LED}} \left(\frac{190\mathrm{mm}}{18\mathrm{mm}} - \frac{24cm}{1024 \cdot 24\mu\mathrm{m}} \right) \tag{5.56}$$

Condition (5.56) does not seem very restrictive for z_{LED} as both fractions assume a value of roughly 10. However, the tube was not well aligned to the optical axis which means that the effective inner diameter D_{tube} was smaller. Internal reflections were visible in the periphery of the detector when placing the source within $\approx$ 1cm distance of the first pinhole even though the source usually never extended wider than 5 mm. Usually, the source was placed at around 5 cm distance to the SCF in order to avoid avoid any incoherent internal reflections.

The second issue accompanying the chosen SCF was the apparent separation of sources at the plane of the second pinhole. A look at Fig. 5.10b reveals that the camera obscura principle holds for the primary source, imaging laterally displaced sources according to their respective angles to the optical axes. This property posed a problem for the four-chip array of the RGBW-LED. Effectively, the LED array would need to be positioned far away enough for all chips to be within the cone of

valid angles. In practice, it was impossible to position the LED far away for all chips to contribute *and* keep the LED array on the optical axes at the same time. Eventually a diffuser was used to superimpose all light sources in a secondary source plane, yielding a medium efficient primary source despite its high-flux capabilities.

Regardless of the source, the comparatively small photon flux after the SCF required careful shielding from parasitic illumination which was achieved with a blackened custom cardboard enclosure that can be seen (without lit) in Fig. 5.9. Unfortunately, aligning and changing the primary source became a hit-and-miss endeavour due to the chosen design. The camera, in turn, could be utilised at full potential. The combination of mechanical shutter with large pixels of high sensitivity and dynamic range is originally designed for low-light environments like in fluorescence or astronomical imaging.

5.4.3. Ptychography

The feasibility of broadband ptychography was now verified experimentally. This section presents results from six datasets with six different sources, a green laser (S1), a warm white LED (S2) and an RGBW LED with all color channels active (S3) and each color channel active (S4-S6).

Diffraction data
In order to reach a high dynamic range, all diffraction images were stitched together from a series of exposures. Starting from the longest exposure, each pixel counting more than 50000 events (80 % fill) was replaced subsequently by a scaled value of the next shorter exposure. The exposure series (in seconds) was $[1.0, 4.0, 16.0, 64.0]$ for the white LED source (S2), $[2.0, 8.0, 32.0, 128.0]$ for any color combination combination of the RGBW LED (S3-S6) and $[0.2, 0.8, 3.0]$ for the green laser source (S1). The stitched diffraction images were cropped around the optical axis and binned by 3x3 pixels yielding diffraction images of 256×256 pixels and a dynamic range of roughly 10^5. Selected diffraction images are depicted in Fig. 5.11d-f and in Fig. 5.11g-i for the green laser and white led, respectively.

Setup geometry
The distances from detector to sample and from sample to pinhole were 230 mm and 21 mm, respectively. The design mean wavelength for the propagator was 532 nm which is the spectral line of the green laser diode. A binned detector pixel had a side length of 72 µm resulting in a pixel size of 6.63 µm in the sample plane which corresponds to a resolution of $(13.2\,\mu\text{m})^{-1}$ or $75\,\text{mm}^{-1}$

Scans pattern

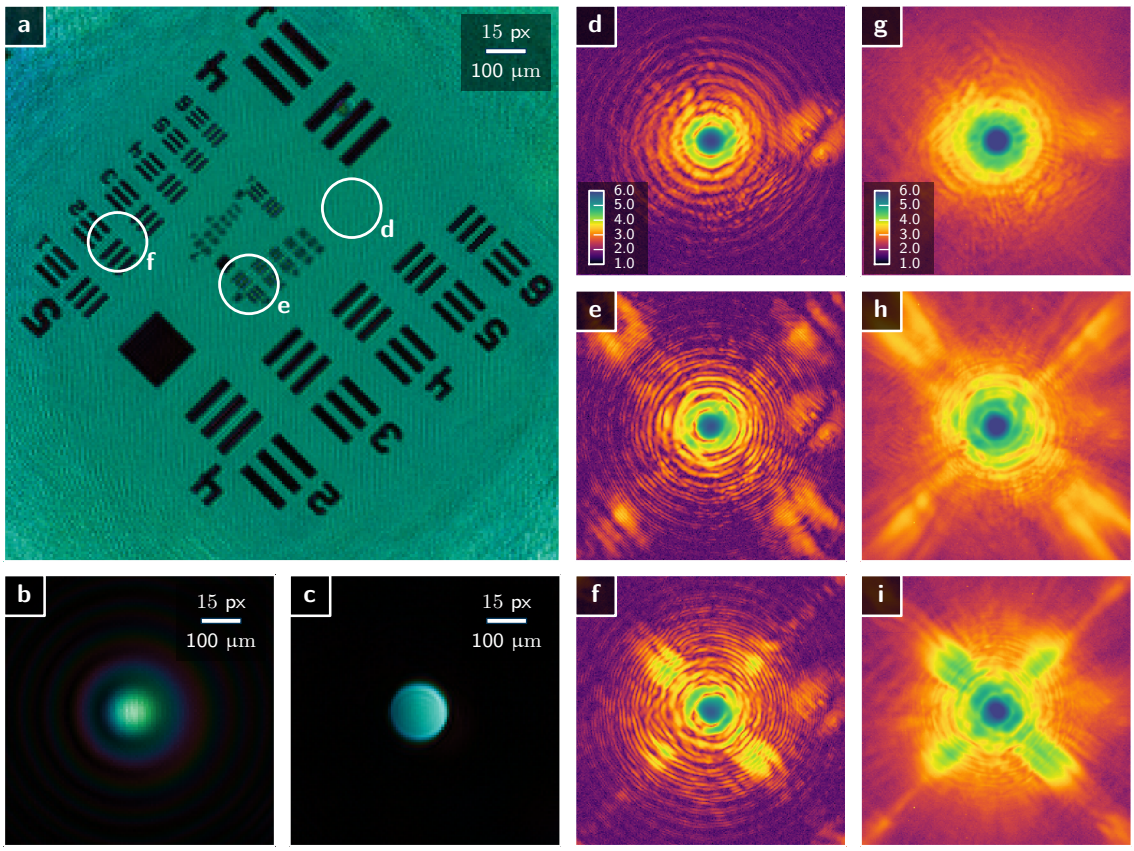


Figure 5.11. – Overview of the visible light experiments with ultrabroad spectra. **a** Wave field image of the reconstructed object for the laser data set. **b** Wave field image of the reconstructed probe. **c** Wave field image of the probe back propagated by 21 mm to the pinhole plane. The circular patches in **a** have the same diameter as the second pinhole of the SCF. They are centred around scan positions with distinctly different diffraction images. The rightmost circle rests in an area with almost no scattering from the sample. The corresponding diffraction image (**d**) is thus mainly a far field diffraction image of the optics and features the common airy disc pattern. **g** Diffraction image at the same position, but with a white LED as primary source. the airy disc pattern is washed out due to the ultrabroad spectrum. **d**, **e** Diffraction images from positions with large-angle and small-angle diffraction of the specimen. The corresponding broadband diffraction images are in panel **h** and **i**

All scans used a round scan pattern of equidistant shells. The shell distance was 43 µm for S1 and 50 µm for the LED sources. The final number of diffraction patterns for each scan are listed in Tab 5.5. The varying scan patterns were not intentional but merely reflect adaptation to a restricted budget of laboratory time.

Probe

As mentioned previously, the probe was shaped by the sample-facing pinhole aperture

of the SCF. Due to the gap between pinhole and sample, the probe presents dispersion as a result of free-space propagation. A reconstruction image of the probe's wave field from the laser data set can be seen in Fig. 5.11b. The slight deviation from a circular symmetry arises from an inhomogeneous illumination of the second pinhole. In fact, the back-propagated wave field readily shows a gradient in the pinhole aperture, indicating a slight misalignment of the laser path to the optical axis.

Object

The sample was a USAF-1951 optical resolution target (see appendix, A.3). At optical wavelengths, it qualifies as binary opaque object. A reconstructed image of the sample transmission is depicted in Fig. 5.11a

Reconstructions

Dataset	S1	S2	S3	S4	S5	S6
Light source	laser	white	rgb	red	green	blue
Number of images	375	330	275	275	275	275
Photons per image [10^6]	158	534	597	208	226	210
Relative LL error ϵ_{ll}	10.08 %	5.06 %	6.34 %	9.16 %	7.08 %	8.29 %
Pixel LL error ϵ_{LL}	24.49	20.81	36.60	26.66	17.25	22.06

Table 5.5. – Overview of the reconstruction errors in relation to the number of diffraction images and photon flux.

For visible-light ptychography, the free-space propagation path for diffraction is much shorter than for X-ray ptychography. Additionally, the compact, obstructed design of the setup prevented a precise measurement of the distance with a ruler. Due to the distance-wavelength equivalence of the Fresnel propagator, a measurement error for the propagation distance directly translates into an offset in wavelength for the spectral density. But for the same reason, a broadband reconstruction scheme can be interpreted as a search algorithm for the propagation distance if a monochromatic light source is used (see eq.(5.42)). The precision depends on the sensitivity of the implemented spectral propagation (see (5.39)),

$$\triangle \mathsf{z}_{\min} = \frac{2\bar{\mathsf{z}}}{\mathsf{N}} \tag{5.57}$$

which is approximately 2 mm here. Therefore, the data set S1 served as reference dataset for distance calibration and was reconstructed multiple times with a broadband strategy of successively increasing sensitivity. The "distance" of the mode with the highest power was used as mean propagation distance for the next reconstruction with finer sensitivity $\triangle \mathsf{N}/\mathsf{N}$. The last reconstruction of this series revealed that the

minimal data sensitivity was roughly $5/\mathsf{N} \approx 2\,\%$ here. Eventually, the laser data set was reconstructed with a monochromatic model using 400 iterations of DM and 100 iterations of ML refinement.

All other datasets were reconstructed using a broadband model (see eq. (5.41)) with a sensitivity of $10/\mathsf{N} \approx 4\,\%$. The comparatively large separation, $\triangle\lambda \approx 21\,\mathrm{nm}$, avoids cross-talk between neighbouring wavelengths and allows to cover a very broad bandwidth of 250 nm with only 13 spectral modes. The reconstruction engine consisted of 700 iterations of DM followed by 150 iterations of ML refinement. The object was assumed non-dispersive while the probe was assumed dispersive. All spectral probe modes used a propagated pinhole of equal power as starting guess which means a box-type initial spectrum. Tab.5.5 summarizes the overall good convergence of the applied engines.

5.4.4. Results

Recovery of broadband spectra

From the model (5.41) of a broadband illumination,

$$I_\mathrm{j}(\mathbf{s}) = \sum_\lambda \mathcal{P}_\lambda \left\{ p_\lambda(\mathbf{x}) o(\mathbf{x} - \mathbf{y}_\mathrm{j}) \right\}$$

we identify

$$\xi_\lambda = \int |p_\lambda(\mathbf{x})|^2 \,\mathrm{d}\mathbf{x} \tag{5.58}$$

as the part of the spectrum occupied by the chromatic mode. For the white light dataset (S2), Fig. 5.12d shows the reconstructed discrete spectrum ξ_λ from the 13 chromatic probe modes in the form of a bar plot. The bars are coloured corresponding to the colour of their respective wavelength. For validation, the LED source was removed from the setup to record the spectrum with a common visible-light spectrometer from OceanOptics. Scaled with $\triangle\lambda$, said spectrum is plotted as black line into the same panel d. Both the reference spectrum and the power of the chromatic modes show the characteristic shape of a white LED consisting of a blue emission peak and a broad emission arc from green to red from the excited phosphor. For better comparison with the discrete spectrum ξ_λ, the locally integrated spectrum,

$$\tilde{\xi}_\lambda \equiv \int_{-\triangle\lambda/2}^{\triangle\lambda/2} \xi(\lambda' - \lambda)\mathrm{d}\lambda' \tag{5.59}$$

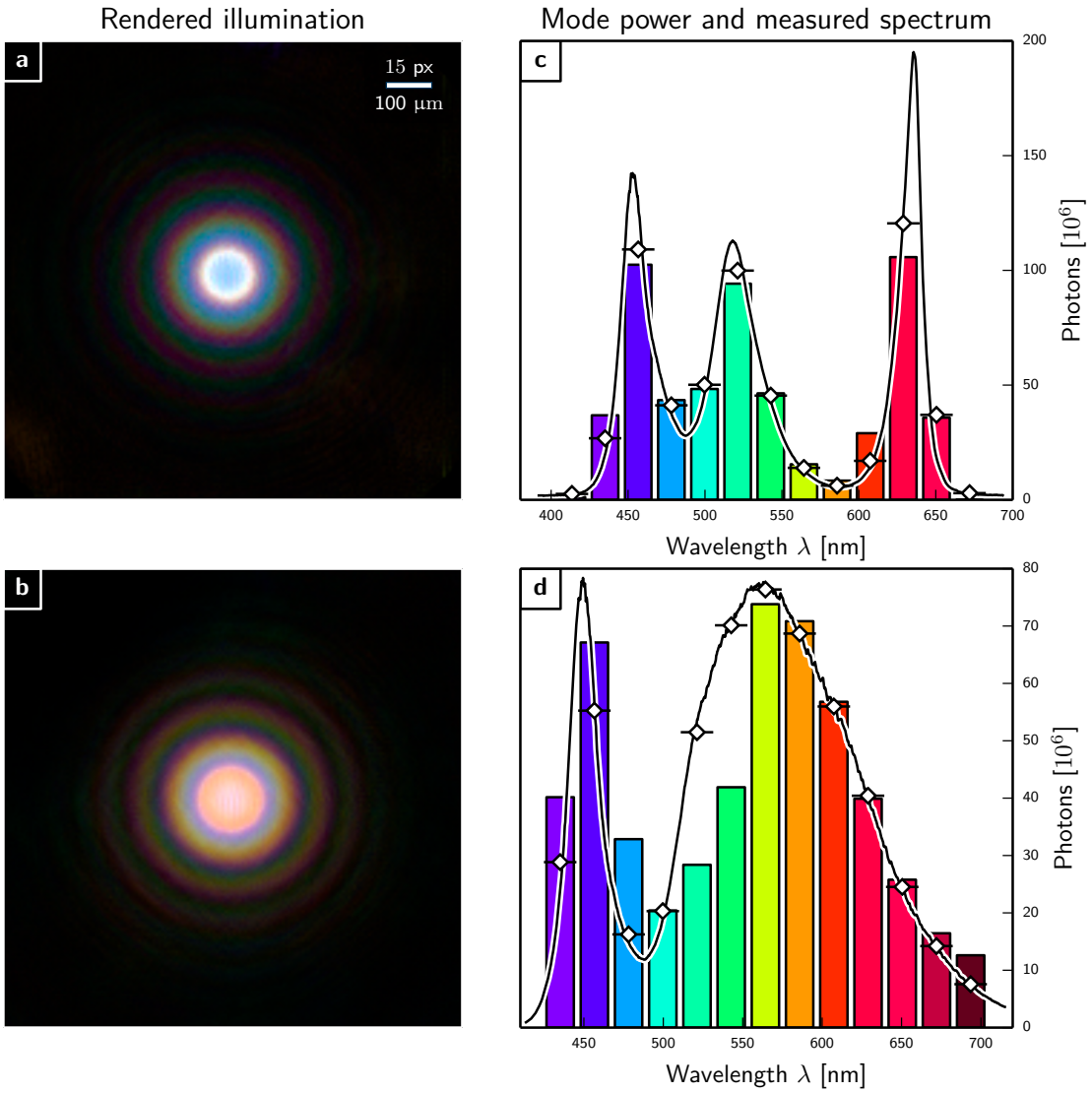


Figure 5.12. – Ptychography with spectral modes and polychromatic illumination of ultra-broad bandwidth. **a,b** illumination from RGB-LED and white LED, respectively, rendered as if observed by a human eye in the specimen plane. The dispersive property of free-space propagation is nicely observable through a rainbow-effect in the circular diffraction fringes of the airy-pattern. For the rendering, the spectral density $|p_\lambda(\mathbf{x})|^2$ was multiplied with the vector representation of the wavelength λ in XYZ color space. Then the colors were added pixel-wise and the resulting XYZ color matrix was transformed into RGB color space. **c,d** Retrieved spectra for RGB-LED and white LED, respectively as bar plot. The bars are coloured corresponding to the perceived color of their respective wavelength. A reference spectrum acquired with a common visible-light spectrometer is scaled with $\triangle\lambda$ and plotted as black line into the same panel. For best comparison, the locally averaged spectrum is also plotted using a marker consisting of a white rhombus above a black stroke of length $\triangle\lambda$

is also plotted into panel d. The respective a marker consists of a white rhombus above a horizontal black stroke of length $\triangle\lambda \approx 21\,\mathrm{nm}$. We notice that the recovered spectrum matches exceptionally well for wavelength greater than 550 nm. However, for smaller wavelengths we observe a certain discrepancy. The green part of the spectrum is attenuated while the blue part is amplified with respect to the reference spectrum. As shown in Fig. 5.13 the sudden drop for the green wavelenghts can

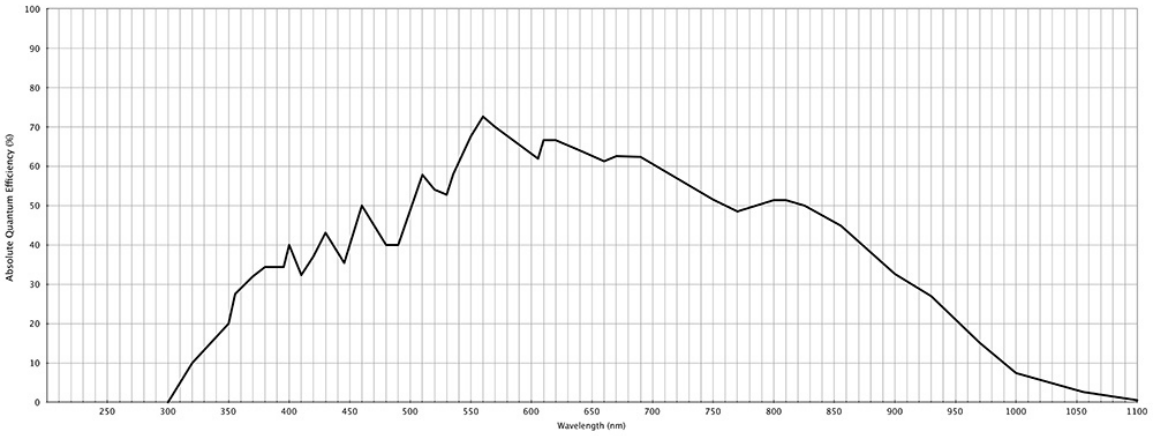


Figure 5.13. – Quantum efficiency of the monochrome ProLiant PL1001 camera.

be explained with the quantum efficiency curve of the CCD camera. The quantum efficiency declines by almost a factor of 2 from 550 nm to 500 nm. After that, the quantum efficiency remains aproximately on the same level. For wavelengths smaller than 500 nm we would therefore still expect the reconstructed spectrum ξ_λ to assume lower values than $\tilde{\xi}_\lambda$. However, the reverse behaviour is observable. That means that another stronger effect covers the low quantum efficiency. One possible candidate for that effect would be a form of chromatic dispersion from the SCF:
Smaller wavelengths diffract less after the first pinhole and stay closer to the optical axis in the free propagation path of the tube. Therefore, the sample-facing pinhole filters less flux from the shorter wavelengths than it does for longer wavelengths. As the flux scales with the cross section of the pinhole, it should scale squared with the inverse wavelength which means that longer wavelengths are more attenuated than shorter wavelengths.
Another possible explanation would be an inhomogeneous coating of the LED chip by the phosphor. As a result, the ratio between primary emission line and secondary emission arc from the phosphor may not be uniform with solid angle. Therefore, this ratio may differ from the reference spectrum to the experiment simply because the photons were extraced at different angles.

The case is different for the RGB dataset (S3). First of all, because the color channels were individually dimmable, the LED currents were calibrated in an approximate

manner before the experiment. The calibration happened in a fairly straight forward manner, taking diffraction patterns at equal dimming level and comparing the number of photons in the detector for each channel. The resulting dimming ratio for R:G:B was approximately 0.5:1:0.33 such that each colour would contribute equally to the total photon count. Matching the dimming level to the expected signal effectively compensates for the chromatic efficiency of detector and SCF.
The validation with the spectrometer also required a different approach. The input to the spectrometer was a tip of an optical fibre, i.e. a very localised test point. As the light sources on the chip are spatially separated, the power ratio of the color channels at the fibre tip varied in the spectrum when moving the tip, even with an additional diffuser in between and even when applying the exact same currents through the chips as was done in the experiment.
Instead, spectra were recorded for each color channel and then scaled according to the number of photons in tab. 5.5 before they were superimposed to form the RGB spectrum in Fig. 5.12c. This reference spectrum exhibits a remarkable conformity with the retrieved spectrum.

In summary, retrieving a spectrum over such an ultrabroad spectral range works exceptionally well.

Fig. 5.12a & b depict the broadband illumination as it would be observed by a human eye. The images were rendered in the following way. The spectral density $|p_\lambda(\mathbf{x})|^2$ was multiplied with the vector representation of the wavelength λ in XYZ color space. Then the colors were added pixel-wise and the resulting XYZ color matrix transformed into RGB color space.

Here, this RGB rendering of the spectral density was given precedence over showing the individual spectral modes, as these are "just" scaled Bessel functions from pinhole diffraction.

Reconstructed transmission of the USAF resolution target

Fig. 5.14 contains absorption images of the reconstructed object transmission for all six datasets. All reconstructions are of almost equal quality which is largely due to the similar (high) photon count in each scan. The purple arrow marks the 2nd element in the sixth group which corresponds to a line pair resolution of 71.84 mm m. With a resolution of $75\,\mathrm{mm}^{-1}$ that line pair is expected to be the smallest line pair still resolvable within the diffraction limit. That is apparently true for the blue (d), green (e) and white (c) LED illumination. The red LED (f) struggles to resolve line pair group 6 with good detail, because the algorithms resolves with finer detail than the diffraction limit for red light in this geometry.

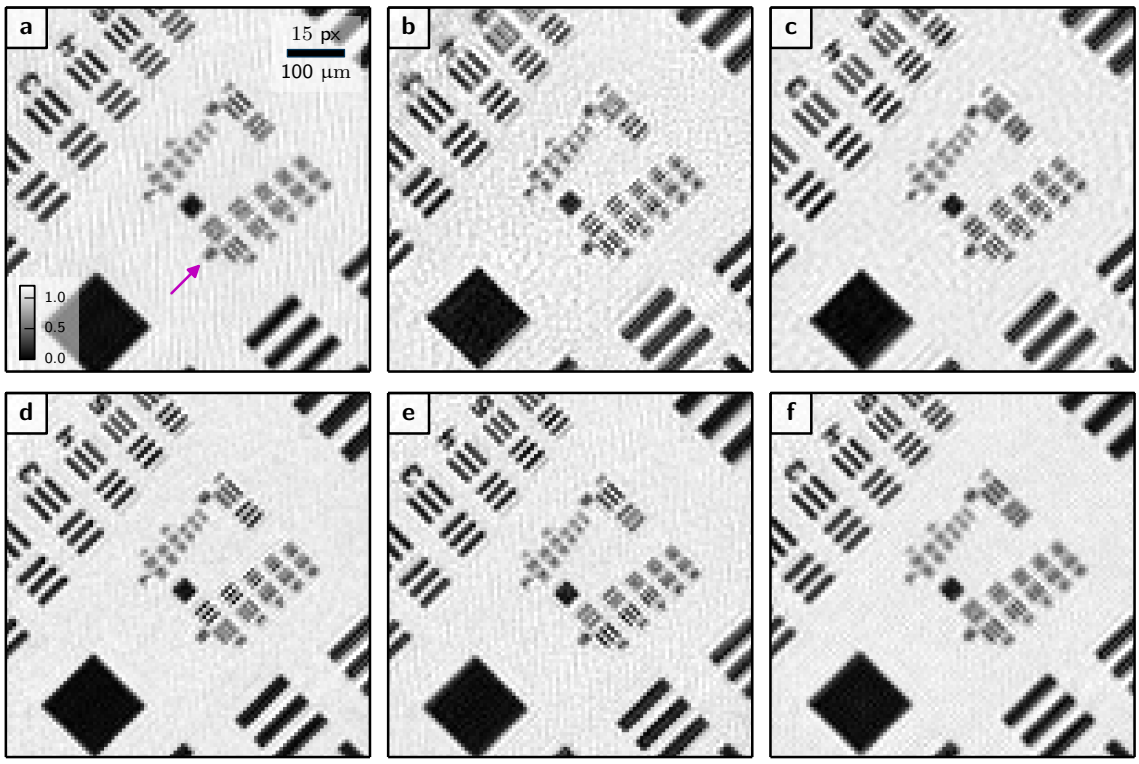


Figure 5.14. – Absorption images of the reconstructed object for six different light sources: **a** Green laser diode. **b** RGBW LED with all coloured diodes (red, green and blue) powered on. **c** White LED. **d-f** RGBW LED with power to the blue (**d**), green (**e**) and red (**f**) chip.

Discussion

The spectrum recovery from a probe mode reconstruction appears to work exceptionally well. In addition to the spectrum, the full cross-spectral density of the probe is characterised. This remarkable result would potentially enable the experimenter to characterise the wave front at the sample stage for all wavelengths simultaneously.

Future experiments should cover other types of specimens with a more unidirectional diffraction contrast. For algorithmic recovery it would be interesting to alter the number of spectral modes and spectral sampling $\triangle\lambda$ of the propagator. For best comparability and verification a spectrometer should be included in a revised setup, maybe on an alternative stage. However, even without an in-place spectrometer, there is little doubt that the reconstructed spectra are inaccurate. In addition to the comprehensive simulation study on symmetric spectrum in sec. 5.3 and in order to ensure feasibility, the experiment was simulated and analysed prior to the actual experiments. These simulations employed spectra with the characteristic shape of a white LED. With the same reconstruction model as used for the experimental data, it

was possible to retrieve the spectrum from the reconstruction with high accuracy.

5.5. Conclusion

In section 5.1 we realised that polychromatic blurring of the large-angle far-field speckle is mainly caused by the chromatic dispersion of free space propagation. Even then, it requires large bandwidths $\Lambda > 2\,\%$ for the error to rise to significant values. In the simulation, the diffraction patterns (see Fig. 5.1) contain a high number of speckles when compared to typical diffraction patterns. As a consequence, these simulations were also more susceptible to broadband deterioration from a polychromatic spectrum because the speckle count corresponds to the space-bandwidth product QD. Therefore, the lower the speckle count and dynamic range, the higher should be the threshold for a broadband spectrum to impair data quality. As the threshold may even be much higher than the 2 %, it explains why the broadband experiments at ESRF in Sec. 4.1 did not require chromatic modes at all for reconstruction. In general for a synchrotron beamline operating with hard X-rays, we conclude that a monochromator is *not* required for successful far-field ptychographic imaging. Instead, the full undulator harmonic may be used.

A polychromatic approach may be in order if the experimental conditions become increasingly demanding and the diffraction patterns become increasingly rich in information density. When the monochromatic model starts to fail, the simulations in Section 5.3 suggest that a model consisting of a non-dispersive probe with chromatic probe modes is the next best candidate. This model could be used in first instance as a small correction where the additional spectral modes pick up the flux that would otherwise cause artefacts in the object reconstruction. Furthermore, chromatic probe modes even allow to retrieve the spectrum of the illumination with astonishing accuracy. Section 5.4 demonstrated that the entire spectral density of the incoming illumination may be recovered from a single ptychographic datset.

However, spectral "modes" do not compose an Eigenvector decomposition as every spectral component is independent statistically from neighbouring spectral modes. As such it does not seem wise to allow an infinite amount of modes which in turn is theoretically allowed for probe modes of the mutual intensity. Similarly, the dispersion relation of the propagator does in fact help as it disentangles contributions from neighbouring spectral components. If the propagator would be indifferent towards the anticipated sampling density $\triangle\,\lambda$, the chromatic modes of the object o_λ would behave as if they were conventional object modes o_n of the mutual intensity. In this case, the spectral modes give up individual meaning and the result becomes ambiguous. Although not tested here, we may thus predict that it is not possible to

place an undulator spectrum to cover a range of absorption edges of the specimen and expect chemical sensitivity in the reconstruction.

We did not go into detail about the development of spectral modes during reconstruction. When starting from a box spectrum, spectral modes usually evolve in a gradual and slow process. The process lacks the typical tipping point at which the algorithm usually decides on a base set of modes for the mutual intensity. Moreover, it is not uncommon for spectral modes to "negotiate" their power contribution. The intensity of individual spectral modes overshoots or undershoots first and converges in an oscillatory manner to the final value.

6. Summary and Outlook

Ptychography scans the specimen through a coherent illumination and images the diffraction contrast for each transversal shift. As it samples the specimen both in real and reciprocal space simultaneously, ptychography creates high-dimensional and possibly redundant images of phase space. In **Chapter 2** we have seen that imaging partially coherent wave fields as described by second order correlation functions requires phase-spaces of higher dimensionality than classical single pattern CDI. At the expense of reduced redundancy, ptychography can accurately incorporate partial coherence in its phase-space. These partially coherent conditions require substantial alterations to the forward diffraction model in terms of a coherent mode expansion, both for spatial and temporal coherence. Even though these alterations have been known in the CDI community (Whitehead et al., 2009; Flewett et al., 2009; Abbey et al., 2011; Chen et al., 2009), ptychography appears to be the single diffraction method with the potential to retrieve entire correlation functions with no prior knowledge about their cause.

Even though not limited to, many methods and ideas presented in this thesis used a far-field imaging geometry. In far-field geometry, sampling the probe-object shift and the diffraction angles correspond to orthogonal independent planes in phase-space. For higher Fresnel numbers, Nugent (2007) noted that these planes do not span a right angle as paraxial propagation corresponds to a rotation of the sampling planes in phase space. With recent techniques like near-field ptychography (Stockmar et al., 2013) or multi-distance ptychography (Robisch and Salditt, 2013), the future may withhold other schemes that sample phase space along sets of planes not yet considered. Due to a non-orthogonal sampling of phase space, it may no longer be possible to separate effects from vibrations and partial spatial coherence in offset density function and point spread function, respectively.

In contrast to conventional microscopy, ptychographic imaging does not require any objective lensing system but instead relies on iterative phase-retrieval algorithms. They utilise the redundancy in the data to reconstruct the wave field of the illumination *and* the complex-valued transmission of the specimen at diffraction limited resolution. Ptychographic algorithms have traditionally been implemented in software on a per-instrument basis in various degrees of user-friendliness and sophistication.

Chapter 3 detailed a novel software framework, called Ptypy, which is designed to serve a plethora of use-cases, platforms and instruments. The software design itself is unique in its formalism. It provides a convenient abstraction layer from the physical model in order to allow or concise algorithmic implementations and portability across setup geometries. The chapter provided insight in Ptypy's design principles, structure and API as well as a step-by-step guide about how to implement reconstruction engines. The capabilities of the framework were illustrated with probe and object reconstructions from visible light and X-ray data.

Ptypy has already started to accrue a user-base at the Diamond Light Source and the European Synchrotron Radiation Facility. In order to lower the entry barrier for new groups, ptychographic software should be available to every researcher in the field. With establishing and providing a software framework for every interested academic party at `http://ptycho.github.com/ptypy`, we may incite a long-missing debate about parameter standards and proliferation of ptychographic data and reconstructions.

A major feature of Ptypy is its capability to include and handle complicated models such as the mode decomposition for probe and object also known as mixed-states (Thibault and Menzel, 2013). **Chapter 4** demonstrated a broad range of experimental application for this extended model:

An experiment at ID22NI in Section 4.1 demonstrated the feasibility of high-flux ptychography using an unfiltered undulator harmonic and a scintillator-coupled CCD detector. Here, the mode decomposition of the illumination proves manifestly beneficial compared to other attempted corrective measures such as background subtraction. It even allows to compensate for inhomogeneous response of the detector quadrants. With the relaxed constraints on experimental conditions, these results pave the way to higher resolutions and faster image acquisition at high-flux synchrotron end-stations.

In Section 4.2 decoherence was introduced in a synchrotron experiment by an oscillatory movement of the specimen stage. The analysis showed that the extended model has the capabilities to compensate for vibrations or similar effects which can be formulated as a convolution of the scan coordinate with an offset density function. Here, both probe and object mode decompositions were applied to compensate for the vibrations. Furthermore, the section detailed how the offset density function may be deconvolved from either set of reconstructed modes. These results provide conceptional understanding and analysis tools for vibration modes which are particularly useful in view of on-the-fly data acquisition planned at various synchrotron beamlines (Pelz et al., 2014; Deng et al., 2015a)

Though the extended diffraction model is very useful it also enlarges the parameter

space and requires more computations. As much as the model complexity increases, the analyst needs to plan more carefully the starting conditions and engine parameters. For example, the starting conditions influence the tipping point when the algorithm starts to fill the modes with the appropriate wave fields and begins to converge. The tipping point in turn sets the minimum number of iterations. Faced with increased complexity, some researcher may refrain from a mixed state reconstruction and rather settle for a higher discrepancy between a coherent model and physical reality. Future algorithmic developments of ptychography will need to flatten today's steep learning curve by automatically choosing and adapting reconstruction parameters.

Fundamentally different from imaging under partial spatial coherence is ptychography with partial temporal coherence. In **Chapter 5** we reviewed in detail the effect of an illumination with a broad spectral bandwidth. In Section 5.1 we have seen that it takes a while before the broad spectrum introduces a significant error. In far-field conditions with speckle rich diffraction patterns, the dispersion of the free diffraction path is the dominant cause for deviation from a monochromatic response. Even then, bandwidths up to $\Lambda < 3\,\%$ are readily tolerated by a monochromatic reconstruction model. This surprising result was demonstrated both in simulation in Section 5.3.1 and in the ptychography experiment of Section 4.1 where the full "pink" spectrum of an undulator harmonic probed the specimen.

Occasionally, the experimental conditions require an advanced model to cope with a broadband spectra. In that case, it makes most sense to allow for a spectral "mode" expansion for the probe only. The simulation indicated, that this model delivers the best fit for the chosen far-field conditions. The recovered spectral modes in turn, allow to determine the effective spectrum of the illumination. The model appears robust enough to recover even ultra-broad spectra ($\Lambda > 30\,\%$) as produced by *white* sources without any a-priori information. This capability was successfully demonstrated in a visible-light experiment where a white LED served as primary light source.

However, the chapter could barely scratch the surface of the parameter space. Only far-field conditions were considered and the selection of dispersion relations of probe and object was limited to three each. Further analysis is required, especially for "thick" specimens as used in experiments for ptycho-tomography (Zanette et al., 2015; Dierolf et al., 2010a) or near-field ptychography (Stockmar et al., 2015b) where the total phase shift from air to specimen center may exceed a few 2π. The near-field regime or regimes of other Fresnel numbers may inherently react differently to a broadband spectrum and warrant further investigation.

Concluding remarks

Diffraction contrast is a unique characteristic of coherent light sources. Ptychography and other CDI techniques, are therefore especially well suited to making good use of

the high-brilliance X-rays produced by 3rd and 4th generation synchrotron sources. They will continue to benefit from the overall trend in synchrotron technology towards higher coherent yields. In fact, many scanning X-ray microscopy beamlines consider an upgrade to ptychographic phase-retrieval in the near future, which in turn increases the demand for access to ptychographic reconstruction software.

Additionally, small-scale, partially coherent X-ray sources may eventually allow X-ray ptychography in the laboratories, for example, in form of a higher-harmonic laser source (Rothhardt et al., 2014) or a liquid metal jet (Zanette et al., 2014) or a small-scale synchrotron (Schleede et al., 2012). The coherence of these sources will be smaller than that of a synchrotron source. Hence, it stands to reason that a higher coherent photon yield will imply a higher cost – be it the capital investment in the machine or the duration of the experiments. In times of ever increasing, cheaper and omnipresent computation power the researcher may decide in favour of an advanced but computational intense algorithm rather than a simpler coherent algorithm in conjunction with additional cost.

With providing a reconstruction software and demonstrating the applicability to non-ideal, partially coherent conditions this thesis contributes to both aspects and thus pushes ptychography further on its path towards a wide-spread and established microscopy technique.

Bibliography

B. Abbey, L. W. Whitehead, H. M. Quiney, D. J. Vine, G. A. Cadenazzi, C. A. Henderson, K. A. Nugent, E. Balaur, C. T. Putkunz, A. G. Peele, G. J. Williams, and I. McNulty. Lensless imaging using broadband X-ray sources. *Nat Photon*, 5(7):420–424, 2011. doi:10.1038/nphoton.2011.125. (Cited on pages 3 and 137.)

J. Als-Nielsen and D. McMorrow. *Elements of Modern X-ray Physics: Second Edition*. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2011. doi:10.1002/9781119998365. (Cited on page 101.)

R. H. T. Bates. Fourier phase problems are uniquely solvable in more than one dimension. 1. Underlying theory. *Optik*, 61(3):247–262, 1982. (Cited on pages 17 and 20.)

D. J. Batey, D. Claus, and J. M. Rodenburg. Information multiplexing in ptychography. *Ultramicroscopy*, 138:13–21, 2014. doi:10.1016/j.ultramic.2013.12.003. (Cited on pages 122 and 124.)

H. H. Bauschke, P. L. Combettes, and D. R. Luke. Phase retrieval, error reduction algorithm, and Fienup variants: a view from convex optimization. *Josa a*, 19(7):1334–45, 2002. doi:10.1364/JOSAA.19.001334. (Cited on page 17.)

G. Binnig, C. F. Quate, and C. Gerber. Atomic force microscope. *Physical Review Letters*, 56(9):930–933, 1986. doi:10.1103/PhysRevLett.56.930. (Cited on page 38.)

G. Binnig and H. Rohrer. Scanning tunneling microscopy from birth to adolescence. *Reviews of Modern Physics*, 59(3):615–625, 1987. doi:10.1103/RevModPhys.59.615. (Cited on page 38.)

G. Binning, H. Rohrer, C. Gerber, and E. Weibel. Surface studies by scanning tunneling microscopy. *Physical Review Letters*, 49(1):57–61, 1982. doi:10.1103/PhysRevLett.49.57. (Cited on page 38.)

M. Born, E. Wolf, and A. B. Bhatia. *Principles of Optics: Electromagnetic Theory of Propagation, Interference and Diffraction of Light*. Cambridge University Press, 7 edn., 1999. (Cited on pages 23, 31, and 36.)

O. Bunk, M. Dierolf, S. Kynde, I. Johnson, O. Marti, and F. Pfeiffer. Influence of the overlap parameter on the convergence of the ptychographical iterative engine. *Ultramicroscopy*, 108(5):481–487, 2008. doi:10.1016/j.ultramic.2007.08.003. (Cited on pages 45 and 67.)

V. Chamard, M. Allain, P. Godard, A. Talneau, G. Patriarche, and M. Burghammer. Strain in a silicon-on-insulator nanostructure revealed by 3D x-ray Bragg ptychography. *Scientific reports*, 5:9827, 2015. doi:10.1038/srep09827. (Cited on page 3.)

H. N. Chapman. Phase-retrieval X-ray microscopy by Wigner-distribution deconvolution. *Ultramicroscopy*, 66(3-4):153–172, 1996. doi:10.1016/S0304-3991(96)00084-8. (Cited on page 40.)

B. Chen, R. A. Dilanian, S. Teichmann, B. Abbey, A. G. Peele, G. J. Williams, P. Hannaford, L. Van Dao, H. M. Quiney, and K. A. Nugent. Multiple wavelength diffractive imaging. *Physical Review A - Atomic, Molecular, and Optical Physics*, 79(2):023809, 2009. doi:10.1103/PhysRevA.79.023809. (Cited on page 137.)

J. N. Clark, X. Huang, R. J. Harder, and I. K. Robinson. Dynamic imaging using ptychography. *Physical Review Letters*, 112(11):1–5, 2014. doi:10.1103/PhysRevLett.112.113901. (Cited on pages 3, 53, 55, and 96.)

J. N. Clark and A. G. Peele. Simultaneous sample and spatial coherence characterisation using diffractive imaging. *Applied Physics Letters*, 99(15):154103, 2011. doi:10.1063/1.3650265. (Cited on page 75.)

J. W. Cooley and J. W. Tukey. An algorithm for machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, 1965. doi:10.1090/S0025-5718-1965-0178586-1. (Cited on page 19.)

L. Dalcín, R. Paz, M. Storti, and J. D'Elía. MPI for Python: Performance improvements and MPI-2 extensions. *Journal of Parallel and Distributed Computing*, 68(5):655–662, 2008. doi:10.1016/j.jpdc.2007.09.005. (Cited on page 58.)

B. J. Daurer. *Advanced Image Processing Algorithms for Coherent X-ray nanoCT*. M.sc. thesis, Technische Universität München, 2013. (Cited on page 51.)

J. Deng, Y. S. G. Nashed, S. Chen, N. W. Phillips, T. Peterka, R. Ross, S. Vogt, C. Jacobsen, and D. J. Vine. Continuous motion scan ptychography: characterization for increased speed in coherent x-ray imaging. *Opt. Express*, 23(5):5438–5451, 2015a. doi:10.1364/OE.23.005438. (Cited on pages 55, 73, 97, and 138.)

J. Deng, D. J. Vine, S. Chen, Y. S. G. Nashed, Q. Jin, N. W. Phillips, T. Peterka, R. Ross, S. Vogt, and C. J. Jacobsen. Simultaneous cryo X-ray ptychographic and fluorescence microscopy of green algae. *Proceedings of the National Academy of Sciences of the United States of America*, 112(8):2314–9, 2015b. doi:10.1073/pnas.1413003112. (Cited on page 55.)

M. Dierolf, A. Menzel, P. Thibault, P. Schneider, C. M. Kewish, R. Wepf, O. Bunk, and F. Pfeiffer. Ptychographic X-ray computed tomography at the nanoscale. *Nature*, 467(7314):436–439, 2010a. doi:10.1038/nature09419. (Cited on pages 3, 55, and 139.)

M. Dierolf, P. Thibault, A. Menzel, C. M. Kewish, K. Jefimovs, U. Schlichting, K. Von König, O. Bunk, and F. Pfeiffer. Ptychographic coherent diffractive imaging of weakly scattering specimens. *New Journal of Physics*, 12(3):035017, 2010b. doi:10.1088/1367-2630/12/3/035017. (Cited on pages 47, 55, and 76.)

M. J. Dierolf. *Ptychographic X-ray Microscopy and Tomography*. Ph.D. thesis, TU München, 2015. (Cited on pages 2, 51, and 52.)

T. B. Edo, D. J. Batey, A. M. Maiden, C. Rau, U. Wagner, Z. D. Pesic, T. A. Waigh, and J. M. Rodenburg. Sampling in x-ray ptychography. *Physical Review A - Atomic, Molecular, and Optical Physics*, 87(5):1–8, 2013. doi:10.1103/PhysRevA.87.053850. (Cited on page 2.)

S. Eisebitt, J. Lüning, W. F. Schlotter, M. Lörgen, O. Hellwig, W. Eberhardt, and J. Stöhr. Lensless imaging of magnetic nanostructures by X-ray spectro-holography. *Nature*, 432(7019):885–888, 2004. doi:10.1038/nature03139. (Cited on pages 1 and 18.)

V. Elser. Phase retrieval by iterated projections. *Journal of the Optical Society of America A*, 20(1):40–55, 2002. doi:10.1364/JOSAA.20.000040. (Cited on pages 17, 21, and 48.)

V. Elser. Solution of the crystallographic phase problem by iterated projections. *Acta Crystallographica Section A: Foundations of Crystallography*, 59(3):201–209, 2003. doi:10.1107/S0108767303002812. (Cited on pages 17, 21, and 48.)

B. Enders, M. Dierolf, P. Cloetens, M. Stockmar, F. Pfeiffer, and P. Thibault. Ptychography with broad-bandwidth radiation. *Applied Physics Letters*, 104(17), 2014. doi:10.1063/1.4874304. (Cited on pages 3, 4, 53, 96, and 101.)

B. Enders, K. Giewekemeyer, T. Kurz, S. Podorov, and T. Salditt. Non-iterative coherent diffractive imaging using a phase-shifting reference frame. *New Journal of Physics*, 11, 2009. doi:10.1088/1367-2630/11/4/043021. (Cited on pages 1 and 18.)

H. M. L. Faulkner and J. M. Rodenburg. Movable aperture lensless transmission microscopy: A novel phase retrieval algorithm. *Physical Review Letters*, 93(2):023903–1, 2004. doi:10.1103/PhysRevLett.93.023903. (Cited on pages 45 and 47.)

J. R. Fienup. Phase retrieval algorithms: a comparison. *Applied Optics*, 21(15):2758–2769, 1982. doi:10.1364/AO.21.002758. (Cited on page 17.)

J. R. Fienup. Reconstruction of a complex-valued object from the modulus of its Fourier transform using a support constraint. *Journal of the Optical Society of America A*, 4(1):118–123, 1986. doi:10.1364/JOSAA.4.000118. (Cited on pages 2, 17, 20, 21, and 47.)

S. Flewett, H. Quiney, C. Tran, and K. Nugent. Extracting coherent modes from partially coherent wavefields. *Optics letters*, 34(14):2198–2200, 2009. doi:10.1364/OL.34.002198. (Cited on page 137.)

R. W. Gerchberg and W. O. Saxton. A practical algorithm for the determination of phase from image and diffraction plane pictures. *Optik*, 35(2):237–246, 1972. doi:10.1070/QE2009v039n06ABEH013642. (Cited on pages 2, 17, 21, and 47.)

K. Giewekemeyer, P. Thibault, S. Kalbfleisch, A. Beerlink, C. M. Kewish, M. Dierolf, F. Pfeiffer, and T. Salditt. Quantitative biological imaging by ptychographic x-ray diffraction microscopy. *Proceedings of the National Academy of Sciences of the United States of America*, 107(2):529–534, 2010. doi:10.1073/pnas.0905846107. (Cited on page 69.)

P. Godard, G. Carbone, M. Allain, F. Mastropietro, G. Chen, L. Capello, a. Diaz, T. H. Metzger, J. Stangl, and V. Chamard. Three-dimensional high-resolution quantitative microscopy of extended crystals. *Nature communications*, 2:568, 2011. doi:10.1038/ncomms1569. (Cited on page 3.)

J. W. Goodman. *Statistical Optics*. Wiley, 2 edn., 1985. (Cited on pages 23, 25, and 32.)

J. W. Goodman. *Introduction to Fourier Optics*. McGraw-Hill series in electrical and computer engineering. McGraw-Hill, 2 edn., 2005. (Cited on pages 7, 9, 11, and 52.)

B. E. Granger and M. Ragan-Kelley. Python bindings for ZeroMQ. 2015. (Cited on page 58.)

M. Guizar-Sicairos and J. R. Fienup. Direct image reconstruction from a Fourier intensity pattern using HERALDO. *Optics letters*, 33(22):2668–2670, 2008a. doi:10.1364/OL.33.002668. (Cited on pages 1 and 18.)

M. Guizar-Sicairos and J. R. Fienup. Phase retrieval with transverse translation diversity: a nonlinear optimization approach. *Optics Express*, 16(10):7264–7278, 2008b. doi:10.1117/12.792622. (Cited on pages 2 and 47.)

M. Guizar-Sicairos, I. Johnson, A. Diaz, M. Holler, P. Karvinen, H.-C. Stadler, R. Dinapoli, O. Bunk, and A. Menzel. High-throughput ptychography using Eiger: scanning X-ray nano-imaging of extended regions. *Optics Express*, 22(12):14859–14870, 2014. doi:10.1364/OE.22.014859. (Cited on pages 51 and 52.)

R. Hegerl and W. Hoppe. Dynamische Theorie der Kristallstrukturanalyse durch Elektronenbeugung im inhomogenen Primärstrahlwellenfeld. *Ber. Bunsenges. phys. Chem.*, 74(11):1148–1154, 1970. doi:10.1002/bbpc.19700741112. (Cited on page 2.)

B. Henrich, J. Becker, R. Dinapoli, P. Goettlicher, H. Graafsma, H. Hirsemann, R. Klanner, A. K. Srivastava, U. Trunk, and C. Youngman. The adaptive gain integrating pixel detector AGIPD a detector for the European XFEL. *Nuclear Inst. and Methods in Physics Research, A*, 633:1–4, 2010. doi:10.1016/j.nima.2010.06.107. (Cited on page 80.)

G. Herink, D. R. Solli, M. Gulde, and C. Ropers. Field-driven photoemission from nanostructures quenches the quiver motion. *Nature*, 483(7388):190–193, 2012. (Cited on page 123.)

O. Hignette, P. Cloetens, G. Rostaing, P. Bernard, and C. Morawe. Efficient sub 100 nm focusing of hard x rays. *Review of Scientific Instruments*, 76(6):063709, 2005. doi:10.1063/1.1928191. (Cited on page 75.)

P. Hintjens. *ZeroMQ*. O'Reilly Media, 2013. (Cited on page 58.)

M. R. Howells, T. Beetz, H. N. Chapman, C. Cui, J. M. Holton, C. J. Jacobsen, J. Kirz, E. Lima, S. Marchesini, H. Miao, D. Sayre,

D. A. Shapiro, J. C. H. Spence, and D. Starodub. An assessment of the resolution limitation due to radiation-damage in X-ray diffraction microscopy. *Journal of Electron Spectroscopy and Related Phenomena*, 170(1-3):4–12, 2009. doi:10.1016/j.elspec.2008.10.008. (Cited on page 73.)

X. Huang, H. Yan, R. Harder, Y. Hwu, I. K. Robinson, and Y. S. Chu. Optimization of overlap uniformness for ptychography. *Optics express*, 22(10):12634–12644, 2014. doi:10.1364/OE.22.012634. (Cited on page 45.)

I. Johnson, A. Bergamaschi, J. Buitenhuis, R. Dinapoli, D. Greiffenberg, B. Henrich, T. Ikonen, G. Meier, A. Menzel, A. Mozzanica, V. Radicci, D. K. Satapathy, B. Schmitt, and X. Shi. Capturing dynamics with Eiger, a fast-framing X-ray detector. *Journal of Synchrotron Radiation*, 19(6):1001–1005, 2012. doi:10.1107/S0909049512035972. (Cited on page 1.)

P. Kirkpatrick and A. V. Baez. Formation of Optical Images by X-Rays. *J. Opt. Soc. Am.*, 38(9):766–774, 1948. doi:10.1364/JOSA.38.000766. (Cited on page 75.)

P. Kraft, A. Bergamaschi, C. Broennimann, R. Dinapoli, E. F. Eikenberry, B. Henrich, I. Johnson, A. Mozzanica, C. M. Schlepütz, P. R. Willmott, and B. Schmitt. Performance of single-photon-counting PILATUS detector modules. *Journal of Synchrotron Radiation*, 16(3):368–375, 2009. doi:10.1107/S0909049509009911. (Cited on page 1.)

D. R. Luke. Relaxed Averaged Alternating Reflections for Diffraction Imaging. *Inverse Problems*, 37(1):13, 2004. doi:10.1088/0266-5611/21/1/004. (Cited on page 47.)

F. R. N. C. Maia. The Coherent X-ray Imaging Data Bank. *Nature Methods*, 9(9):854–855, 2012. doi:10.1038/nmeth.2110. (Cited on page 57.)

a. M. Maiden, M. J. Humphry, and J. M. Rodenburg. Ptychographic transmission microscopy in three dimensions using a multi-slice approach. *Journal of the Optical Society of America A*, 29(8):1606, 2012. doi:10.1364/JOSAA.29.001606. (Cited on page 47.)

A. M. Maiden and J. M. Rodenburg. An improved ptychographical phase retrieval algorithm for diffractive imaging. *Ultramicroscopy*, 109(10):1256–1262, 2009. doi:10.1016/j.ultramic.2009.05.012. (Cited on pages 2 and 47.)

L. Mandel and E. Wolf. Coherence properties of optical fields. *Reviews of Modern Physics*, 37(2):231–287, 1965. doi:10.1103/RevModPhys.37.231. (Cited on page 24.)

S. Marchesini. A unified evaluation of iterative projection algorithms for phase retrieval. 2007. doi:10.1063/1.2403783. (Cited on pages 2 and 17.)

S. Marchesini, H. He, H. N. Chapman, S. P. Hau-Riege, A. Noy, M. R. Howells, U. Weierstall, and J. C. H. Spence. X-ray image reconstruction from a diffraction pattern alone. *Phys. Rev. B: Condens. Matter Mater. Phys.*, 68(14):140101/1–140101/4, 2003. doi:Artn 140101 Doi 10.1103/Physrevb.68.140101. (Cited on page 2.)

S. Marchesini, A. Schirotzek, F. Maia, and C. Yang. Augmented projections for ptychographic imaging. *arXiv preprint arXiv: ...*, 115009(11):18, 2012. doi:10.1088/0266-5611/29/11/115009. (Cited on page 47.)

G. Martínez-Criado, R. Tucoulou, P. Cloetens, P. Bleuet, S. Bohic, J. Cauzid, I. Kieffer, E. Kosior, S. Labouré, S. Petitgirard, A. Rack, J. A. Sans, J. Segura-Ruiz, H. Suhonen, J. Susini, and J. Villanova. Status of the hard X-ray microprobe beamline ID22 of the European Synchrotron Radiation Facility. *Journal of Synchrotron Radiation*, 19(1):10–18, 2012. doi:10.1107/S090904951104249X. (Cited on page 75.)

B. C. McCallum and J. M. Rodenburg. Simultaneous reconstruction of object and aperture functions from multiple far-field intensity measurements. *Journal of the Optical Society of America A*, 10(2):231, 1993. doi:10.1364/JOSAA.10.000231. (Cited on page 40.)

J. Miao, P. Charalambous, J. Kirz, and D. Sayre. Extending the methodology of X-ray crystallography to allow imaging of micrometre-sized non-crystalline specimens. *Nature*, 400(6742):342–344, 1999. doi:10.1038/22498. (Cited on pages 1 and 47.)

J. Miao and D. Sayre. On possible extensions of X-ray crystallography through diffraction-pattern oversampling. *Acta Crystallographica Section A: Foundations of Crystallography*, 56(6):596–605, 2000. doi:10.1107/S010876730001031X. (Cited on page 20.)

J. Miao, D. Sayre, and H. N. Chapman. Phase retrieval from the magnitude of the Fourier transforms of nonperiodic objects. *Journal of the Optical Society of America A*, 15(6):1662, 1997. doi:10.1364/JOSAA.15.001662. (Cited on page 20.)

Y. S. G. Nashed, D. J. Vine, T. Peterka, J. Deng, R. Ross, and C. Jacobsen. Parallel ptychographic reconstruction. *Optics Express*, 22(26):32082–32097, 2014. doi:10.1364/OE.22.032082. (Cited on page 55.)

K. A. Nugent. X-ray noninterferometric phase imaging: a unified picture. *Journal of the Optical Society of America A*, 24(2):536–547, 2007. doi:10.1364/JOSAA.24.000536. (Cited on pages 39 and 137.)

K. A. Nugent. Coherent methods in the X-ray sciences. *Advances in Physics*, 59(1):1–99, 2009. doi:10.1080/00018730903270926. (Cited on page 1.)

T. E. Oliphant. Python for scientific computing. *Computing in Science and Engineering*, 9(3):10–20, 2007. doi:10.1109/MCSE.2007.58. (Cited on page 57.)

X. Ou, R. Horstmeyer, C. Yang, and G. Zheng. Quantitative phase imaging via Fourier ptychographic microscopy. *Optics letters*, 38(22):4845–8, 2013. doi:10.1364/OL.38.004845. (Cited on page 3.)

A. L. Patterson. A Direct Method for the Determination of the Components of Interatomic Distances in Crystals. *Z. Krist.*, 90:517, 1935. (Cited on page 18.)

P. M. Pelz, M. Guizar-Sicairos, P. Thibault, I. Johnson, M. Holler, and A. Menzel. On-the-fly scans for X-ray ptychography. *Applied Physics Letters*, 105(25):251101, 2014. doi:10.1063/1.4904943. (Cited on pages 53, 73, 97, and 138.)

F. Pfeiffer, C. Kottler, O. Bunk, and C. David. Hard X-ray phase tomography with low-brilliance sources. *Physical Review Letters*, 98(10):108105, 2007. doi:10.1103/PhysRevLett.98.108105. (Cited on pages 25 and 106.)

C. Ponchut. Characterization of X-ray area detectors for synchrotron beamlines. *Journal of Synchrotron Radiation*, 13(2):195–203, 2006. doi:10.1107/S0909049505034278. (Cited on pages 74 and 76.)

A. Robisch and T. Salditt. Phase retrieval for object and probe using a series of defocus near-field images. *Optics express*, 21(20):23345–23357, 2013. doi:10.1364/OE.21.023345. (Cited on pages 53, 55, and 137.)

J. M. Rodenburg. Ptychography and related diffractive imaging methods. *Advances in Imaging and Electron Physics*, 150(07):87–184, 2008. doi:10.1016/S1076-5670(07)00003-1. (Cited on page 2.)

J. Rothhardt, S. Hädrich, A. Klenke, S. Demmler, A. Hoffmann, T. Gotschall, T. Eidam, M. Krebs, J. Limpert, and A. Tünnermann. 53W average power few-cycle fiber laser system generating soft x rays up to the water window. *Optics Letters*, 39(17):5224–5227, 2014. doi:10.1364/OL.39.005224. (Cited on page 140.)

D. Sayre. Some implications of a theorem due to Shannon. *Acta Crystallographica*, 5(6):843–843, 1952. doi:10.1107/S0365110X52002276. (Cited on page 18.)

D. Sayre. X-ray crystallography: The past and present of the phase problem. *Structural Chemistry*, 13(1):81–96, 2002. doi:10.1023/A:1013477415486. (Cited on page 1.)

D. Sayre and H. N. Chapman. X-ray microscopy. *Acta Crystallographica Section A*, 51(3):237–252, 1995. doi:10.1107/S0108767394011803. (Cited on page 1.)

S. Schleede, F. G. Meinel, M. Bech, J. Herzen, K. Achterhold, G. Potdevin, A. Malecki, S. Adam-Neumair, S. F. Thieme, F. Bamberg, K. Nikolaou, A. Bohla, A. Ö. Yildirim, R. Loewen, M. Gifford, R. Ruth, O. Eickelberg, M. Reiser, and F. Pfeiffer. Emphysema diagnosis using X-ray dark-field imaging at a laser-driven compact synchrotron light source. *Proceedings of the National Academy of Sciences*, 109(44):17880–17885, 2012. doi:10.1073/pnas.1206684109. (Cited on page 140.)

M. M. Seibert, T. Ekeberg, F. R. N. C. Maia, M. Svenda, J. Andreasson, O. Joensson, D. Odic, B. Iwan, A. Rocker, D. Westphal, M. Hantke, D. P. DePonte, A. Barty, J. Schulz, L. Gumprecht, N. Coppola, A. Aquila, M. Liang, T. A. White, A. Martin, C. Caleman, S. Stern, C. Abergel, V. Seltzer, J.-M. Claverie, C. Bostedt, J. D. Bozek, S. Boutet, A. A. Miahnahri, M. Messerschmidt, J. Krzywinski, G. Williams, K. O. Hodgson, M. J. Bogan, C. Y. Hampton, R. G. Sierra, D. Starodub, I. Andersson, S. Bajt, M. Barthelmess, J. C. H. Spence, P. Fromme, U. Weierstall, R. Kirian, M. Hunter, R. B. Doak, S. Marchesini, S. P. Hau-Riege, M. Frank, R. L. Shoeman, L. Lomb, S. W. Epp, R. Hartmann, D. Rolles, A. Rudenko, C. Schmidt, L. Foucar, N. Kimmel, P. Holl, B. Rudek, B. Erk, A. Hoemke, C. Reich, D. Pietschner, G. Weidenspointner, L. Strueder, G. Hauser, H. Gorke, J. Ullrich, I. Schlichting, S. Herrmann, G. Schaller, F. Schopper, H. Soltau, K.-U. Kuehnel, R. Andritschke, C.-D. Schroeter, F. Krasniqi, M. Bott, S. Schorb, D. Rupp, M. Adolph, T. Gorkhover, H. Hirsemann, G. Potdevin, H. Graafsma, B. Nilsson, H. N. Chapman, and J. Hajdu. Single mimivirus particles intercepted and imaged with an X-ray laser. *Nature*, 470(7332):78–U86, 2011. doi:10.1038/nature09748. (Cited on page 2.)

C. E. Shannon. Communication in the Presence of Noise. *Proceedings of the IRE*, 37(1):10–21, 1949. doi:10.1109/JRPROC.1949.232969. (Cited on page 18.)

D. A. Shapiro, Y.-S. Yu, T. Tyliszczak, J. Cabana, R. Celestre,

W. Chao, K. Kaznatcheev, A. L. D. Kilcoyne, F. Maia, S. Marchesini, Y. S. Meng, T. Warwick, L. L. Yang, and H. A. Padmore. Chemical composition mapping with nanometre resolution by soft X-ray microscopy. *Nature Photonics*, 8(10):765–769, 2014. doi:10.1038/nphoton.2014.207. (Cited on page 55.)

Sharp Camera Team. *Sharp Camera Documentation*. Available online (last accessed March 2nd, 2015): \url{http://sharpcamera.bitbucket.org}, 2014. (Cited on page 55.)

M. Sotomayor, M. Sotomayor, K. Schulten, K. Schulten, E. Evans, E. Evans, G. I. Bell, G. I. Bell, M. M. Davis, M. M. Davis, K. Ritchie, and K. Ritchie. Far-Field Optical Nanoscopy. *Cell*, 2(May):1153–1158, 2007. doi:10.1126/science.1137395. (Cited on page 38.)

J. C. H. Spence, U. Weierstall, and M. Howells. Phase recovery and lensless imaging by iterative methods in optical, X-ray and electron diffraction. *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, 360(1794):875–895, 2002. doi:10.1098/rsta.2001.0972. (Cited on page 17.)

M. Stockmar, P. Cloetens, I. Zanette, B. Enders, M. Dierolf, F. Pfeiffer, and P. Thibault. Near-field ptychography: phase retrieval for inline holography using a structured illumination. *Scientific reports*, 3:1927, 2013. doi:10.1038/srep01927. (Cited on pages 3, 52, 53, 97, 109, and 137.)

M. Stockmar, M. Hubert, M. Dierolf, B. Enders, R. Clare, S. Allner, A. Fehringer, I. Zanette, J. Villanova, J. Laurencin, P. Cloetens, F. Pfeiffer, and P. Thibault. X-ray nanotomography using near-field ptychography. *Optics Express*, 23(10):12720, 2015a. doi:10.1364/OE.23.012720. (Cited on page 52.)

M. Stockmar, I. Zanette, M. Dierolf, B. Enders, R. Clare, F. Pfeiffer, P. Cloetens, A. Bonnin, and P. Thibault. X-ray near-field ptychography for optically thick specimens. *Physical Review Applied*, 3(1):1–6, 2015b. doi:10.1103/PhysRevApplied.3.014005. (Cited on pages 3, 52, 96, 97, and 139.)

The HDF Group. Hierarchical Data Format, version 5. 1997. (Cited on page 57.)

P. Thibault. High-Resolution Scanning X-ray. *Science*, 379(July):379–383, 2008. doi:10.1126/science.1158573. (Cited on pages 2, 15, 47, 48, 55, 69, and 70.)

P. Thibault, M. Dierolf, O. Bunk, A. Menzel, and F. Pfeiffer. Probe retrieval in ptychographic coherent diffractive imaging. *Ultramicroscopy*, 109(4):338–343, 2009. doi:10.1016/j.ultramic.2008.12.011. (Cited on pages 2, 64, 76, and 94.)

P. Thibault and V. Elser. X-Ray Diffraction Microscopy. *Annual Review of Condensed Matter Physics*, 1(1):237–255, 2010. doi:10.1146/annurev-conmatphys-070909-104034. (Cited on page 1.)

P. Thibault and M. Guizar-Sicairos. Maximum-likelihood refinement for coherent diffractive imaging. *New Journal of Physics*, 14(6):063004, 2012. doi:10.1088/1367-2630/14/6/063004. (Cited on pages 46, 47, 49, 50, 62, and 69.)

P. Thibault and A. Menzel. Reconstructing state mixtures from diffraction measurements supplement. *Nature*, 494(7435):68–71, 2013. doi:10.1038/nature11806. (Cited on pages 3, 6, 29, 53, 75, 81, 89, 96, 97, 99, and 138.)

P. Trtik, A. Diaz, M. Guizar-Sicairos, A. Menzel, and O. Bunk. Density mapping of hardened cement paste using ptychographic X-ray computed tomography. *Cement and Concrete Composites*, 36(1):71–77, 2013. doi:10.1016/j.cemconcomp.2012.06.001. (Cited on page 3.)

S. Van Der Walt, S. C. Colbert, and G. Varoquaux. The NumPy array: A structure for efficient numerical computation. *Computing in Science and Engineering*, 13(2):22–30, 2011. doi:10.1109/MCSE.2011.37. (Cited on pages 57 and 59.)

K. Wakonig. *Influence of the scan pattern density on the reconstruction quality in coherent X-ray diffraction microscopy*. M.sc. thesis, Technische Universität München, 2015. (Cited on pages 45 and 51.)

Z. Wen, C. Yang, X. Liu, and S. Marchesini. Alternating direction methods for classical and ptychographic phase retrieval. *Inverse Problems*, 28(11):115010, 2012. doi:10.1088/0266-5611/28/11/115010. (Cited on page 57.)

L. W. Whitehead, G. J. Williams, H. M. Quiney, D. J. Vine, R. A. Dilanian, S. Flewett, K. A. Nugent, A. G. Peele, E. Balaur, and I. McNulty. Diffractive imaging using partially coherent X rays. *Physical Review Letters*, 103(24):243902, 2009. doi:10.1103/PhysRevLett.103.243902. (Cited on pages 3 and 137.)

R. N. Wilke, M. Vassholz, and T. Salditt. Semi-transparent central stop in high-resolution x-ray ptychography using kirkpatrick-baez focusing. *Acta Crystallographica Section A: Foundations of Crystallography*, 69(5):490–497, 2013. doi:10.1107/S0108767313019612. (Cited on pages 55 and 80.)

E. Wolf. New theory of partial coherence in the space-frequency domain. Part I: spectra and cross spectra of steady-state sources. *Journal of the Optical Society of*

America, 72(3):343–351, 1982. doi:10.1364/JOSA.72.000343. (Cited on pages 23, 24, and 28.)

E. Wolf. *Introduction to the Theory of Coherence and Polarization of Light*. Cambridge University Press, 2007. (Cited on pages 23, 29, 30, 31, and 32.)

I. Zanette, B. Enders, M. Dierolf, P. Thibault, R. Gradl, a. Diaz, M. Guizar-Sicairos, a. Menzel, F. Pfeiffer, and P. Zaslansky. Ptychographic X-ray nanotomography quantifies mineral distributions in human dentine. *Scientific reports*, 5:9210, 2015. doi:10.1038/srep09210. (Cited on pages 3 and 139.)

I. Zanette, T. Zhou, A. Burvall, U. Lundström, D. H. Larsson, M. Zdora, P. Thibault, F. Pfeiffer, and H. M. Hertz. Speckle-Based X-Ray Phase-Contrast and Dark-Field Imaging with a Laboratory Source. *Phys. Rev. Lett.*, 112(25):253903, 2014. doi:10.1103/PhysRevLett.112.253903. (Cited on page 140.)

F. Zhang, J. Vila-Comamala, A. Diaz, F. Berenguer, R. Bean, B. Chen, A. Menzel, I. K. Robinson, and J. M. Rodenburg. Translation position determination in ptychographic coherent diffraction imaging. *Optics Express*, 21(11):13592, 2013. doi:10.1364/OE.21.013592. (Cited on page 55.)

A. Appendix

A.1. Some conventions

Variables
If not indicated otherwise

- a variable in *bold-face* type represents a vector, $\mathbf{a} \in \mathbb{C}^n, \mathbb{R}^n$,
- a variable in *sans-serif* type represents an integral number, $\mathsf{a} \in \mathbb{Z}, \mathbb{N}$,
- and a variable in the common *italic* type is usually a scalar, $a \in \mathbb{R}, \mathbb{C}$.

Fourier transform
For functions $g, h : \mathbf{x} \in \mathbb{R}^n \to g(\mathbf{x}) \in \mathbb{C}$, we denote

$$\mathcal{F}g(\mathbf{v}) \equiv \int_{\mathbb{R}^n} g(\mathbf{x})\, e^{-2\pi \mathrm{i}\, \mathbf{x}\mathbf{v}} \mathrm{d}\mathbf{x}\,, \tag{A.1}$$

as the *Fourier transform* of g and indicate that with the label $\mathcal{F}$. $\mathbf{v} \in \mathbb{R}^n$ is called the *reciprocal coordinate* to $\mathbf{x}$. Usually, the domain of integration is inferred from the integration variable (here $\mathrm{d}\mathbf{x}$) and the domain subscript $\mathbb{R}^n$ is dropped when the integral extends over the entire vector space. The inverse transformation to (A.1) is called the *inverse Fourier transform* and indicated by the label $\mathcal{F}^*$,

$$\mathcal{F}^* f(\mathbf{x}) \equiv \int f(\mathbf{v})\, e^{2\pi \mathrm{i}\, \mathbf{x}\mathbf{v}} \mathrm{d}\mathbf{v}\,. \tag{A.2}$$

If the integration is carried out on a subspace of $\mathbb{R}^n$, we indicate that with a subscript to the label $\mathcal{F}$. For example for $(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^2 \times \mathbb{R}^2$ we write the two-dimensional Fourier transform with respect to $\mathbf{x}$ as,

$$\mathcal{F}_{\mathbf{x}} \left\{ g(\mathbf{x}, \mathbf{y}) \right\} (\mathbf{v}, \mathbf{y}) \equiv \int g(\mathbf{x}, \mathbf{y})\, e^{-2\pi \mathrm{i}\, \mathbf{x}\mathbf{v}} \mathrm{d}\mathbf{x}\,, \tag{A.3}$$

which also allows to write explicit formulas, e.g. $\mathcal{F}_{\mathbf{x}} \left\{ \mathbf{x}^2 + \mathbf{y}\mathbf{x} \right\} (\mathbf{v})$. In cases where the argument $\mathbf{v}$ of the Fourier transform does not need to be accessible in an equation,

we can use a shorter form,

$$\underset{\mathbf{x},\mathbf{v}}{\mathcal{F}}\{g(\mathbf{x},\mathbf{y})\} \equiv \int g(\mathbf{x},\mathbf{y})\, e^{-2\pi \mathrm{i}\,\mathbf{x}\mathbf{v}} \mathrm{d}\mathbf{x}\,. \tag{A.4}$$

Whether expression (A.1), (A.3) or (A.4) is used, will depend on the context and complexity of the equations.

Convolution and cross-correlation
For two functions $g, h : \mathbf{x} \in \mathbb{R}^n \to g(\mathbf{x}) \in \mathbb{C}$, we denote

$$f(\mathbf{x}) \star g(\mathbf{x}) \equiv (f \star g)(\mathbf{x}) = \int f^*(\mathbf{y})\, g(\mathbf{y}+\mathbf{x})\, \mathrm{d}\mathbf{y} \tag{A.5}$$

as the *cross-correlation* and

$$f(\mathbf{x}) \otimes g(\mathbf{x}) \equiv (f \otimes g)(\mathbf{x}) = \int f(\mathbf{y})\, g(\mathbf{x}-\mathbf{y})\, \mathrm{d}\mathbf{y} \tag{A.6}$$

as the *convolution* of f and g. Both correlations may be expressed in terms of Fourier transforms,

$$\mathcal{F}(f \otimes g) = \mathcal{F}f \cdot \mathcal{F}g \quad \text{and} \quad \mathcal{F}(f \star g) = \mathcal{F}f \cdot (\mathcal{F}g)^* \tag{A.7}$$

where the asterisk represents complex conjugation. If either convolution or cross-correlation is carried out on a subspace of $\mathbb{R}^n$, we indicate that with a subscript to the label "$\otimes$" or "$\star$" as in the following example:

$$f(\mathbf{x},\mathbf{v}) \otimes_{\mathbf{v}} g(\mathbf{x},\mathbf{v}) \equiv \int f(\mathbf{x},\mathbf{w})\, g(\mathbf{x},\mathbf{v}-\mathbf{w})\, \mathrm{d}\mathbf{w}\,. \tag{A.8}$$

Common functions
The `rect`-function is the characteristic function for the unit cube $\left[-\frac{1}{2},\frac{1}{2}\right] \times \left[-\frac{1}{2},\frac{1}{2}\right] \times \ldots \subset \mathbb{R}^n$, or in explicit form:

$$\mathrm{rect}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \left[-\frac{1}{2},\frac{1}{2}\right] \times \left[-\frac{1}{2},\frac{1}{2}\right] \times \ldots \\ 0 & \text{otherwise}\,. \end{cases} \tag{A.9}$$

The `sinc`-function is the Fourier transform to the `rect`-function,

$$\mathrm{sinc}(\mathbf{v}) = \mathcal{F}_{\mathbf{x}}\left\{\mathrm{rect}(\mathbf{x})\right\}(\mathbf{v})\,. \tag{A.10}$$

For example in two dimensions, the explicit form may be written as,

$$\mathrm{sinc}(\mathbf{v}) = \mathrm{sinc}((v_{\mathrm{I}}, v_{\mathrm{II}})) = \frac{\sin(\pi v_{\mathrm{I}})}{\pi v_{\mathrm{I}}} \frac{\sin(\pi v_{\mathrm{II}})}{\pi v_{\mathrm{II}}}\,, \tag{A.11}$$

which is the direct product of the `sinc`-function for each cartesian coordinate.

A.2. Deconvolution from an eigenvector decomposition

We seek to minimize the following cost-function:

$$\mathcal{L}(\psi,\mu) = \iint \left|\Phi(\mathbf{x}_1,\mathbf{x}_2) - \mu(\mathbf{x}_1-\mathbf{x}_2)\psi^*(\mathbf{x}_1)\psi(\mathbf{x}_2)\right|^2 \mathrm{d}\mathbf{x}_1\mathrm{d}\mathbf{x}_2 \tag{A.12}$$

where Φ can be expressed in terms of (orthogonal) modes

$$\Phi(\mathbf{x}_1,\mathbf{x}_2) = \sum_n \phi_n^*(\mathbf{x}_1)\phi_n(\mathbf{x}_2)\,. \tag{A.13}$$

The minimum of the least squares cost function (A.12) is found at roots of its gradient

$$0 \stackrel{!}{=} \nabla\mathcal{L} = (\partial_\mu\mathcal{L}, \partial_{\mu^*}\mathcal{L}, \partial_\psi\mathcal{L}, \partial_{\psi^*}\mathcal{L}) \tag{A.14}$$

As $\mathcal{L}$ is real-valued, the Wirtinger derivative with respect to a complex conjugate of one variable is the same as the complex conjugate of the derivative with respect to the same variable:

$$\left(\frac{\partial\mathcal{L}}{\partial\psi}\right)^* = \frac{\partial\mathcal{L}^*}{\partial\psi^*} = \frac{\partial\mathcal{L}}{\partial\psi^*} \tag{A.15}$$

This property of $\mathcal{L}$ reduces by a factor of two the number of derivatives which we have to calculate. The remaining derivatives may then be formed from complex conjugation. However, they do not yield additional independent constraints at a possible minimum and are thus disregarded here.

To derive the gradients, we first expand the squared modulus in (A.12), where we keep in mind that an exchange of arguments in the integrand of (A.12) is equivalent to complex conjugation, i.e. $\mu(\mathbf{x}_1-\mathbf{x}_2) = \mu^*(\mathbf{x}_2-\mathbf{x}_1)$ and $\Phi^*(\mathbf{x}_1,\mathbf{x}_2) = \Phi(\mathbf{x}_2,\mathbf{x}_1)$.

$$\mathcal{L}(\psi,\mu) = \iint \Phi(\mathbf{x}_1,\mathbf{x}_2)\Phi(\mathbf{x}_2,\mathbf{x}_1)\,\mathrm{d}\mathbf{x}_1\mathrm{d}\mathbf{x}_2 \tag{A.16}$$

$$- \iint \Phi(\mathbf{x}_1,\mathbf{x}_2)\mu(\mathbf{x}_2-\mathbf{x}_1)\psi^*(\mathbf{x}_2)\psi(\mathbf{x}_1)\,\mathrm{d}\mathbf{x}_1\mathrm{d}\mathbf{x}_2 \tag{A.17}$$

$$- \iint \Phi(\mathbf{x}_2,\mathbf{x}_1)\mu(\mathbf{x}_1-\mathbf{x}_2)\psi^*(\mathbf{x}_1)\psi(\mathbf{x}_2)\,\mathrm{d}\mathbf{x}_1\mathrm{d}\mathbf{x}_2 \tag{A.18}$$

$$+ \iint \left|\mu(\mathbf{x}_1-\mathbf{x}_2)\psi^*(\mathbf{x}_1)\psi(\mathbf{x}_2)\right|^2 \mathrm{d}\mathbf{x}_1\mathrm{d}\mathbf{x}_2 \tag{A.19}$$

The derivative with respect to μ is

$$\frac{\partial \mathcal{L}}{\partial \mu(\mathbf{x})} = -\iint \Phi(\mathbf{x}_1, \mathbf{x}_2)\delta(\mathbf{x} - \mathbf{x}_2 + \mathbf{x}_1)\psi^*(\mathbf{x}_2)\psi(\mathbf{x}_1)\,\mathrm{d}\mathbf{x}_1\mathrm{d}\mathbf{x}_2 \tag{A.20}$$

$$-\iint \Phi(\mathbf{x}_2, \mathbf{x}_1)\delta(\mathbf{x} - \mathbf{x}_1 + \mathbf{x}_2)\psi^*(\mathbf{x}_1)\psi(\mathbf{x}_2)\,\mathrm{d}\mathbf{x}_1\mathrm{d}\mathbf{x}_2 \tag{A.21}$$

$$+\iint \delta(\mathbf{x} - \mathbf{x}_1 + \mathbf{x}_2)\mu(\mathbf{x}_2 - \mathbf{x}_1)\left|\psi^*(\mathbf{x}_1)\psi(\mathbf{x}_2)\right|^2\mathrm{d}\mathbf{x}_1\mathrm{d}\mathbf{x}_2 \tag{A.22}$$

$$+\iint \delta(\mathbf{x} - \mathbf{x}_2 + \mathbf{x}_1)\mu(\mathbf{x}_1 - \mathbf{x}_2)]\left|\psi^*(\mathbf{x}_1)\psi(\mathbf{x}_2)\right|^2\mathrm{d}\mathbf{x}_1\mathrm{d}\mathbf{x}_2 \tag{A.23}$$

$$= -2\int \Phi(\mathbf{x}_1, \mathbf{x}_1 + \mathbf{x})\psi^*(\mathbf{x}_1 + \mathbf{x})\psi(\mathbf{x}_1)\,\mathrm{d}\mathbf{x}_1 + 2\mu(-\mathbf{x})\int \left|\psi^*(\mathbf{x}_1)\psi(\mathbf{x}_1 + \mathbf{x})\right|^2\mathrm{d}\mathbf{x}_1\ . \tag{A.24}$$

The derivative has one root at

$$\mu(-\mathbf{x}) = \mu^*(\mathbf{x}) = \frac{\int \Phi(\mathbf{x}_1, \mathbf{x}_1 + \mathbf{x})\psi^*(\mathbf{x}_1 + \mathbf{x})\psi(\mathbf{x}_1)\,\mathrm{d}\mathbf{x}_1}{\int |\psi^*(\mathbf{x}_1)\psi(\mathbf{x}_1 + \mathbf{x})|^2\,\mathrm{d}\mathbf{x}_1} \tag{A.25}$$

With the help of auxiliary functions $\chi_n(\mathbf{x}) = \phi_n^*(\mathbf{x})\,\psi(\mathbf{x})$, the expression (A.25) may be rewritten as a sum of autocorrelations,

$$\mu(\mathbf{x}) = \frac{\sum_n \chi_n(\mathbf{x}) \star \chi_n(\mathbf{x})}{\sum_n |\psi(\mathbf{x})|^2 \star |\psi(\mathbf{x})|^2} \tag{A.26}$$

Zeroing the other derivative with respect to ψ^*

$$\frac{\partial \mathcal{L}}{\partial \psi^*(\mathbf{x})} = -\iint \Phi(\mathbf{x}_1, \mathbf{x}_2)\mu(\mathbf{x}_2 - \mathbf{x}_1)\delta(\mathbf{x} - \mathbf{x}_2)\psi(\mathbf{x}_1)\,\mathrm{d}\mathbf{x}_1\mathrm{d}\mathbf{x}_2 \tag{A.27}$$

$$-\iint \Phi(\mathbf{x}_2, \mathbf{x}_1)\mu(\mathbf{x}_1 - \mathbf{x}_2)\delta(\mathbf{x} - \mathbf{x}_1)\psi(\mathbf{x}_2)\,\mathrm{d}\mathbf{x}_1\mathrm{d}\mathbf{x}_2 \tag{A.28}$$

$$+\iint \delta(\mathbf{x} - \mathbf{x}_1)\psi(\mathbf{x}_1)\left|\mu(\mathbf{x}_2 - \mathbf{x}_1)\psi(\mathbf{x}_2)\right|^2\mathrm{d}\mathbf{x}_1\mathrm{d}\mathbf{x}_2 \tag{A.29}$$

$$+\iint \delta(\mathbf{x} - \mathbf{x}_2)\psi(\mathbf{x}_2)\left|\mu(\mathbf{x}_2 - \mathbf{x}_1)\psi(\mathbf{x}_1)\right|^2\mathrm{d}\mathbf{x}_1\mathrm{d}\mathbf{x}_2 \tag{A.30}$$

$$= -2\int \Phi(\mathbf{x}_1, \mathbf{x})\mu(\mathbf{x} - \mathbf{x}_1)\psi(\mathbf{x}_1)\,\mathrm{d}\mathbf{x}_1 + 2\psi(\mathbf{x})\int \left|\mu(\mathbf{x} - \mathbf{x}_1)\psi(\mathbf{x}_1)\right|^2\mathrm{d}\mathbf{x}_1\ . \tag{A.31}$$

yields another condition for the minimum of $\mathcal{L}$:

$$\psi(\mathbf{x}) = \frac{\sum_n \phi_n(\mathbf{x})\,[\chi_n(\mathbf{x}) \otimes \mu(\mathbf{x})]}{\sum_n |\psi(\mathbf{x})|^2 \otimes |\mu(\mathbf{x})|^2}\ . \tag{A.32}$$

The last equation (A.32) may be interpreted as a projection and solved iteratively

$$\psi^{(\mathrm{i}+1)} = \frac{\sum_n \phi_n \left[(\phi_n^* \, \psi^{(\mathrm{i})}) \otimes \mu \right]}{\sum_n |\psi^{(\mathrm{i})}|^2 \otimes |\mu|^2 + \varepsilon_\psi} \,. \tag{A.33}$$

where ε_ψ is a small constant to avoid division by zero. Every few iterations when the projection stagnates, the kernel μ needs to be updated according to

$$\mu = \frac{\sum_n (\phi_n^* \, \psi^{(\mathrm{i})}) \star (\phi_n^* \, \psi^{(\mathrm{i})})}{\sum_n |\psi^{(\mathrm{i})}|^2 \star |\psi^{(\mathrm{i})}|^2 + \varepsilon_\mu} \,. \tag{A.34}$$

A.3. USAF-1951 resolution target

Group	0	1	2	3	4	5	6	7
Element								
1	1.00	2.00	4.00	8.00	16.00	32.00	64.00	128.00
2	1.12	2.24	4.49	8.98	17.96	35.92	71.84	143.70
3	1.26	2.52	5.04	10.08	20.16	40.32	80.63	161.30
4	1.41	2.83	5.66	11.31	22.63	45.25	90.51	181.00
5	1.59	3.17	6.35	12.70	25.40	50.80	101.60	203.20
6	1.78	3.56	7.13	14.25	28.51	57.02	114.00	228.10

Table A.1. – USAF-1951 resolution chart with the resolution in line pairs/mm

A.4. Presentations and Publications

Peer-reviewed publications as first author:

- B. Enders, M. Dierolf, P. Cloetens, M. Stockmar, F. Pfeiffer, and P. Thibault. *Ptychography with broad-bandwidth radiation.* Applied Physics Letters, **104**(17), 2014. doi:10.1063/1.4874304

- B. Enders, M. Dierolf, F. Pfeiffer, and P. Thibault. *Ptypy – A computational framework for ptychography.* in preparation for submission to Proceedings of the Royal Society A

- B. Enders, K. Giewekemeyer, T. Kurz, S. Podorov, and T. Salditt. *Non-iterative coherent diffractive imaging using a phase-shifting reference frame.* New Journal of Physics, 11, 2009. ISSN 13672630. doi:10.1088/1367-2630/11/4/043021.

Peer-reviewed publications as co-author

- I. Zanette, B. Enders, M. Dierolf, P. Thibault, R. Gradl, A. Diaz, M. Guizar-Sicairos, A. Menzel, F. Pfeiffer, and P. Zaslansky. *Ptychographic X-ray nanotomography quantifies mineral distributions in human dentine.* Scientific reports, 5:9210, 2015. ISSN 2045-2322. doi:10.1038/srep09210.

- M. Stockmar, P. Cloetens, I. Zanette, B. Enders, M. Dierolf, F. Pfeiffer, and P. Thibault. *Near-field ptychography: phase retrieval for inline holography using a structured illumination.* Scientific reports, 3:1927, 2013. ISSN 2045-2322. doi:10.1038/srep01927.

- M. Stockmar, M. Hubert, M. Dierolf, B. Enders, R. Clare, S. Allner, A. Fehringer, I. Zanette, J. Villanova, J. Laurencin, P. Cloetens, F. Pfeiffer, and P. Thibault. *X-ray nanotomography using near-field ptychography.* Optics Express, 23(10):12720, 2015a. ISSN 1094-4087. doi:10.1364/OE.23.012720

- M. Stockmar, I. Zanette, M. Dierolf, B. Enders, R. Clare, F. Pfeiffer, P. Cloetens, A. Bonnin, and P. Thibault. *X-Ray Near-Field Ptychography for Optically Thick Specimens.* Physical Review Applied, 3(1):1–6, 2015b. ISSN 2331-7019. doi:10.1103/PhysRevApplied.3.014005

Non peer-reviewed publications

- B. Enders, *Single-Step Inversion for Coherent Diffraction Imaging – A Generalization to Fourier Transform Holography.* Diplomarbeit (thesis), Georg-August-Universität zu Göttingen, 2009

- ESRF Highlights 2014, *"Robust lensless imaging with a high-flux instrument"*, http://www.esrf.eu/home/UsersAndScience/Publications/Highlights/highlights-2014/XRI/XRI03.html

Presentations

- XRM 2014, October 2014, Melbourne, Australia (Talk+poster)

- XRM 2014 Big Data Satellite Workshop, October 2014, Melbourne, Australia (Talk)

- Coherence 2014, September 2014, Chicago, IL, USA (Poster)

- EUROMAT 2013, September 2013, Sevilla, Spain (Talk)

- Ptycho 2013, May 2013, Schloss Hohenkammern, Germany (Invited Talk)

- XRM 2012, August 2012, Shanghai, China (Poster)

- Coherence Conference, June 2012, Fukuoka, Japan (Talk)

- DPG Spring Meeting, March 2012, Berlin, Germany (Talk)

- SSOM 3d Symposium, March 2012, Les Diablerets, Switzerland (Poster)

- ESPCA-FAPESP Advanced School, January 2011, Campinas, Brazil (Poster)

B. Ptypy documentation

In the following sections, an excerpt of the online documentation about Ptypy is appended. It is also accessible online at https://ptycho.github.io/ptypy.

B.1. Tutorials and installation

CHAPTER

ONE

PTYPY DOCUMENTATION CONTENTS

1.1 Getting started with Ptypy

1.1.1 Installation

General installation instructions

Being a python package, PtyPy depends on a number of other packages which are listed below. Once its dependecies are met, ptypy should work *out-of-the-box*.

Note: PtyPy is developed for Python 2.7.x and is currently incompatible with Python 3.x. Python 3 support is planned in future releases.

Essential packages

- NumPy (homepage: http://www.numpy.org)
- SciPy (homepage: http://www.scipy.org)
- h5py (homepage: http://www.h5py.org)

Recommended packages for additional functionality

Please note that PtyPy is an alpha release and lacks rigorous import checking. There may be parts of code that implicitly ask for any of the packages listed here. Therefore, we recommend to install these packages.

- Matplotlib for any kind of plotting or image generation (homepage: http://www.matplotlib.org) and python bindings for QT4
- mpi4py for CPU-node parallelisation, contains python bindings for the Message Passaging Interface, (homepage: https://bitbucket.org/mpi4py/mpi4py/)
- pyzmq for a threaded server on the main node to perfrom asynchronous client-server communication, contains python bindings for the ZeroMQ protocol, (homepage: http://github.com/zeromq/pyzmq). This package is needed for non-blocking plots of the reconstruction run (e.g. for *Run a plotclient in a separate process*).

Optional packages

Other very useful packages are

- Ipython (homepage: http://www.ipython.org)

Installation on Debian/Ubuntu

PtyPy is developed on the current LTS version of Ubuntu which is the recommended operating system for PtyPy

Prerequisites

Many of the required python packages are available in the repositories. Just type (with sudo rights)

```
$ sudo apt-get install python-numpy python-scipy python-h5py\
$ python-matplotlib python-mpi4py python-pyzmq
```

Get ptypy

Download and unpack PtyPy from the sources:

```
$ wget https://github.com/ptycho/ptypy/archive/master.zip
$ unzip master.zip -d /tmp/; rm master.zip
```

or make a clone of the PtyPy github repository:

```
$ git clone https://github.com/ptycho/ptypy.git /tmp/ptypy-master
```

CD into the download directory (e.g. /tmp/ptypy-master) and install PtyPy with

```
$ python setup.py install --user
```

The local `--user` install is recommended instead of a system-wide `sudo` install. It makes it easier to quickly apply fixes yourself. However, for an all-user system-wide install, type

```
$ sudo python setup.py install
```

Installation on Windows

These installation instructions were contributed by M. Stockmar and tested with *Windows 8.1 Enterprise (64 bit)* on a core i7 Thinkpad w520 in February 2015. No python was installed before on that machine.

Note: There might be also simpler ways to get a full scientific python suite for 32 bit Windows. Please let us know if you managed to get PtyPy running on a system not listed here.

Prerequisites

- Download and install python 2.7.x from http://www.python.org (Make sure you have the 64 bit version) Click *yes* when asked to add python to the path environment variable.
- Go to the command line and install wheel:

```
$ pip install wheel
```

- Install Microsoft's implementation for MPI to use multiple CPUs from https://msdn.microsoft.com/en-us/library/bb524831(v=vs.85).aspx Other MPI implementations may work as well but were not tested.
- Install the QT-Framework if you don't have it already. http://qt-project.org/

Download all other binaries

The binaries will be downloaded from a website which offers various builds for different python versions for Windows (both 32 and 64 bit) in the form of a python wheel. Make sure you choose the correct version for your windows and system.

Numpy and some other builds are linked against the Intel Math Kernel Library (MKL) which is supposed to be fast.

- First make sure you have the Microsoft Visual C++ 2010 redistributable package (maybe also the 2008 version).
  - 2010 (64 bit, for 32 bit version check the website):
    http://www.microsoft.com/en-us/download/details.aspx?id=14632
  - 2008 (64 bit, for 32 bit check the website):
    http://www.microsoft.com/en-us/download/details.aspx?id=15336
- Now, go to to http://www.lfd.uci.edu/~gohlke/pythonlibs/ and download the latest pip binaries and update pip:

```
$ python.exe pip-6.0.8-py2.py3-none-any.whl/pip install pip-6.0.8-py2.py3-none-any.whl.
```

  (The file name might change slightly if a newer version of pip is available)

- Then download all the other binaries as whl-files. A whl-file can be installed via command line according to:

```
$ pip install filename.whl
```

  Download and install the binaries in the following order:

  1. `numpy`
  2. `pillow` (replacement for PIL)
  3. `matplotlib`
  4. `ipython`
  5. `h5py`
  6. `scipy` (may install additional packages as well)
  7. `mpi4py` (choose the one for MS MPI if you have installed MS MPI)
  8. `pyzmq`
  9. `pyqt4` (QT framework was installed before on the testing machine)
  10. `pyside`

Get ptypy

Download PtyPy from https://github.com/ptycho/ptypy/archive/master.zip and unzip to any directory, for example `C:\Temp`. Change into that directly and install from commandline:

```
$ cd C:\Temp\ptypy-master
$ python setup.py install
```

1.1.2 Quickstart with a minimal script

Note: This tutorial was generated from the python source `[ptypy_root]/tutorial/minimal_script.py` using `ptypy/doc/script2rst.py`. You are encouraged to modify the parameters and rerun the tutorial with:

```
$ python [ptypy_root]/tutorial/minimal_script.py
```

This tutorial explains the minimal settings to get a reconstruction runnig in PtyPy. A PtyPy script consists of two parts:

- Creation of a parameter tree with parameters as listed in *Parameter tree structure* and
- calling a `Ptycho` instance with this parameter tree, specifying a level that determines how much the Ptycho instance will do.

Preparing the parameter tree

We begin with opening an empty python file of arbitrary name in an editor of your choice, e.g.:

```
$ gedit minimal_script.py
```

Next we create an empty parameter tree. In PtyPy, parameters are managed by the `Param` class which is a convenience class subclassing Python's dict type. It is designed in such a way that dictionary items can be accessed also as class attributes, making the scripts and code much more readable.

```
>>> from ptypy import utils as u
>>> p = u.Param()  # root level
```

We set the verbosity to a high level, in order to have information on the reconstruction process printed to the terminal. See `verbose_level`.

```
>>> p.verbose_level = 3
```

We limit this reconstruction to single precision. The other choice is to use double precision.

```
>>> p.data_type = "single"
```

We give this reconstruction the name `'minimal'` although it will automatically choose one from the file name of the script if we put in `None`. (But then the tutorial may not work on your computer as the chosen run name may differ from the one that this tutorial was created with)

```
>>> p.run = 'minimal'
```

Next, we set the home path. The `Ptycho` instance will use this path as base for any other relative file path (e.g `io.autosave.path` or `io.rfile`). Relative paths lack a leading "/" (or "C:\" on windows). Make sure to replace all "/" with "\" if you run the scripts on a Windows system.

```
>>> p.io = u.Param()
>>> p.io.home = "/tmp/ptypy/"
```

We want an intermediate result of the reconstruction to be dumped regularly every 20 reconstructions.

```
>>> p.io.autosave = u.Param()
>>> p.io.autosave.interval = 20
```

In this tutorial we switch off the threaded plotting client.

```
>>> p.io.autoplot = False
```

Since we do not want to plot anything, we don't need the interaction server either.

```
>>> p.io.interaction = False
```

Now we have to insert actual parameters associated with a ptychographic scan.

PtyPy is designed to support reconstruction from mutliple scans. The `scan` branch of the tree holds all *common* parameters for scans and acts as a default template when there are more than one scan. Scan-specific parameters have to be placed in another branch called `scans`. If there is only one scan, parameters can be given in either

branch. In this tutorial we do not bother to enter parameters here so we leave the branch empty (It will be filled with the defaults of *scan*) .

```
>>> p.scan = u.Param()
```

Each individual scan is represented by a branch in `scans`. The parameters in these branches are those that differ from the *defaults* in the `scan` branch mentioned above. Obviously at least the `data` branch will differ from *scan.data*. In this tutorial we create a new scan parameter branch `MF` where we only specify the data branch and tell PtyPy to use scan meta data when possible.

```
>>> p.scans = u.Param()
>>> p.scans.MF = u.Param()
>>> p.scans.MF.if_conflict_use_meta = True
>>> p.scans.MF.data = u.Param()
```

As data source we have choosen the *'test'* source. That will make PtyPy use the internal *MoonFlowerScan* class. This class is meant for testing, and it provides/simulates diffraction patterns without using the more complex generic *SimScan* class.

```
>>> p.scans.MF.data.source = 'test'
```

We set the diffraction frame shape to a small value (128x128px) and limit the number af diffraction patterns at 100. The *MoonFlowerScan* instance will balance the diffraction patterns accordingly.

```
>>> p.scans.MF.data.shape = 128
>>> p.scans.MF.data.num_frames = 100
```

We skip saving the "prepared" data file for now. The PtyPy data management is described in detail in *Data management*

```
>>> p.scans.MF.data.save = None
```

Needlees to say, we need to specify a reconstruction engine. We choose 40 iterations of difference map algorithm.

```
>>> p.engines = u.Param()
>>> p.engines.engine00 = u.Param()
>>> p.engines.engine00.name = 'DM'
>>> p.engines.engine00.numiter = 40
>>> p.engines.engine00.numiter_contiguous = 5
```

Running ptypy

We import the *Ptycho* class and pass the tree `p` at level 5. This level tells *Ptycho* to initialize everything and start the reconstruction using all reconstruction engines in `p.engines` immediately upon construction.

```
>>> from ptypy.core import Ptycho
>>> P = Ptycho(p, level=5)
Verbosity set to 3
Data type:               single

---- Ptycho init level 1 ------------------------------------------------------
Model: sharing probe between scans (one new probe every 1 scan)
Model: sharing probe between scans (one new probe every 1 scan)

---- Ptycho init level 2 ------------------------------------------------------
Prepared 92 positions
Processing new data.
---- Enter PtyScan.initialixe() -----------------------------------------------
             Common weight : True
                    shape = (128, 128)
All experimental positions : True
                    shape = (92, 2)
Scanning positions (92) are fewer than the desired number of scan points (100).
```

```
Resetting `num_frames` to lower value
---- Leaving PtyScan.initialixe() ------------------------------------------
ROI center is [ 64.  64.], automatic guess is [ 63.44565217  63.55434783].
Feeding data chunk
Importing data from MF as scan MF.
End of scan reached
End of scan reached

--- Scan MF photon report ---
Total photons   : 3.66e+09
Average photons : 3.98e+07
Maximum photons : 7.13e+07
-----------------------------

---- Creating PODS ----------------------------------------------------------
Found these probes :
Found these objects:
Process 0 created 92 new PODs, 1 new probes and 1 new objects.

---- Probe initialization ---------------------------------------------------
Initializing probe storage S00G00 using scan MF
Found no photon count for probe in parameters.
Using photon count 7.13e+07 from photon report

---- Object initialization --------------------------------------------------
Initializing object storage S00G00 using scan MF
Simulation resource is object transmission

---- Creating exit waves ----------------------------------------------------

Process #0 ---- Total Pods 92 (92 active) ----
-------------------------------------------------------------------------------
(C)ontnr : Memory : Shape            : Pixel size      : Dimensions      : Views
(S)torgs : (MB)   : (Pixel)          : (meters)        : (meters)        : act.
-------------------------------------------------------------------------------
Cprobe   :    0.1 : complex64
S00G00   :    0.1 :        1*128*128 :   6.36*6.36e-08 :   8.14*8.14e-06 :    92
Cmask    :    1.5 :   bool
S0000    :    1.5 :       92*128*128 :   1.72*1.72e-04 :   2.20*2.20e-02 :    92
Cexit    :   12.1 : complex64
S0000G00 :   12.1 :       92*128*128 :   6.36*6.36e-08 :   8.14*8.14e-06 :    92
Cobj     :    1.3 : complex64
S00G00   :    1.3 :        1*394*408 :   6.36*6.36e-08 :   2.51*2.60e-05 :    92
Cdiff    :    6.0 : float32
S0000    :    6.0 :       92*128*128 :   1.72*1.72e-04 :   2.20*2.20e-02 :    92



---- Ptycho init level 3 ----------------------------------------------------

---- Ptycho init level 4 ----------------------------------------------------

==== Starting DM-algoritm. ===================================================

Parameter set:
* id3V0S9PUCL0           : ptypy.utils.parameters.Param(16)
  * clip_object          : None
  * fourier_relax_factor : 0.05
  * numiter_contiguous   : 5
  * overlap_converge_... : 0.1
  * probe_update_start   : 0
  * probe_inertia        : 0.001
  * name                 : DM
```

```
  * subpix_start         : 0
  * update_object_first  : True
  * obj_smooth_std       : None
  * alpha                : 1
  * overlap_max_itera... : 10
  * object_inertia       : 0.1
  * numiter              : 40
  * probe_support        : 0.8
  * subpix               : linear
================================================================================
---------------------------------- Autosaving ----------------------------------
WARNING root - Save file exists but will be overwritten (force_overwrite is True)
Generating copies of probe, object and parameters and runtime
Saving to /tmp/ptypy/dumps/minimal/minimal_None_0000.ptyr
--------------------------------------------------------------------------------
Time spent in Fourier update: 1.14
Time spent in Overlap update: 0.35
Iteration #5 of DM :: Time 1.49
Errors :: Fourier 1.21e+03, Photons 1.84e+03, Exit 8.43e+02
Time spent in Fourier update: 1.17
Time spent in Overlap update: 0.21
Iteration #10 of DM :: Time 1.38
Errors :: Fourier 9.49e+02, Photons 8.09e+02, Exit 6.06e+02
Time spent in Fourier update: 1.16
Time spent in Overlap update: 0.21
Iteration #15 of DM :: Time 1.37
Errors :: Fourier 8.88e+02, Photons 5.32e+02, Exit 4.18e+02
Time spent in Fourier update: 1.15
Time spent in Overlap update: 0.22
Iteration #20 of DM :: Time 1.37
Errors :: Fourier 7.29e+02, Photons 2.91e+02, Exit 2.48e+02
---------------------------------- Autosaving ----------------------------------
WARNING root - Save file exists but will be overwritten (force_overwrite is True)
Generating copies of probe, object and parameters and runtime
Saving to /tmp/ptypy/dumps/minimal/minimal_DM_0020.ptyr
--------------------------------------------------------------------------------
Time spent in Fourier update: 1.16
Time spent in Overlap update: 0.21
Iteration #25 of DM :: Time 1.37
Errors :: Fourier 5.26e+02, Photons 1.44e+02, Exit 1.23e+02
Time spent in Fourier update: 1.14
Time spent in Overlap update: 0.21
Iteration #30 of DM :: Time 1.35
Errors :: Fourier 3.84e+02, Photons 6.54e+01, Exit 5.93e+01
Time spent in Fourier update: 1.15
Time spent in Overlap update: 0.21
Iteration #35 of DM :: Time 1.35
Errors :: Fourier 2.69e+02, Photons 3.51e+01, Exit 3.05e+01
Time spent in Fourier update: 1.14
Time spent in Overlap update: 0.22
Iteration #40 of DM :: Time 1.36
Errors :: Fourier 1.61e+02, Photons 2.11e+01, Exit 1.53e+01
WARNING root - Save file exists but will be overwritten (force_overwrite is True)
Generating shallow copies of probe, object and parameters and runtime
Saving to /tmp/ptypy/recons/minimal/minimal_DM.ptyr
```

From the terminal log, we note that there was an autosave every 20 iterations and the error reduced from iteration to iteration.

1.1.3 Utilies/Binaries for convenience

PtyPy provides a few utility scripts to make life easier for you, the user. They are located in `[ptypy_root]/scripts`. In case of a user install on Ubuntu Linux, they are copied to `~/.local/bin`

Note: Due to the early stage of developmnet, these scripts may see substantial changes in further releases, i.e. the call signature may change.

Plotting from a reconstruction/dump file (*.ptyr)

`ptypy.plot` is an automatic plotting script that installs on Unix systems It has the syntax

```
$ ptypy.plot [-h] [-l LAYOUT] [-t IMFILE] ptyrfile
```

For our minimal example this translates to

```
$ ptypy.plot /tmp/ptypy/recons/minimal/minimal_DM.ptyr -t minimal.png
```

and the image looks like this (Fig. 1.1)

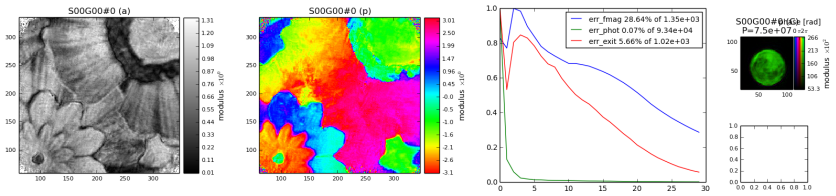


Fig. 1.1: Example plot made with `ptypy.plot` using the *default* layput

Inspecting a hdf5 compatible file

Sometimes we want to quickly inspect what is in a *hdf5* file that was created by PtyPy. For such cases, we can use `ptypy.inspect`.

```
$ ptypy.inspect [-h] [-p PATH] [--report] [-d DEPTH] h5file
```

For example, a quick view at the top level can be realized with

```
$ ptypy.inspect /tmp/ptypy/recons/minimal/minimal_DM.ptyr -d 1
```

which has the following the output:

```
* content [Param 4]:
    * obj [dict 1]:
    * pars [Param 9]:
    * probe [dict 1]:
    * runtime [Param 8]:
* header [dict 2]:
    * description [string = "Ptypy .h5 compatible storage format"]
    * kind [string = "minimal"]
```

If we are interested solely in the probe we could use

```
$ ptypy.inspect /tmp/ptypy/recons/minimal/minimal_DM.ptyr -d 1 -p content/probe
```

which has the following the output:

```
* S00G00 [dict 13]:
    * DataTooSmall [scalar = False]
    * ID [string = "S00G00"]
    * _center [array = [64 64]]
    * _origin [array = [ -4.07172778e-06  -4.07172778e-06]]
    * _pool [Param 0]:
    * _psize [array = [  6.36207466e-08   6.36207466e-08]]
    * data [1x128x128 complex64 array]
    * fill_value [scalar = 0.0]
    * layermap [list = [0]]
    * model_initialized [scalar = True]
    * nlayers [scalar = 1]
    * padonly [scalar = False]
    * shape [tuple = (1, 128, 128)]
```

We omitted the result for the complete file to save some space but you are encouraged to try:

```
$ ptypy.inspect /tmp/ptypy/recons/minimal/minimal_DM.ptyr
```

Create a new template for a reconstruction script

In cases where we want to create a new reconstruction script from scratch, it may be cumbersome to write each and every parameter that we want to change. But, with the help of `ptypy.new`, we can create a python script which is prefilled with defaults:

```
$ ptypy.new [-h] [-u ULEVEL] [--short-doc] [--long-doc] pyfile
```

In the folder `[ptypy_root]/templates` you find two scripts that where auto-generated with the following calls:

```
$ ptypy.new -u 0 --long-doc pars_few_alldoc.py
$ ptypy.new -u 2  pars_all_nodoc.py
```

And here is a quick view of the first one with much documentation (only the first 50 lines).

Run a plotclient in a separate process

PtyPy supports a Client/Server approach. That means that the reconstruction process runs on a remote server (cluster) while we can monitor the progress on a local machine.

In this case, we need to start a plotting Client on a separate machine). You can implement your own plotting client but you may find it convenient to use the plotting utility `ptypy.plotclient`:

```
$ ptypy.plotclient [-h] [-l LAYOUT] [--dump] [--movie] [-i INTERVAL] [-d DIRECTORY]
```

Note: None of the options are currently implemented. The plotting client receives all information from the server it connects to. Work in progress ..

1.1.4 More script templates

Besides the script from which section *Quickstart with a minimal script* was generated, there is a trinity of similar scripts at your disposal that you can temper with.

All-in-one

We encourage you to use the script `[ptypy_root]/templates/minimal_prep_and_run.py` and modify the *recipe* part of the data parameter branch. Observe what changes in the reconstruction when scan parameters change.

```
"""
This script is a test for ptychographic reconstruction in the absence
of actual data. It uses the test Scan class
`ptypy.core.data.MoonFlowerScan` to provide "data".
"""
import ptypy
from ptypy.core import Ptycho
from ptypy import utils as u
p = u.Param()

# for verbose output
p.verbose_level = 3

# set home path
p.io = u.Param()
p.io.home = "/tmp/ptypy/"
p.autosave = None

# max 100 frames (128x128px) of diffraction data
p.scans = u.Param()
p.scans.MF = u.Param()
p.scans.MF.data= u.Param()
p.scans.MF.data.source = 'test'
p.scans.MF.data.shape = 128
p.scans.MF.data.num_frames = 100
p.scans.MF.data.save = None

## special recipe paramters for this scan ##
p.scans.MF.data.recipe = u.Param()
# position distance in fraction of illumination frame
p.scans.MF.data.recipe.density = 0.2
# total number of photon in empty beam
p.scans.MF.data.recipe.photons = 1e8
# Gaussian FWHM of possible detector blurring
p.scans.MF.data.recipe.psf = 0.

# attach a reconstrucion engine
p.engines = u.Param()
p.engines.engine00 = u.Param()
p.engines.engine00.name = 'DM'
p.engines.engine00.numiter = 30

# prepare and run
P = Ptycho(p,level=5)
```

Creating a .ptyd data-file

We encourage you to use this script `[ptypy_root]/templates/make_sample_ptyd.py` to create various different samples and see what happens if the data processing parameters are changed. If you have become curious, move forward to *Data management* and take a look at PtyPy's data management. Check out the data parameter branch `scan.data` for detailed parameter descriptions.

```
"""
This script creates a sample *.ptyd data file using the built-in
test Scan `ptypy.core.data.MoonFlowerScan`
"""
import sys
import time
import ptypy
from ptypy import utils as u

```

```
# for verbose output
u.verbose.set_level(3)

# create data parameter branch
data = u.Param()
data.dfile = 'sample.ptyd'
data.shape = 128
data.num_frames = 100
data.save = 'append'
data.label=None
data.psize=None
data.energy=None
data.center=None
data.distance = None
data.auto_center = None
data.rebin = None
data.orientation = None

# create PtyScan instance
MF = ptypy.core.data.MoonFlowerScan(data)
MF.initialize()
for i in range(2):
    # autoprocess data
    msg = MF.auto(100)
    time.sleep(2)
    # logs the out put of .auto() to terminal prompt
    u.verbose.logger.info(u.verbose.report(msg), extra={'allprocesses': True})
```

Loading a data file to run a reconstruction

The script `[ptypy_root]/templates/minimal_load_and_run.py` should resembles the case of data analysis after the experiment has taken place. Take a challenging sample data from before and alter the reconstruction parameters and algorithms to find out if you can make the recontruction converge. Check out the engine parameter branch `engine` for detailed parameter descriptions.

```
"""
This script is a test for ptychographic reconstruction after an
experiment has been carried out and the data is available in ptypy's
data file format in "/tmp/ptypy/sample.ptyd"
"""
import ptypy
from ptypy.core import Ptycho
from ptypy import utils as u

p = u.Param()
p.verbose_level = 3
p.io = u.Param()
p.io.home = "/tmp/ptypy/"
p.autosave = None

p.scans = u.Param()
p.scans.MF = u.Param()
p.scans.MF.data= u.Param()
p.scans.MF.data.source = 'sample.ptyd'#'file'
p.scans.MF.data.dfile = None#'sample.ptyd'

p.engine = u.Param()

## Common defaults for all engines
p.engine.common = u.Param()
# Total number of iterations
```

```
p.engine.common.numiter = 100
# Number of iterations to be executed in one go
p.engine.common.numiter_contiguous = 1
# Fraction of valid probe area (circular) in probe frame
p.engine.common.probe_support = None
# Number of iterations before probe update starts
p.engine.common.probe_update_start = 2
# Clip object amplitude into this intrervall
p.engine.common.clip_object = None   # [0,1]

## DM default parameters
p.engine.DM = u.Param()
p.engine.DM.name = "DM"
# HIO parameter
p.engine.DM.alpha = 1
# Probe fraction kept from iteration to iteration
p.engine.DM.probe_inertia = 0.01
# Object fraction kept from iteration to iteration
p.engine.DM.object_inertia = 0.1
# If False: update object before probe
p.engine.DM.update_object_first = True
# Gaussian smoothing (FWHM, pixel units) of object
p.engine.DM.obj_smooth_std = 10
# Loop the overlap constraint until probe changes lesser than this fraction
p.engine.DM.overlap_converge_factor = 0.5
# Maximum iterations to be spent inoverlap constraint
p.engine.DM.overlap_max_iterations = 100
# If rms of model vs diffraction data is smaller than this fraction,
# Fourier constraint is considered fullfilled
p.engine.DM.fourier_relax_factor = 0.05

p.engines = u.Param()
p.engines.engine00 = u.Param()
p.engines.engine00.name = 'DM'
p.engines.engine00.numiter = 30
p.engines.engine01 = u.Param()
p.engines.engine01.name = 'ML'
p.engines.engine01.numiter = 20

P = Ptycho(p,level=5)
```

1.2 Concepts and Classes

1.2.1 Tutorial: Data access - Storage, View, Container

Note: This tutorial was generated from the python source `[ptypy_root]/tutorial/ptypyclasses.py` using `ptypy/doc/script2rst.py`. You are encouraged to modify the parameters and rerun the tutorial with:

```
$ python [ptypy_root]/tutorial/ptypyclasses.py
```

This tutorial explains how PtyPy accesses data / memory. The data access is governed by three classes: *Container*, *Storage*, and *View*

First a short reminder from the *classes* Module about the classes this tutorial covers.

Container class

```
A high-level container that keeps track of sub-containers (Storage)
and Views onto them. A container can copy itself to produce a buffer
needed for calculations. Some basic Mathematical operations are
```

```
    implemented at this level as in-place operations.
    In general, operations on arrays should be done using the Views, which
    simply return numpy arrays.
```

Storage class

```
    The sub-container, wrapping a numpy array buffer. A Storage defines a
    system of coordinate (for now only a scaled translation of the pixel
    coordinates, but more complicated affine transformation could be
    implemented if needed). The sub-class DynamicStorage can adapt the size
    of its buffer (cropping and/or padding) depending on the Views.
```

View class

```
    A low-weight class that contains all information to access a 2D piece
    of a Storage within a Container. The basic idea is that the View
    access is controlled by a physical position and its frame, such that
    one is not bothered by memory/array addresses when accessing data.
```

Import some modules

```
>>> import matplotlib as mpl
>>> import numpy as np
>>> import ptypy
>>> from ptypy import utils as u
>>> from ptypy.core import View, Container, Storage, Base
>>> plt = mpl.pyplot
```

A single Storage in one Container

The governing object is a *Container* instance, which we need to construct as first object. It does not need much for an input but the data type.

```
>>> C1 = Container(data_type='real')
```

This class itself holds nothing at first. In order to store data in `C1` we have to add a *Storage* to that container.

```
>>> S1 = C1.new_storage(shape=(1, 7, 7))
```

Similarly we can contruct the Storage from a data buffer. `S1=C1.new_storage(data=np.ones((1,7,7)))` would have also worked.

All of PtyPy's special classes carry a uniue ID, which is needed to communicate these classes across nodes and for saving and loading.

As we haven't specified an ID the Container class picks one for `S1` In this case that will be `S0000` where the *S* refers to the class type.

```
>>> print S1.ID
S0000
```

Let's have a look at what kind of data Storage holds.

```
>>> print S1.formatted_report()[0]
(C)ontnr : Memory : Shape            : Pixel size      : Dimensions      : Views
(S)torgs : (MB)   : (Pixel)          : (meters)        : (meters)        : act.
--------------------------------------------------------------------------------
S0000    :    0.0 :            1*7*7 :   1.00*1.00e+00 :   7.00*7.00e+00 :     0
```

Apart from the ID on the left we discover a few other entries, for example the quantity `psize` which refers to the physical pixel size for the last two dimensions of the stored data.

```
>>> print S1.psize
[ 1.  1.]
```

Many attributes of a *Storage* are in fact *properties*. Changing their value may have an impact on other methods or attributes of the class. For example, one convenient method is Storage.*grids()* that creates coordinate grids for the last two dimensions (see also *ptypy.utils.array_utils.grids()*)

```
>>> y, x = S1.grids()
>>> print y
[[[-3. -3. -3. -3. -3. -3. -3.]
  [-2. -2. -2. -2. -2. -2. -2.]
  [-1. -1. -1. -1. -1. -1. -1.]
  [ 0.  0.  0.  0.  0.  0.  0.]
  [ 1.  1.  1.  1.  1.  1.  1.]
  [ 2.  2.  2.  2.  2.  2.  2.]
  [ 3.  3.  3.  3.  3.  3.  3.]]]

>>> print x
[[[-3. -2. -1.  0.  1.  2.  3.]
  [-3. -2. -1.  0.  1.  2.  3.]
  [-3. -2. -1.  0.  1.  2.  3.]
  [-3. -2. -1.  0.  1.  2.  3.]
  [-3. -2. -1.  0.  1.  2.  3.]
  [-3. -2. -1.  0.  1.  2.  3.]
  [-3. -2. -1.  0.  1.  2.  3.]]]
```

which are coordinate grids for vertical and horizontal axes respectively. We note that these coordinates have their center at

```
>>> print S1.center
[3 3]
```

Now we change a few properties. For example,

```
>>> S1.center = (2, 2)
>>> S1.psize = 0.1
>>> g = S1.grids()
>>> print g[0]
[[[-0.2 -0.2 -0.2 -0.2 -0.2 -0.2 -0.2]
  [-0.1 -0.1 -0.1 -0.1 -0.1 -0.1 -0.1]
  [ 0.   0.   0.   0.   0.   0.   0. ]
  [ 0.1  0.1  0.1  0.1  0.1  0.1  0.1]
  [ 0.2  0.2  0.2  0.2  0.2  0.2  0.2]
  [ 0.3  0.3  0.3  0.3  0.3  0.3  0.3]
  [ 0.4  0.4  0.4  0.4  0.4  0.4  0.4]]]

>>> print g[1]
[[[-0.2 -0.1  0.   0.1  0.2  0.3  0.4]
  [-0.2 -0.1  0.   0.1  0.2  0.3  0.4]
  [-0.2 -0.1  0.   0.1  0.2  0.3  0.4]
  [-0.2 -0.1  0.   0.1  0.2  0.3  0.4]
  [-0.2 -0.1  0.   0.1  0.2  0.3  0.4]
  [-0.2 -0.1  0.   0.1  0.2  0.3  0.4]
  [-0.2 -0.1  0.   0.1  0.2  0.3  0.4]]]
```

We observe that the center has in fact moved one pixel up and one left. The *center()* property uses pixel units. If we want to use a physical quantity to shift the center, we may instead set the top left pixel to a new value, which shifts the center to a new position.

```
>>> S1.origin -= 0.12
>>> y, x = S1.grids()
>>> print y
[[[-0.32 -0.32 -0.32 -0.32 -0.32 -0.32 -0.32]
  [-0.22 -0.22 -0.22 -0.22 -0.22 -0.22 -0.22]
  [-0.12 -0.12 -0.12 -0.12 -0.12 -0.12 -0.12]
  [-0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02]
  [ 0.08  0.08  0.08  0.08  0.08  0.08  0.08]
```

```
   [ 0.18  0.18  0.18  0.18  0.18  0.18  0.18]
   [ 0.28  0.28  0.28  0.28  0.28  0.28  0.28]]]

>>> print x
[[[-0.32 -0.22 -0.12 -0.02  0.08  0.18  0.28]
  [-0.32 -0.22 -0.12 -0.02  0.08  0.18  0.28]
  [-0.32 -0.22 -0.12 -0.02  0.08  0.18  0.28]
  [-0.32 -0.22 -0.12 -0.02  0.08  0.18  0.28]
  [-0.32 -0.22 -0.12 -0.02  0.08  0.18  0.28]
  [-0.32 -0.22 -0.12 -0.02  0.08  0.18  0.28]
  [-0.32 -0.22 -0.12 -0.02  0.08  0.18  0.28]]]

>>> print S1.center
[ 3.2  3.2]
```

Up until now our actual *data* numpy array located at `S1.data` is still filled with ones, i.e. flat. We can use *`Storage.fill`* to fill that container with an array.

```
>>> S1.fill(x+y)
>>> print S1.data
[[[-0.64 -0.54 -0.44 -0.34 -0.24 -0.14 -0.04]
  [-0.54 -0.44 -0.34 -0.24 -0.14 -0.04  0.06]
  [-0.44 -0.34 -0.24 -0.14 -0.04  0.06  0.16]
  [-0.34 -0.24 -0.14 -0.04  0.06  0.16  0.26]
  [-0.24 -0.14 -0.04  0.06  0.16  0.26  0.36]
  [-0.14 -0.04  0.06  0.16  0.26  0.36  0.46]
  [-0.04  0.06  0.16  0.26  0.36  0.46  0.56]]]
```

We can have visual check on the data using *`plot_storage()`*

```
>>> fig = u.plot_storage(S1, 0)
```

See Fig. 1.2 for the plotted image.

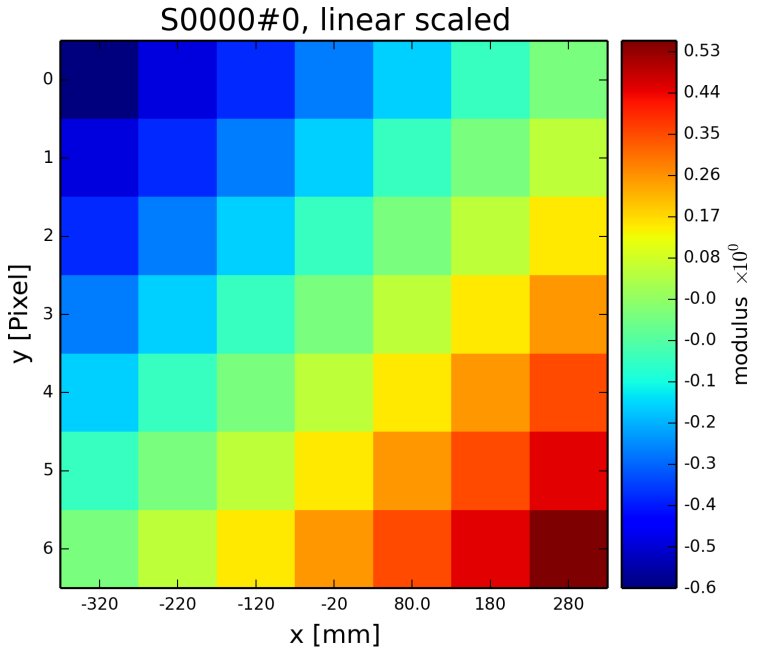


Fig. 1.2: A plot of the data stored in `S1`

Adding Views as a way to access data

Besides being able to access the data directly through its attribute and the corresponding *numpy* syntax, ptypy offers acces through a *View* instance. The View invocation is a bit more complex.

```
>>> from ptypy.core.classes import DEFAULT_ACCESSRULE
>>> ar = DEFAULT_ACCESSRULE.copy()
>>> print ar
* id3V02DED3HG           : ptypy.utils.parameters.Param(6)
  * layer                : 0
  * psize                : 1.0
  * shape                : None
  * coord                : None
  * active               : True
  * storageID            : None
```

Let's say we want a 4x4 view on Storage S1 around the origin. We set

```
>>> ar.shape = (4, 4)  # ar.shape = 4 would have been also valid.
>>> ar.coord = 0.      # ar.coord = (0.,0.) would have been accepted, too.
>>> ar.storageID = S1.ID
>>> ar.psize = None
```

Now we can construct the View. Upon construction, the View will access information from Storage S1 to initialize other attributes for access and geometry.

```
>>> V1 = View(C1, ID=None, accessrule=ar)
```

We see that a number of the accessrule items appear in the View now.

```
>>> print V1.shape
[4 4]

>>> print V1.coord
[ 0.  0.]

>>> print V1.storageID
S0000
```

A few others were set by the automatic update of Storage S1.

```
>>> print V1.psize
[ 0.1  0.1]

>>> print V1.storage
          S0000 :    0.00 MB :: data=(1, 7, 7) @float64 psize=[ 0.1  0.1] center=[ 3.2  3.2]
```

The update also set new attributes of the View which all start with a lower d and are locally cached information about data access.

```
>>> print V1.dlayer, V1.dlow, V1.dhigh
0 [1 1] [5 5]
```

Finally, we can retrieve the data subset by applying the View to the storage.

```
>>> data = S1[V1]
>>> print data
[[-0.44 -0.34 -0.24 -0.14]
 [-0.34 -0.24 -0.14 -0.04]
 [-0.24 -0.14 -0.04  0.06]
 [-0.14 -0.04  0.06  0.16]]
```

It does not matter if we apply the View to Storage S1 or the container C1, or use the View internal View.*data()* property.

```
>>> print np.allclose(data, C1[V1])
True

>>> print np.allclose(data, V1.data)
True
```

The first access yielded a similar result because the *storageID* S0000 is in C1 and the second acces method worked because it uses the View's *storage* attribute.

```
>>> print V1.storage is S1
True

>>> print V1.storageID in C1.S.keys()
True
```

We observe that the coordinate [0.0,0.0] is not part of the grid in S1 anymore. Consequently, the View was put as close to [0.0,0.0] as possible. The coordinate in data space that the View would have as center is the attribute pcoord() while *dcoord()* is the closest data coordinate. The difference is held by *sp()* such that a subpixel correction may be applied if needed (future release)

```
>>> print V1.dcoord, V1.pcoord, V1.sp
[3 3] [ 3.2  3.2] [ 0.2  0.2]
```

Note: Please note that we cannot guarantee any API stability for other attributes / properties besides *.data*, *.shape* and *.coord*

If we set the coordinate to some other value in the grid, we can eliminate the subpixel misfit. By changing the *.coord* property, we need to update the View manually, as the View-Storage interaction is non-automatic apart from the moment the View is constructed - a measure to prevent unwanted feedback loops.

```
>>> V1.coord = (0.08, 0.08)
>>> S1.update_views(V1)
>>> print V1.dcoord, V1.pcoord, V1.sp
[4 4] [ 4.  4.] [ 0.  0.]
```

We observe that the high range limit of the View is close to the border of the data buffer.

```
>>> print V1.dhigh
[6 6]
```

What happens if we push the coordinate further?

```
>>> V1.coord = (0.28, 0.28)
>>> S1.update_views(V1)
>>> print V1.dhigh
[8 8]
```

Now the higher range limit of the View is off bounds for sure. Applying this View to the Storage may lead to undesired behavior, i.e. array concatenation or data access errors.

```
>>> print S1[V1]
[[ 0.16  0.26  0.36]
 [ 0.26  0.36  0.46]
 [ 0.36  0.46  0.56]]

>>> print S1[V1].shape, V1.shape
(3, 3) [4 4]
```

One important feature of the *Storage* class is that it can detect all out-of-bounds accesses and reformat the data buffer accordingly. A simple call to *Storage.reformat()* should do.

```
>>> print S1.shape
(1, 7, 7)
```

```
>>> mn = S1[V1].mean()
>>> S1.fill_value = mn
>>> S1.reformat()
>>> print S1.shape
(1, 4, 4)
```

Oh no, the Storage data buffer has shrunk! But don't worry, that is intended behavior. A call to *.reformat()* crops and pads the data buffer around all **active** Views. We would need to set `S1.padonly = True` if we wanted to avoid that the data buffer is cropped. We leave this as an exercise for now. Instead, we add a new View at a different location to verify that the buffer will adapt to reach both Views.

```
>>> ar2 = ar.copy()
>>> ar2.coord = (-0.82, -0.82)
>>> V2 = View(C1, ID=None, accessrule=ar2)
>>> S1.fill_value = 0.
>>> S1.reformat()
>>> print S1.shape
(1, 15, 15)
```

Ok, we note that the buffer has grown in size. Now, we give the new View some copied values of the other view to make the View appear in a plot.

```
>>> V2.data = V1.data.copy()
>>> fig = u.plot_storage(S1, 2)
```

See Fig. 1.3 for the plotted image.

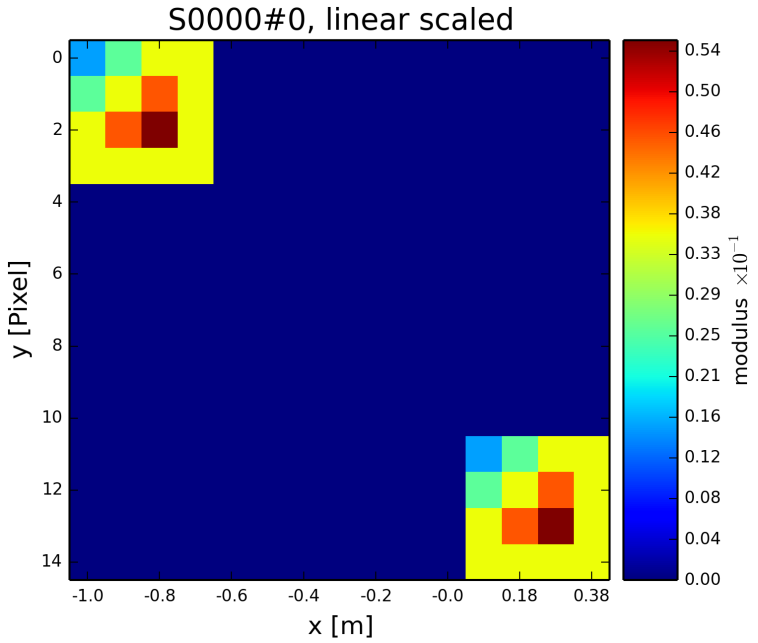


Fig. 1.3: Storage with 4x4 views of the same content.

We observe that the data buffer spans both views. Now let us add more and more Views!

```
>>> for i in range(1, 11):
>>>     ar2 = ar.copy()
>>>     ar2.coord = (-0.82+i*0.1, -0.82+i*0.1)
>>>     View(C1, ID=None, accessrule=ar2)
```

A handy method of the *Storage* class is to determine the *coverage*, which maps, for every pixel of the storage, the number of views having access to this pixel.

```
>>> S1.data[:] = S1.get_view_coverage()
>>> fig = u.plot_storage(S1, 3)
```

See Fig. 1.4 for the plotted image.

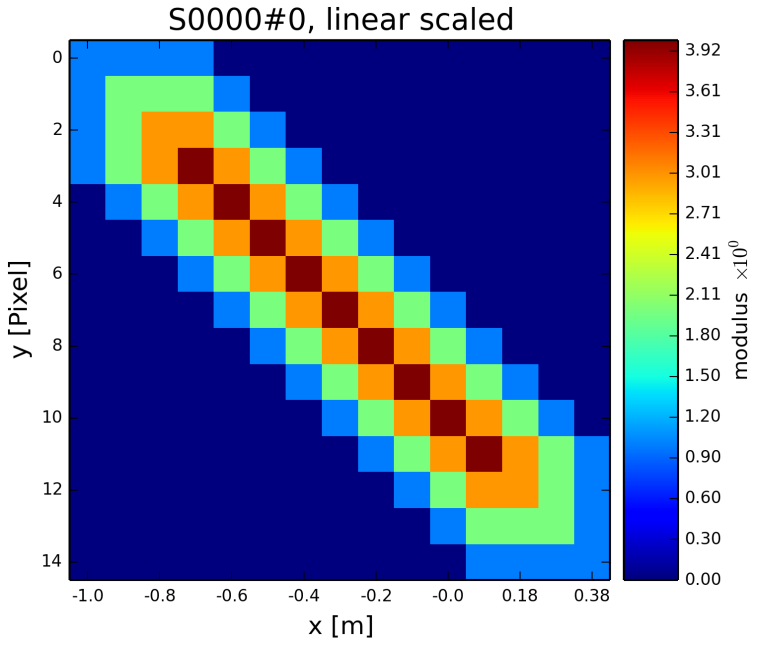


Fig. 1.4: View coverage of data buffer of S1.

Another handy feature of the *View* class is that it automatically creates a Storage instance to a storageID if that ID does not already exist.

```
>>> ar = DEFAULT_ACCESSRULE.copy()
>>> ar.shape = 200
>>> ar.coord = 0.
>>> ar.storageID = 'S100'
>>> ar.psize = 1.0
>>> V3=View(C1, ID=None, accessrule=ar)
```

Finally we have a look at the mischief we managed so far.

```
>>> print C1.formatted_report()
(C)ontnr : Memory : Shape            : Pixel size      : Dimensions      : Views
(S)torgs : (MB)   : (Pixel)          : (meters)        : (meters)        : act.
-------------------------------------------------------------------------------
None     :    0.3 : float64
S100     :    0.3 :       1*200*200 :   1.00*1.00e+00 :   2.00*2.00e+02 :     1
S0000    :    0.0 :         1*15*15 :   1.00*1.00e-01 :   1.50*1.50e+00 :    12
```

1.2.2 Tutorial: Modeling the experiment - Pod, Geometry

Note: This tutorial was generated from the python source [ptypy_root]/tutorial/simupod.py using ptypy/doc/script2rst.py. You are encouraged to modify the parameters and rerun the tutorial with:

```
$ python [ptypy_root]/tutorial/simupod.py
```

In the *Tutorial: Data access - Storage, View, Container* we have learned to deal with the basic storage-and-access classes on small toy arrays.

In this tutorial we will learn how to create *POD* and *Geo* instances to imitate a ptychography experiment and to use larger arrays.

We would like to point out that the "data" created her is not actual data. There is neither light or other wave-like particle involved, nor actual diffraction. You will also not find an actual movement of motors or stages, nor is there an actual detector. Everything should be understood as a test for this software. The selected physical quantities only simulate an experimental setup.

We start again with importing some modules.

```
>>> import matplotlib as mpl
>>> import numpy as np
>>> import ptypy
>>> from ptypy import utils as u
>>> from ptypy.core import View, Container, Storage, Base, POD
>>> plt = mpl.pyplot
>>> import sys
>>> scriptname = sys.argv[0].split('.')[0]
```

We create a managing top-level class instance. We will not use the the *Ptycho* class for now, as its rich set of methods may be a bit overwhelming to start with. Instead we take a plain *Base* instance.

```
>>> P = Base()
>>> P.CType = np.complex128
>>> P.FType = np.float64
```

Set experimental setup geometry and create propagator

Here, we accept a little help from the *Geo* class to provide a propagator and pixel sizes for sample and detector space.

```
>>> from ptypy.core import geometry
>>> g = u.Param()
>>> g.energy = None  # u.keV2m(1.0)/6.32e-7
>>> g.lam = 5.32e-7
>>> g.distance = 15e-2
>>> g.psize = 24e-6
>>> g.shape = 256
>>> g.propagation = "farfield"
>>> G = geometry.Geo(owner=P, pars=g)
```

The Geo instance `G` has done a lot already at this moment. For example, we find forward and backward propagators at `G.propagator.fw` and `G.propagator.bw`. It has also calculated the appropriate pixel size in the sample plane (aka resolution),

```
>>> print G.resolution
[  1.29882812e-05   1.29882812e-05]
```

which sets the shifting frame to be of the following size:

```
>>> fsize = G.shape * G.resolution
>>> print "%.2fx%.2fmm" % tuple(fsize*1e3)
3.32x3.32mm
```

Create probing illumination

Next, we need to create a probing illumination. We start with a suited container that we call *probe*

```
>>> P.probe = Container(P, 'Cprobe', data_type='complex')
```

For convenience, there is a test probing illumination in PtyPy‘s resources.

```
>>> from ptypy.resources import moon_pr
>>> pr = -moon_pr(G.shape)
>>> pr = P.probe.new_storage(data=pr, psize=G.resolution)
>>> fig = u.plot_storage(pr, 0)
```

See Fig. 1.5 for the plotted image.

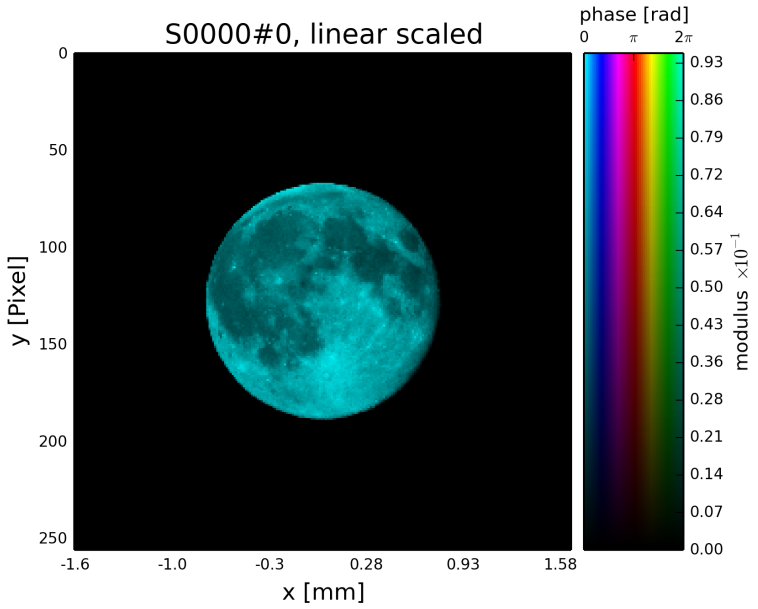


Fig. 1.5: Ptypy’s default testing illumination, an image of the moon.

Of course, we could have also used the coordinate grids from the propagator to model a probe,

```
>>> y, x = G.propagator.grids_sam
>>> apert = u.smooth_step(fsize[0]/5-np.sqrt(x**2+y**2), 1e-6)
>>> pr2 = P.probe.new_storage(data=apert, psize=G.resolution)
>>> fig = u.plot_storage(pr2, 1, channel='c')
```

See Fig. 1.6 for the plotted image.

or the coordinate grids from the Storage itself.

```
>>> pr3 = P.probe.new_storage(shape=G.shape, psize=G.resolution)
>>> y, x = pr3.grids()
>>> apert = u.smooth_step(fsize[0]/5-np.abs(x), 3e-5)*u.smooth_step(fsize[1]/5-np.abs(y), 3e
>>> pr3.fill(apert)
>>> fig = u.plot_storage(pr3, 2)
```

See Fig. 1.7 for the plotted image.

In order to put some physics in the illumination we set the number of photons to 1 billion

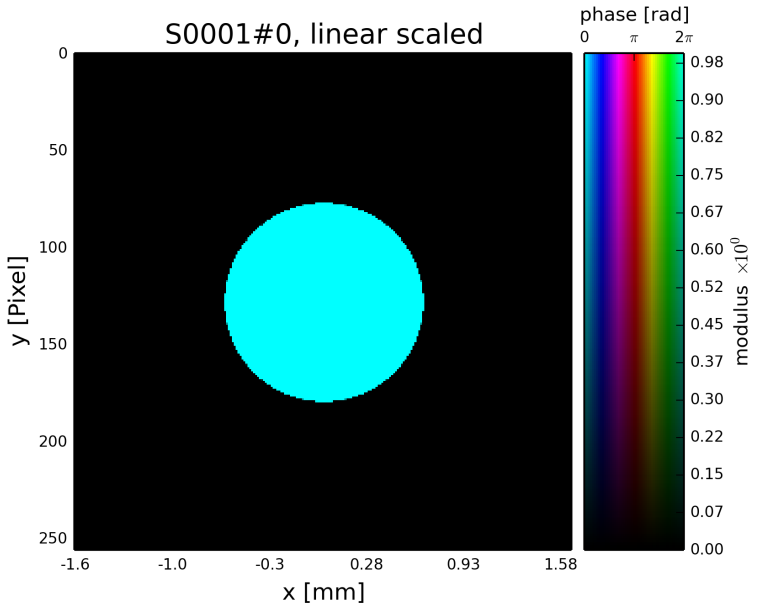


Fig. 1.6: Round test illumination (not necessarily smoothly varying).

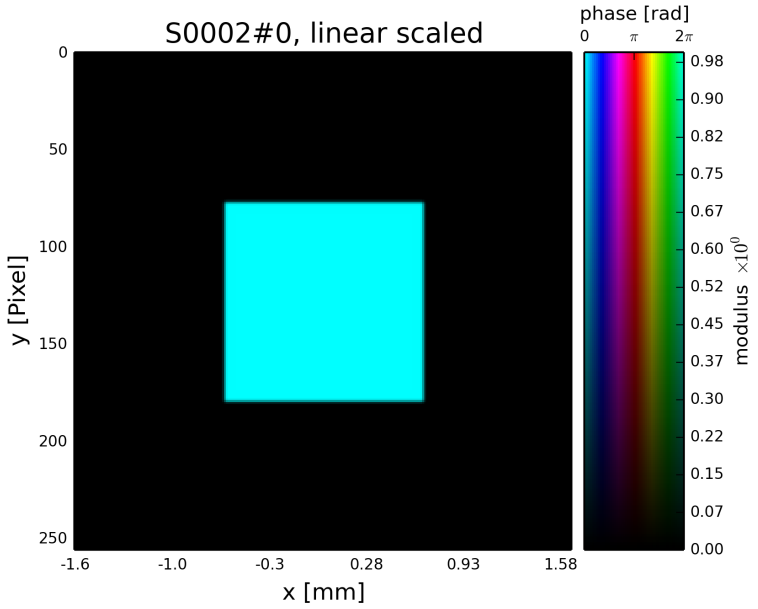


Fig. 1.7: Square test illumination.

```
>>> for pp in [pr, pr2, pr3]:
>>>     pp.data *= np.sqrt(1e9/np.sum(pp.data*pp.data.conj()))
```

and we quickly check if the propagation works.

```
>>> ill = pr.data[0]
>>> propagated_ill = G.propagator.fw(ill)
>>> fig = plt.figure(3)
>>> ax = fig.add_subplot(111)
>>> im = ax.imshow(np.log10(np.abs(propagated_ill)+1))
>>> plt.colorbar(im)
```

See Fig. 1.8 for the plotted image.

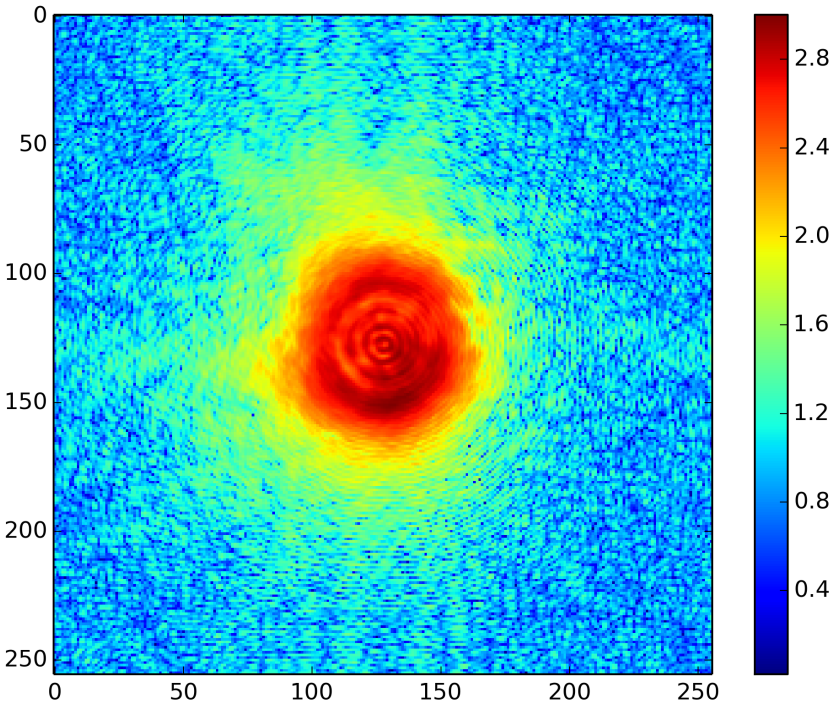


Fig. 1.8: Logarithmic intensity of propagated illumination.

Create scan pattern and object

We use the *ptypy.core.xy* module to create a scan pattern.

```
>>> pos = u.Param()
>>> pos.model = "round"
>>> pos.spacing = fsize[0]/8
>>> pos.steps = None
>>> pos.extent = fsize*1.5
>>> from ptypy.core import xy
>>> positions = xy.from_pars(pos)
>>> fig = plt.figure(4)
>>> ax = fig.add_subplot(111)
>>> ax.plot(positions[:, 1], positions[:, 0], 'o-')
```

See Fig. 1.9 for the plotted image.

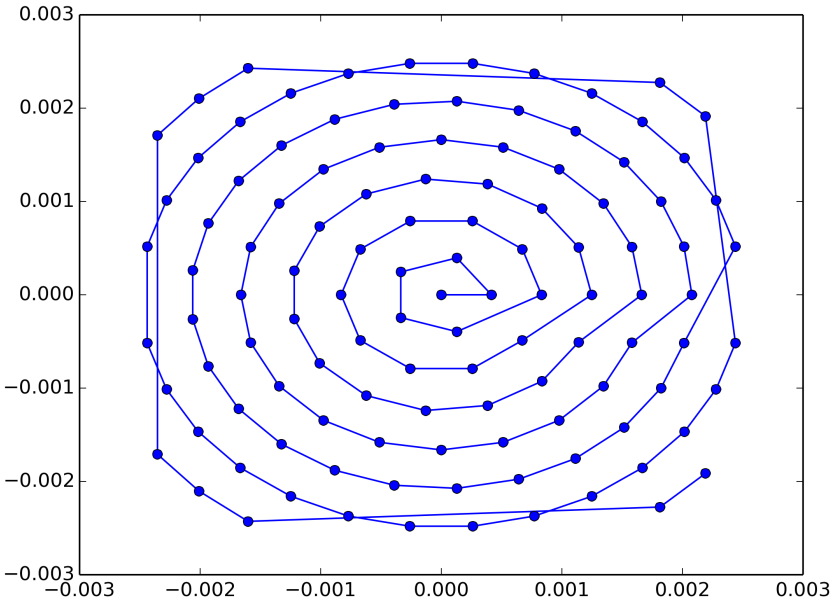


Fig. 1.9: Created scan pattern.

Next, we need to model a sample through an object transmisson function. We start of with a suited container which we call *obj*.

```
>>> P.obj = Container(P, 'Cobj', data_type='complex')
```

As we have learned from the previous *Tutorial: Data access - Storage, View, Container*, we can use `View`'s to create a Storage data buffer of the right size.

```
>>> oar = View.DEFAULT_ACCESSRULE.copy()
>>> oar.storageID = 'S00'
>>> oar.psize = G.resolution
>>> oar.layer = 0
>>> oar.shape = G.shape
>>> oar.active = True
```

```
>>> for pos in positions:
>>>     # the rule
>>>     r = oar.copy()
>>>     r.coord = pos
>>>     V = View(P.obj, None, r)
```

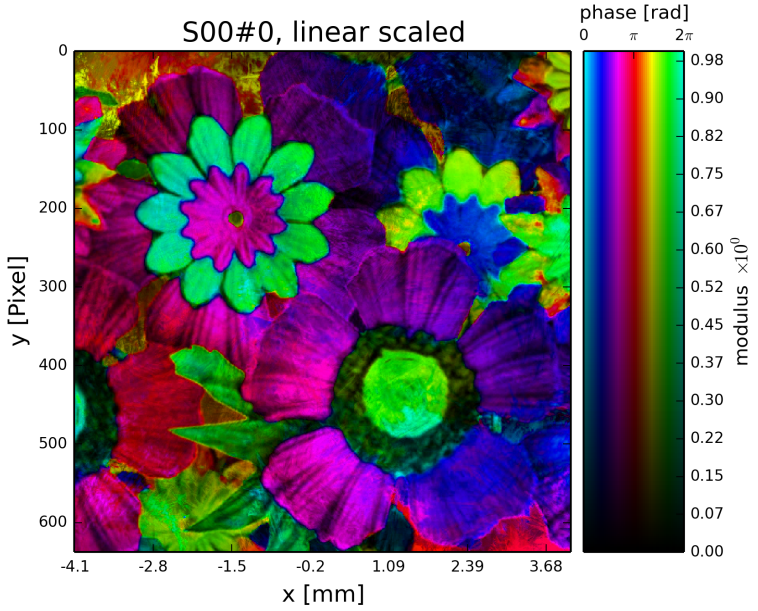
We let the Storages in `P.obj` reformat in order to include all Views. Conveniently, this can be initiated from the top with Container.`reformat()`

```
>>> P.obj.reformat()
>>> print P.obj.formatted_report()
(C)ontnr : Memory : Shape             : Pixel size      : Dimensions      : Views
(S)torgs : (MB)   : (Pixel)           : (meters)        : (meters)        : act.
-------------------------------------------------------------------------------
Cobj     :    6.5 : complex128
S00      :    6.5 :        1*638*632 :   1.30*1.30e-05 :   8.29*8.21e-03 :   114
```

At last we fill the object Storage `S00` with a complex transmission. Again there is a convenience transmission function in the resources

```
>>> from ptypy.resources import flower_obj
>>> storage = P.obj.storages['S00']
>>> storage.fill(flower_obj(storage.shape[-2:]))
>>> fig = u.plot_storage(storage, 5)
```

See Fig. 1.2.2 for the plotted image.



Creating additional Views and the PODs

A single coherent propagation in PtyPy is represented by an instance of the *POD* class.

```
>>> print POD.__doc__

    POD : Ptychographic Object Descriptor

    A POD brings together probe view, object view and diff view. It also
    gives access to "exit", a (coherent) exit wave, and to propagation
    objects to go from exit to diff space.


>>> print POD.__init__.__doc__

        Parameters
        ----------
        ptycho : Ptycho
            The instance of Ptycho associated with this pod.

        ID : str or int
            The pod ID, If None it is managed by the ptycho.

        views : dict or Param
            The views. See :py:attr:`DEFAULT_VIEWS`.

        geometry : Geo
            Geometry class instance and attached propagator
```

For creating a single POD we need a *View* to *probe*, *object*, *exit* wave and *diff* raction containers as well as the *Geo* class instance which represents the experimental setup.

First we create the missing *Container*'s.

```
>>> P.exit = Container(P, 'Cexit', data_type='complex')
>>> P.diff = Container(P, 'Cdiff', data_type='real')
>>> P.mask = Container(P, 'Cmask', data_type='real')
```

We start with one POD and its views.

```
>>> objviews = P.obj.views.values()
>>> obview = objviews[0]
```

We construct the probe View.

```
>>> probe_ar = View.DEFAULT_ACCESSRULE.copy()
>>> probe_ar.psize = G.resolution
>>> probe_ar.shape = G.shape
>>> probe_ar.active = True
>>> probe_ar.storageID = pr.ID
>>> prview = View(P.probe, None, probe_ar)
```

We construct the exit wave View. This construction is shorter as we only change a few bits in the access rule.

```
>>> exit_ar = probe_ar.copy()
>>> exit_ar.layer = 0
>>> exit_ar.active = True
>>> exview = View(P.exit, None, exit_ar)
```

We construct diffraction and mask Views. Even shorter is the construction of the mask View as, for the mask, we are essentially using the same access as for the diffraction data.

```
>>> diff_ar = probe_ar.copy()
>>> diff_ar.layer = 0
>>> diff_ar.active = True
>>> diff_ar.psize = G.psize
>>> mask_ar = diff_ar.copy()
>>> maview = View(P.mask, None, mask_ar)
>>> diview = View(P.diff, None, diff_ar)
```

Now we can create the POD.

```
>>> pods = []
>>> views = {'probe': prview, 'obj': obview, 'exit': exview, 'diff': diview, 'mask': maview}
>>> pod = POD(P, ID=None, views=views, geometry=G)
>>> pods.append(pod)
```

The *POD* is the most important class in PtyPy. Its instances are used to write the reconstruction algorithms using its attributes as local references. For example we can create and store an exit wave in the following convenient fashion:

```
>>> pod.exit = pod.probe * pod.object
```

The result of the calculation above is stored in the appropriate storage of `P.exit`. Therefore we can use this command to plot the result.

```
>>> exit_storage = P.exit.storages.values()[0]
>>> fig = u.plot_storage(exit_storage, 6)
```

See Fig. 1.10 for the plotted image.

The diffraction plane is also conveniently accessible with

```
>>> pod.diff = np.abs(pod.fw(pod.exit))**2
```

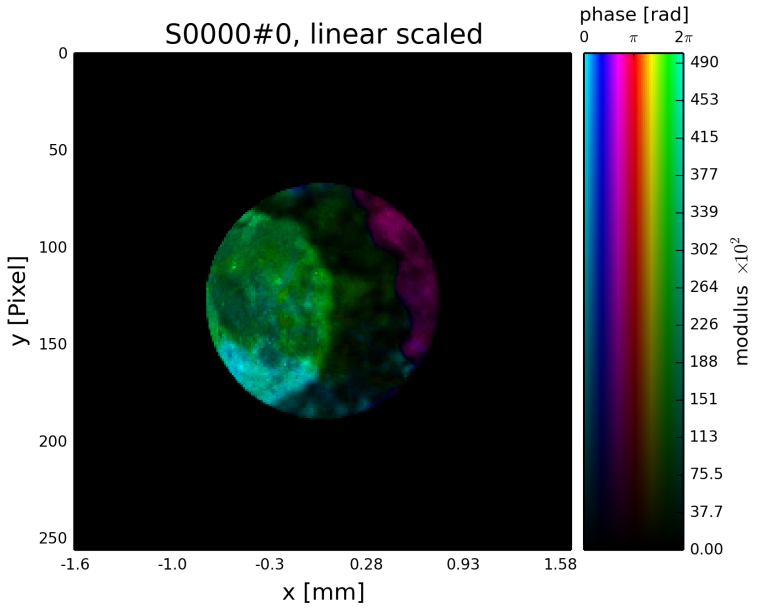


Fig. 1.10: Simulated exit wave using a pod

The result is stored in the diffraction container.

```
>>> diff_storage = P.diff.storages.values()[0]
>>> fig = u.plot_storage(diff_storage, 7, modulus='log')
```

See Fig. 1.2.2 for the plotted image.

Creating the rest of the pods is now straight-forward since the data accesses are similar.

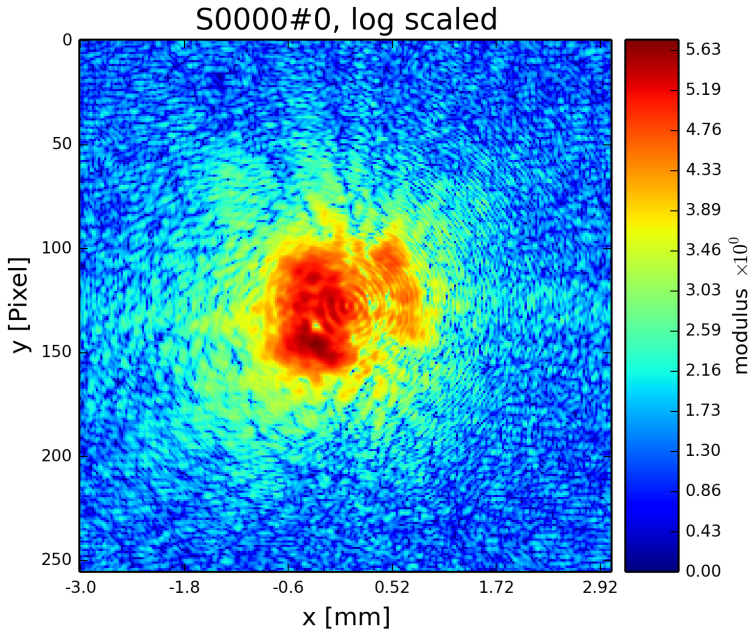
```
>>> for obview in objviews[1:]:
>>>     # we keep the same probe access
>>>     prview = View(P.probe, None, probe_ar)
>>>     # For diffraction and exit wave we need to increase the
>>>     # layer index as there is a new exit wave and diffraction pattern for each
>>>     # scan position
>>>     exit_ar.layer += 1
>>>     diff_ar.layer += 1
>>>     exview = View(P.exit, None, exit_ar)
>>>     maview = View(P.mask, None, mask_ar)
>>>     diview = View(P.diff, None, diff_ar)
>>>     views = {'probe': prview, 'obj': obview, 'exit': exview, 'diff': diview, 'mask': mav
>>>     pod = POD(P, ID=None, views=views, geometry=G)
>>>     pods.append(pod)
```

We let the storage arrays adapt to the new Views.

```
>>> for C in [P.mask, P.exit, P.diff, P.probe]:
>>>     C.reformat()
```

And the rest of the simulation fits in three lines of code!

```
>>> for pod in pods:
>>>     pod.exit = pod.probe * pod.object
>>>     # we use Poisson statistics for a tiny bit of realism in the
>>>     # diffraction images
>>>     pod.diff = np.random.poisson(np.abs(pod.fw(pod.exit))**2)
>>>     pod.mask = np.ones_like(pod.diff)
```



We make a quick check on the diffraction patterns

```
>>> fig = u.plot_storage(diff_storage, 8, slices=':2,:,:', modulus='log')
```

See Fig. 1.11 for the plotted image.

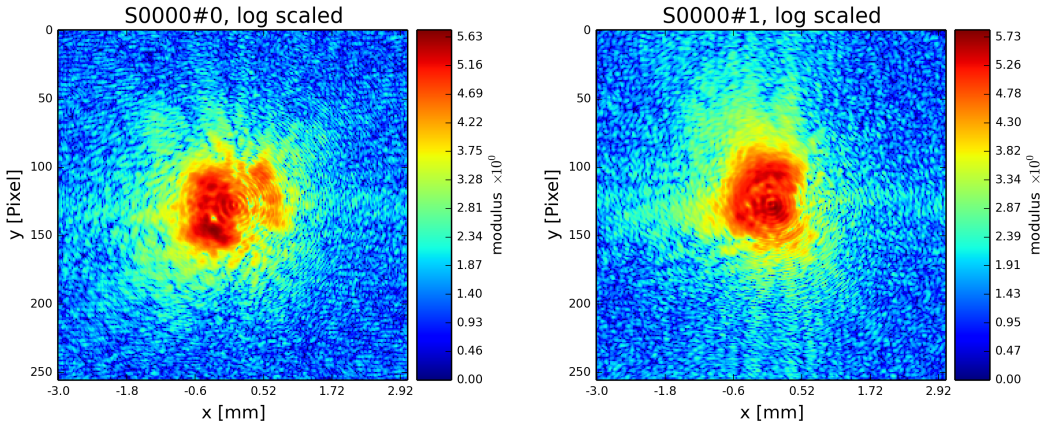


Fig. 1.11: Diffraction patterns with poisson statistics.

Well done! We can move forward to create and run a reconstruction engine as in section *A basic Difference-Map implementation* in *Tutorial: A reconstruction engine from scratch* or store the generated diffraction patterns as in the next section.

Storing the simulation

On unix system we choose the `/tmp` folder

```
>>> save_path = '/tmp/ptypy/sim/'
>>> import os
```

```
>>> if not os.path.exists(save_path):
>>>     os.makedirs(save_path)
```

First, we save the geometric info in a text file.

```
>>> with open(save_path+'geometry.txt', 'w') as f:
>>>     f.write('distance %.4e\n' % G.p.distance)
>>>     f.write('energy %.4e\n' % G.energy)
>>>     f.write('psize %.4e\n' % G.psize[0])
>>>     f.write('shape %d\n' % G.shape[0])
>>>     f.close()
```

Next, we save positions and the diffraction images. We don't burden ourselves for now with converting to an image file format such as .tiff or a data format like .hdf5 but instead we use numpy's binary storage format.

```
>>> with open(save_path+'positions.txt', 'w') as f:
>>>     if not os.path.exists(save_path+'ccd/'):
>>>         os.mkdir(save_path+'ccd/')
>>>     for pod in pods:
>>>         diff_frame = 'ccd/diffraction_%04d.npy' % pod.di_view.layer
>>>         f.write(diff_frame+' %.4e %.4e\n' % tuple(pod.ob_view.coord))
>>>         frame = pod.diff.astype(np.int32)
>>>         np.save(save_path+diff_frame, frame)
```

If you want to learn how to convert this "experiment" into ptypy data file (`.ptyd`), see to *Tutorial : Subclassing PtyScan*

1.2.3 Tutorial: A reconstruction engine from scratch

Note: This tutorial was generated from the python source `[ptypy_root]/tutorial/ownengine.py` using `ptypy/doc/script2rst.py`. You are encouraged to modify the parameters and rerun the tutorial with:

```
$ python [ptypy_root]/tutorial/ownengine.py
```

In this tutorial, we want to provide the information needed to create an engine compatible with the state mixture expansion of ptychogrpahy as described in Thibault et. al 2013 *[Thi2013]* .

First we import ptypy and the utility module

```
>>> import ptypy
>>> from ptypy import utils as u
>>> import numpy as np
```

Preparing a managing Ptycho instance

We need to prepare a managing `Ptycho` instance. It requires a parameter tree, as specified in *Parameter tree structure*

First, we create a most basic input paramater tree. There are many default values, but we specify manually only a more verbose output and single precision for the data type.

```
>>> p = u.Param()
>>> p.verbose_level = 3
>>> p.data_type = "single"
```

Now, we need to create a set of scans that we wish to reconstruct in the reconstruction run. We will use a single scan that we call 'MF' and mark the data source as 'test' to use the PtyPy internal *MoonFlowerScan*

```
>>> p.scans = u.Param()
>>> p.scans.MF = u.Param()
>>> p.scans.MF.data = u.Param()
>>> p.scans.MF.data.source = 'test'
>>> p.scans.MF.data.shape = 128
>>> p.scans.MF.data.num_frames = 600
```

This bare parameter tree will be the input for the *Ptycho* class which is constructed at `level=2`. It means that it creates all necessary basic *Container* instances like *probe*, *object diff* , etc. It also loads the first chunk of data and creates all *View* and *POD* instances, as the verbose output will tell.

```
>>> P = ptypy.core.Ptycho(p, level=2)
Verbosity set to 3
Data type:               single

---- Ptycho init level 1 ------------------------------------------------------
Model: sharing probe between scans (one new probe every 1 scan)
Model: sharing probe between scans (one new probe every 1 scan)

---- Ptycho init level 2 ------------------------------------------------------
Prepared 433 positions
Processing new data.
---- Enter PtyScan.initialixe() ----------------------------------------------
             Common weight : True
                    shape = (128, 128)
All experimental positions : True
                    shape = (433, 2)
Scanning positions (433) are fewer than the desired number of scan points (600).
Resetting `num_frames` to lower value
---- Leaving PtyScan.initialixe() --------------------------------------------
ROI center is [ 64.  64.], automatic guess is [ 63.52886836  63.52655889].
Feeding data chunk
Importing data from MF as scan MF.
End of scan reached
End of scan reached

--- Scan MF photon report ---
Total photons   : 1.51e+10
Average photons : 3.49e+07
Maximum photons : 7.95e+07
-----------------------------

---- Creating PODS -----------------------------------------------------------
Found these probes :
Found these objects:
Process 0 created 433 new PODs, 1 new probes and 1 new objects.

---- Probe initialization ----------------------------------------------------
Initializing probe storage S00G00 using scan MF
Found no photon count for probe in parameters.
Using photon count 7.95e+07 from photon report

---- Object initialization ---------------------------------------------------
Initializing object storage S00G00 using scan MF
Simulation resource is object transmission

---- Creating exit waves -----------------------------------------------------

Process #0 ---- Total Pods 433 (433 active) ----
------------------------------------------------------------------------------
```

```
(C)ontnr : Memory : Shape             : Pixel size       : Dimensions       : Views
(S)torgs : (MB)   : (Pixel)           : (meters)         : (meters)         : act.
-------------------------------------------------------------------------------
Cprobe   :    0.1 : complex64
S00G00   :    0.1 :         1*128*128 :   6.36*6.36e-08 :   8.14*8.14e-06 :   433
Cmask    :    7.1 :   bool
S0000    :    7.1 :       433*128*128 :   1.72*1.72e-04 :   2.20*2.20e-02 :   433
Cexit    :   56.8 : complex64
S0000G00 :   56.8 :       433*128*128 :   6.36*6.36e-08 :   8.14*8.14e-06 :   433
Cobj     :    4.6 : complex64
S00G00   :    4.6 :         1*762*761 :   6.36*6.36e-08 :   4.85*4.84e-05 :   433
Cdiff    :   28.4 : float32
S0000    :   28.4 :       433*128*128 :   1.72*1.72e-04 :   2.20*2.20e-02 :   433
```

A quick look at the diffraction data

```
>>> diff_storage = P.diff.storages.values()[0]
>>> fig = u.plot_storage(diff_storage, 0, slices=(slice(2), slice(None), slice(None)), modul
```

See Fig. 1.12 for the plotted image.

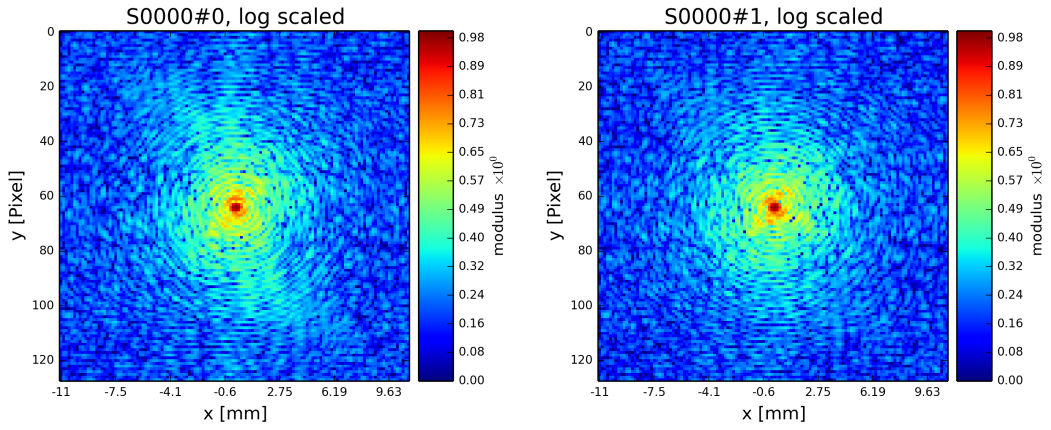


Fig. 1.12: Plot of simulated diffraction data for the first two positions.

We don't need to use PtyPy's `Ptycho` class to arrive at this point. The structure `P` that we arrive with at the end of *Tutorial: Modeling the experiment - Pod, Geometry* suffices completely.

Probe and object are not so exciting to look at for now. As default, probes are initialized with an aperture like support.

```
>>> probe_storage = P.probe.storages.values()[0]
>>> fig = u.plot_storage(P.probe.S['S00G00'], 1)
```

See Fig. 1.13 for the plotted image.

A basic Difference-Map implementation

Now we can start implementing a simple DM algorithm. We need three basic functions, one is the `fourier_update` that implements the Fourier modulus constraint.

$$\psi_{d,\lambda,k} = \mathcal{D}^{-1}_{\lambda,z}\left\{\sqrt{I_d}\frac{\mathcal{D}_{\lambda,z}\{\psi_{d,\lambda,k}\}}{\sum_{\lambda,k}|\mathcal{D}_{\lambda,z}\{\psi_{d,\lambda,k}\}|^2}\right\}$$

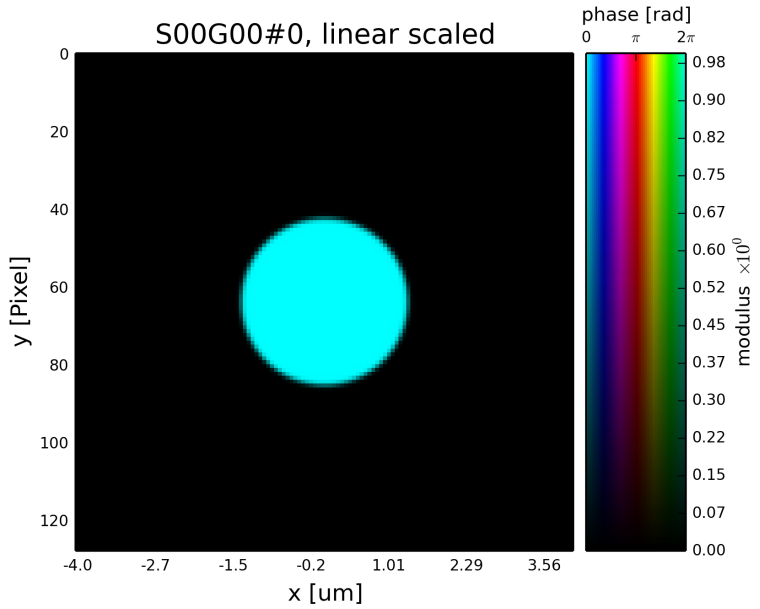


Fig. 1.13: Plot of the starting guess for the probe.

```
>>> def fourier_update(pods):
>>>     import numpy as np
>>>     pod = pods.values()[0]
>>>     # Get Magnitude and Mask
>>>     mask = pod.mask
>>>     modulus = np.sqrt(np.abs(pod.diff))
>>>     # Create temporary buffers
>>>     Imodel = np.zeros_like(pod.diff)
>>>     err = 0.
>>>     Dphi = {}
>>>     # Propagate the exit waves
>>>     for gamma, pod in pods.iteritems():
>>>         Dphi[gamma] = pod.fw(2*pod.probe*pod.object - pod.exit)
>>>         Imodel += Dphi[gamma] * Dphi[gamma].conj()
>>>     # Calculate common correction factor
>>>     factor = (1-mask) + mask * modulus / (np.sqrt(Imodel) + 1e-10)
>>>     # Apply correction and propagate back
>>>     for gamma, pod in pods.iteritems():
>>>         df = pod.bw(factor*Dphi[gamma]) - pod.probe*pod.object
>>>         pod.exit += df
>>>         err += np.mean(np.abs(df*df.conj()))
>>>     # Return difference map error on exit waves
>>>     return err
```

```
>>> def probe_update(probe, norm, pods, fill=0.):
>>>     """
>>>     Updates `probe`. A portion `fill` of the probe is kept from
>>>     iteration to iteration. Requires `norm` buffer and pod dictionary
>>>     """
>>>     probe *= fill
>>>     norm << fill + 1e-10
>>>     for name, pod in pods.iteritems():
>>>         if not pod.active: continue
>>>         probe[pod.pr_view] += pod.object.conj() * pod.exit
>>>         norm[pod.pr_view] += pod.object * pod.object.conj()
```

```
>>>     # For parallel usage (MPI) we have to communicate the buffer arrays
>>>     probe.allreduce()
>>>     norm.allreduce()
>>>     probe /= norm
```

```
>>> def object_update(obj, norm, pods, fill=0.):
>>>     """
>>>     Updates `object`. A portion `fill` of the object is kept from
>>>     iteration to iteration. Requires `norm` buffer and pod dictionary
>>>     """
>>>     obj *= fill
>>>     norm << fill + 1e-10
>>>     for pod in pods.itervalues():
>>>         if not pod.active: continue
>>>         pod.object += pod.probe.conj() * pod.exit
>>>         norm[pod.ob_view] += pod.probe * pod.probe.conj()
>>>     obj.allreduce()
>>>     norm.allreduce()
>>>     obj /= norm
```

```
>>> def iterate(Ptycho, num):
>>>     # generate container copies
>>>     obj_norm = P.obj.copy(fill=0.)
>>>     probe_norm = P.probe.copy(fill=0.)
>>>     errors = []
>>>     for i in range(num):
>>>         err = 0
>>>         # fourier update
>>>         for di_view in Ptycho.diff.V.itervalues():
>>>             if not di_view.active: continue
>>>             err += fourier_update(di_view.pods)
>>>         # probe update
>>>         probe_update(Ptycho.probe, probe_norm, Ptycho.pods)
>>>         # object update
>>>         object_update(Ptycho.obj, obj_norm, Ptycho.pods)
>>>         # print error
>>>         errors.append(err)
>>>         if i % 3==0: print err
>>>     # cleanup
>>>     P.obj.delete_copy()
>>>     P.probe.delete_copy()
>>>     #return error
>>>     return errors
```

We start off with a small number of iterations.

```
>>> iterate(P, 9)
654487.5068
402504.203106
310208.171329
```

We note that the error (here only displayed for 3 iterations) is already declining. That is a good sign. Let us have a look how the probe has developed.

```
>>> fig = u.plot_storage(P.probe.S['S00G00'], 2)
```

See Fig. 1.14 for the plotted image.

Looks like the probe is on a good way. How about the object?

```
>>> fig = u.plot_storage(P.obj.S['S00G00'], 3, slices='0,100:-100,100:-100', mask=150)
```

See Fig. 1.15 for the plotted image.

Ok, let us do some more iterations. 36 will do.

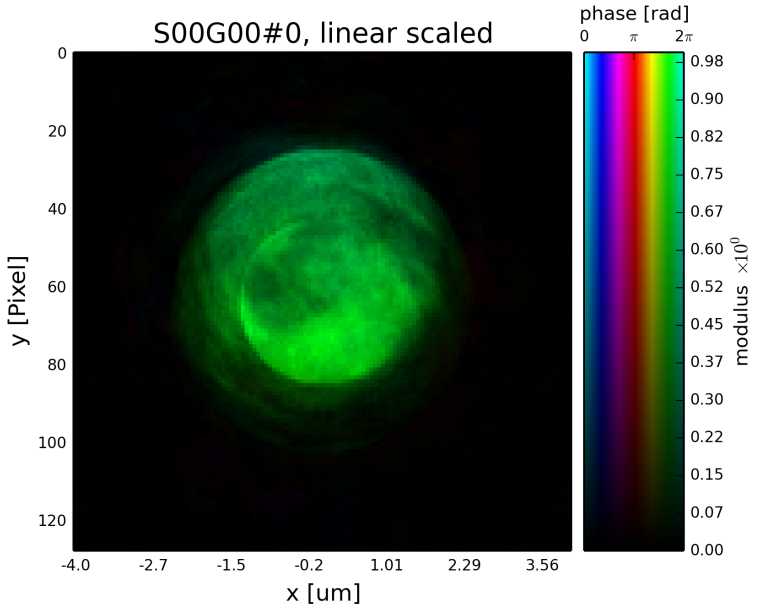


Fig. 1.14: Plot of the reconstructed probe after 9 iterations. We observe that the actaul illumination of the sample must be larger than the initial guess.

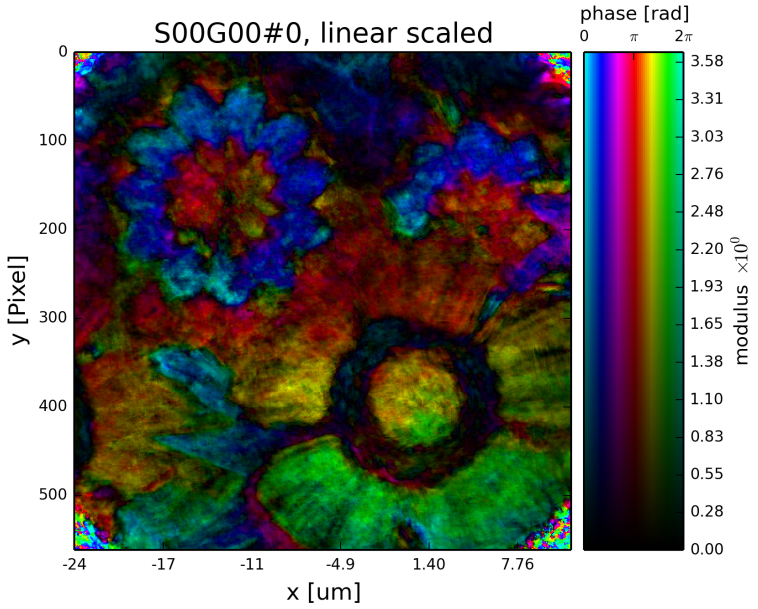


Fig. 1.15: Plot of the reconstructed object after 9 iterations. It is not quite clear what object is reconstructed

```
>>> iterate(P, 36)
214034.748076
134467.004503
96999.4902739
76652.0036214
57138.8288743
37419.3737076
27540.0084388
25271.4170264
25298.0185254
25702.5043972
25502.5819497
24432.8977358
```

Error is still on a steady descent. Let us look at the final reconstructed probe and object.

```
>>> fig = u.plot_storage(P.probe.S['S00G00'], 4)
```

See Fig. 1.16 for the plotted image.

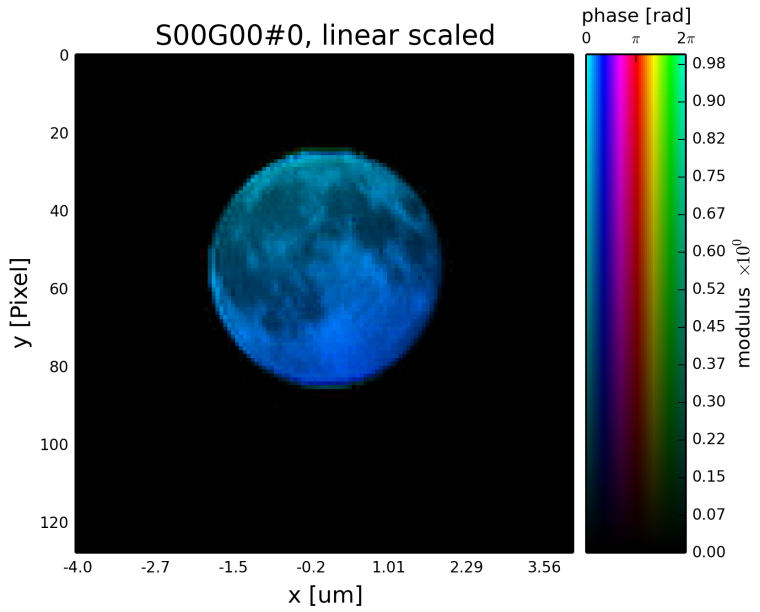


Fig. 1.16: Plot of the reconstructed probe after a total of 45 iterations. It's a moon !

```
>>> fig = u.plot_storage(P.obj.S['S00G00'], 5, slices='0,100:-100,100:-100', mask=150)
```

See Fig. 1.17 for the plotted image.

1.3 Data management

Note: In this chapter, We refer to the *raw input data* with *data* and not to data stored in memory of the computer by `Storage` instances. With the term *preparation* we refer to all data processing steps prior to the reconstruction and avoid the ambiguous term *processing* although it may be more familiar to the reader.

Consider the following generic steps which every ptychographer has to complete prior to a successful image reconstruction.

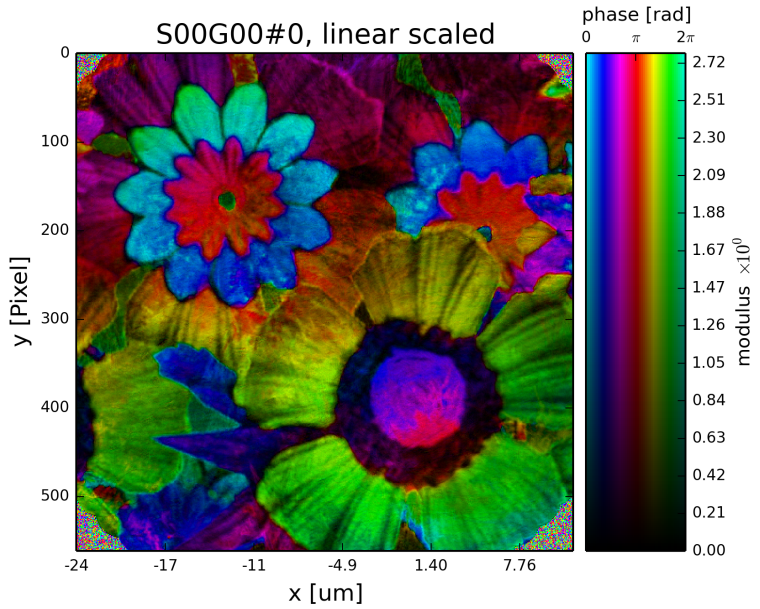


Fig. 1.17: Plot of the reconstructed object after a total of 45 iterations. It's a bunch of flowers !

(A) ***Conducting a scanning diffraction experiment.*** While or after the experiment is performed, the researcher is left with *raw images* acquired from the detector and *meta data* which, in general, consists of scanning positions along with geometric information about the setup, e.g. photon *energy*, propagation *distance*, *detector pixel size* etc.

(B) ***Preparing the data.*** In this step, the user performs a subset of the following actions

- select the appropriate region of the detector where the scattering events were counted,
- apply possible *pixel corrections* to convert the detector counts of the chosen diffraction frame into photon counts, e.g. flat-field and dark-field correction,
- switch image orientation to match with the coordinate system of the reconstruction algorithms,
- assign a suited mask to exclude invalid pixel data (hot or dead pixel, overexposure),
- and/or simply rebin the data.

Finally the user needs to zip the diffraction frames together with the scanning positions.

(C) ***Saving the processed data or feed the data into recontruction process.*** In this step the user needs to save the data in a suitable format or provide the data directly for the reconstruction engine.

Data management in PtyPy deals with **(B)** and **(C)** as a ptychography reconstruction software naturally **cannot** provide actual experimental data. Nevertheless, the treatment of raw data is usually very similar for every experiment. Consequently, PtyPy provides an abstract base class, called `PtyScan`, which aims to help with steps (B) and (C). In order to adapt PtyPy for a specific experimental setup, we simply subclass `PtyScan` and reimplement only that subset of its methods which are affected by the specifics of the experiemental setup (see *Tutorial : Subclassing PtyScan*).

1.3.1 The PtyScan class

`PtyScan` is the abstract base class in PtyPy that manages raw input data.

A PtyScan instance is constructed from a set of generic parameters, see `scan.data` in the ptypy parameter tree.

It provides the following features:

Parallelization When PtyPy is run across several MPI processes, PtyScan takes care of distributing the scan-point indices among processes such that each process only loads the data it will later use in the reconstruction. Hence, the load on the network is not affected by the number of processes. The parallel behavior of `PtyScan`, is controlled by the parameter `scan.data.load_parallel`. It uses the `LoadManager`

Preparation PtyScan can handle a few of the raw processing steps mentioned above.

- Selection a region-of-interest from the raw detector image. This selection is controlled by the parameters `scan.data.auto_center`, and `scan.data.shape` and `scan.data.center`.
- Switching of orientation and rebinning are controlled by `scan.data.orientation` and `scan.data.rebin`.
- Finding a suitable mask or weight for pixel correction is left to the user, as this is a setup-specific implementation. See `load_weight()`, `load_common()`, `load()` and `correct()` for detailed explanations.

Packaging PtyScan packs the prepared *data* together with the used scan point *indices*, scan *positions* and a *weight* (=mask) and geometric *meta* information. This package is requested by the managing instance `ModelManager` on the call `new_data()`.

Because data acquisition and preparation can happen during a reconstruction process, it is possible to specify the minimum number of data frames passed to each process on a *new_data()* by setting the value of `scan.data.min_frames`. The total number of frames processed for a scan is set by `scan.data.num_frames`.

If not extracted from other files, the user may set the photon energy with `scan.data.energy`, the propagation distance from sample to detector with `scan.data.distance` and the detector pixel size with `scan.data.psize`.

Storage PtyScan and its subclass are capable of storing the data in an *hfd5*-compatible *[HDF]* file format. The data file names have a custom suffix: `.ptyd`.

A detailed overview of the *.ptyd* data file tree is given below in the section *Ptyd file format*

The parameters `scan.data.save` and `scan.data.chunk_format` control the way PtyScan saves the processed data.

Note: Although *h5py [h5py]* supports parallel write, this feature is not used in ptypy. At the moment, all mpi nodes send their prepared data to the master node which writes the date to a file.

1.3.2 Usage scenarios

The PtyScan class of PtyPy provides support for three use cases.

Beamline integreted use.

In this use case, the researcher has integrated PtyPy into the beamline end-station or experimental setup with the help of a custom subclass of `PtyScan` that we call `UserScan`. This subclass has its own methods to extract many of the of the generic parameters of `scan.data` and also defaults for specific custom parameters, for instance file paths or file name patterns (for a detailed introduction on how to subclass PtyScan, see *Tutorial : Subclassing PtyScan*). Once the experiment is completed, the researcher can initiate a reconstruction directly from raw data with a standard reconstruction script.

Post preparation use.

In this use case, the experiment is long passed and the researcher has either used custom subclass of PtyScan or *any other script* that generates a compatible .hdf5 file (see *here*) to save prepared data of that experiment. Reconstruction is supposed to work when passing the data file path in the parameter tree.

Only the input file path needs to be passed either with `source` or with `dfile` when `source` takes the value `'file'`. In that latter case, secondary processing and saving to another file is not supported, while it is allowed in the first case. While the latter case seems infavorable due to the

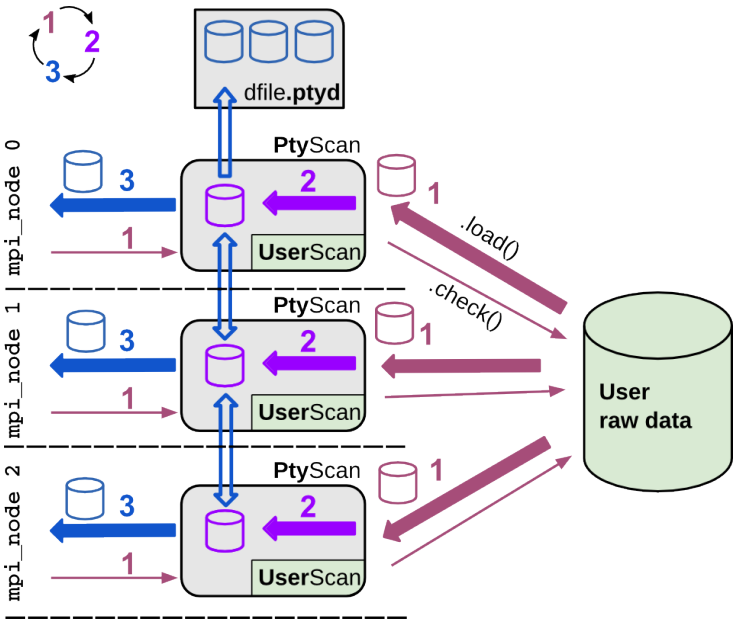


Fig. 1.18: Intergrated use case of *PtyScan*.

A custom subclass `UserScan` serves as a translator between PtyPy's generic parameters and data types and the raw image data and meta data from the experiment. Typically the experiment has to be completed before a reconstruction is started, but with some effort it is even possible to have the reconstruction start immediately after acquisition of the first frame. As data preparation is blended in with the reconstruction process, the reconstruction holds when new data is prepared. Optionally, the prepared data is saved to a `.ptyd` file to avoid having to run the preparation steps for subsequent reconstruction runs.

lack of secondary preparation options, it is meant as a user-friendly transition switch from the first reconstruction at the experiment to post-experiment analysis. Only the `source` parameter needs to be altered in script from `<..>.data.source=<recipe>` to `<..>.data.source='file'` while the rest of the parameters are ignored and may remain untouched.

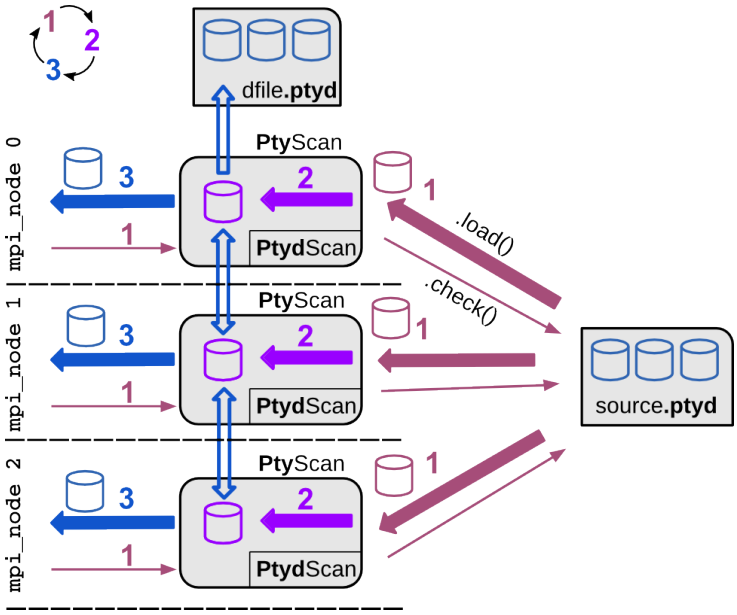


Fig. 1.19: Standard supported use case of `PtyScan`.

If a structure-compatible (see *Ptyd file format*) `*.hdf5`-file is available, PtyPy can be used without customizing a subclass of `PtyScan`. It will use the shipped subclass `PtydScan` to read in the (prepared) raw data.

Preparation and reconstruction on-the-fly with data acquisition.

This use case is for even tighter beamline integration and on-the-fly scans. The researcher has mastered a suitable subclass `UserScan` to prepare data from the setup. Now, the preparation happens in a separate process while image frames are acquired. This process runs a python script where the subclass `UserScan` prepares the data using the `auto()` method. The `save` parameter is set to 'link' in order to create a separate file for each data chunk and to avoid write access on the source file. The chunk files are linked back into the main source `.ptyd` file.

All reconstruction processes may access the prepared data without overhead or notable pauses in the reconstruction. For PtyPy there is no difference if compared to a single source file (a feature of *[HDF]*).

1.3.3 Ptyd file format

Ptypy uses the python module **h5py** *[h5py]* to store and load data in the **H**ierarchical **D**ata **F**ormat *[HDF]* . HDF resembles very much a directory/file tree of today's operating systems, while the "files" are (multidimensonial) datasets.

Ptypy stores and loads the (processed) experimental data in a file with extension *.ptyd*, which is a hdf5-file with a data tree of very simple nature. Comparable to tagged image file formats like *.edf* or *.tiff*, the `ptyd` data file seperates meta information (stored in `meta/`) from the actual data payload (stored in `chunks/`). A schematic overview of the data tree is depicted below.

```
*.ptyd/

      meta/
```

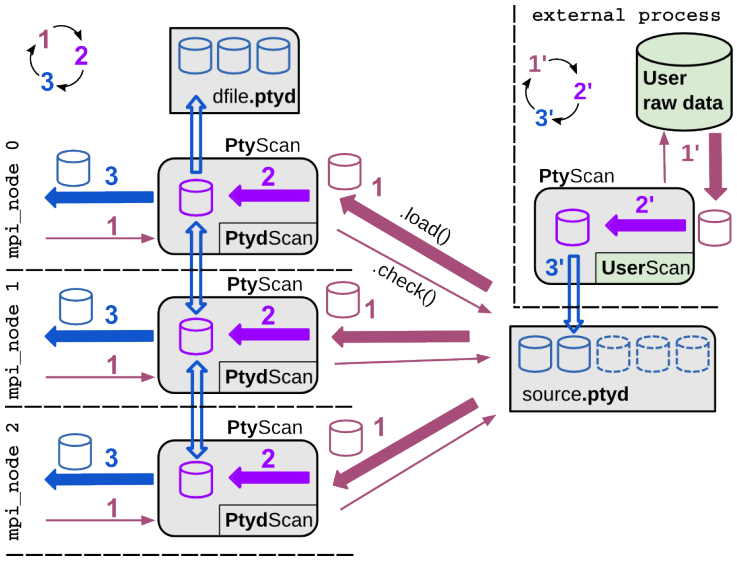


Fig. 1.20: On-the-fly or demon-like use case of `PtyScan`.
A separate process prepares the data *chunks* and saves them in separate files which are linked back into the source data file. This process may run silently as a ''demon" in the background. Reconstructions can start immediately and run without delays or pauses due to data preparation.

```
        [general parameters; optional but very useful]
        version     : str
        num_frames  : int
        label       : str

        [geometric porameters; all optional]
        shape       : int or (int,int)
        energy      : float, optional
        distance    : float, optional
        center      : (float,float) or None, optional
        psize       : float or (float,float), optional
        propagation : "farfield" or "nearfield", optional
        ...

    chunks/

        0/
          data      : array(M,N,N) of float
          indices   : array(M) of int, optional
          positions : array(M ,2) of float
          weights   : same shape as data or empty
        1/
          ...
        2/
          ...
        ...
```

All parameters of `meta/` are a subset of `scan.data`. Omitting any of these parameters or setting the value of the dataset to `'None'` has the same effect.

The first set of parameters

```
version     : str
num_frames  : int
```

```
label       : str
```

are general (optional) parameters.

- `version` is ptypy version this dataset was prepared with (current version is 0.2.0.dev4aff8a9, see `version`).
- `label` is a custom user label. Choose a unique label to your liking.
- `num_frames` indicates how many diffraction image frames are expected in the dataset (see *num_frames*) It is important to set this parameter when the data acquisition is not finished but the reconstruction has already started. If the dataset is complete, the loading class *PtydScan* retrieves the total number of frames from the payload `chunks/`

The next set of optional parameters are

```
shape       : int or (int,int)
energy      : float
distance    : float
center      : (float,float)
psize       : float or (float,float)
propagation : "farfield" or "nearfield"
```

which refer to the experimental scanning geometry.

- `shape` (see *scan.data.shape*)
- `energy` (see *scan.data.energy* or *scan.geometry.energy*)
- `distance` (see *scan.data.distance*)
- `center` : (float,float) (see *scan.data.center*)
- `psize` : float or (float,float) (see *scan.data.psize*)
- `propagation` : “farfield” or “nearfield” (see `scan.data.propagation`)

Finally these parameters will be digested by the *geometry* module in order to provide a suited propagator.

Note: As you may have already noted, there are three ways to specify the geometry of the experiment.

```
bla
```

As walking the data tree and extracting the data from the *hdf5* file is a bit cumbersome with h5py, there are a few convenience function in the *ptypy.io.h5rw* module.

1.3.4 Tutorial : Subclassing PtyScan

Note: This tutorial was generated from the python source `[ptypy_root]/tutorial/subclassptyscan.py` using `ptypy/doc/script2rst.py`. You are encouraged to modify the parameters and rerun the tutorial with:

```
$ python [ptypy_root]/tutorial/subclassptyscan.py
```

In this tutorial, we learn how to subclass *PtyScan* to make ptypy work with any experimental setup.

This tutorial can be used as a direct follow-up to *Tutorial: Modeling the experiment - Pod, Geometry* if section *Storing the simulation* was completed

Again, the imports first.

```
>>> import matplotlib as mpl
>>> import numpy as np
>>> import ptypy
>>> from ptypy import utils as u
```

```
>>> plt = mpl.pyplot
>>> import sys
```

For this tutorial we assume that the data and meta information is in this path:

```
>>> save_path = '/tmp/ptypy/sim/'
```

Furthermore, we assume that a file about the experimental geometry is located at

```
>>> geofilepath = save_path + 'geometry.txt'
>>> print geofilepath
/tmp/ptypy/sim/geometry.txt
```

and has contents of the following form

```
>>> print ''.join([line for line in open(geofilepath, 'r')])
distance 1.5000e-01
energy 2.3319e-03
psize 2.4000e-05
shape 256
```

The scanning positions are in

```
>>> positionpath = save_path + 'positions.txt'
>>> print positionpath
/tmp/ptypy/sim/positions.txt
```

with a list of positions for vertical and horizontanl movement and the image frame from the "camera"

```
>>> print ''.join([line for line in open(positionpath, 'r')][:6])+'....'
ccd/diffraction_0000.npy 1.4658e-03 2.0175e-03
ccd/diffraction_0001.npy 1.8532e-03 1.6686e-03
ccd/diffraction_0002.npy -1.7546e-03 1.1135e-03
ccd/diffraction_0003.npy -1.4226e-03 1.5149e-03
ccd/diffraction_0004.npy -2.0740e-03 1.3049e-04
ccd/diffraction_0005.npy -1.9764e-03 6.4218e-04
....
```

Writing a subclass

A subclass of *PtyScan* takes the same input parameter tree as PtyScan itself, i.e *scan.data*. As the subclass will most certainly require additional parameters, there has to be a flexible additional container. For PtyScan, that is the *scan.data.recipe* parameter. A subclass must extract all additional parameters from this source and, in script, you fill the recipe with the appropriate items.

In this case we can assume that the only parameter of the recipe is the base path `/tmp/ptypy/sim/`. Hence we write

```
>>> RECIPE = u.Param()
>>> RECIPE.base_path = '/tmp/ptypy/sim/'
```

Now we import the default generic parameter set from

```
>>> from ptypy.core.data import PtyScan
>>> DEFAULT = PtyScan.DEFAULT.copy()
```

This would be the perfect point to change any default value. For sure we need to set the recipe parameter

```
>>> DEFAULT.recipe = RECIPE
```

A default data file location may be handy too and we allow saving of data in a single file. And since we know it is simulated data we do not have to find the optical axes in the diffraction pattern with the help of auto_center

```
>>> DEFAULT.dfile = '/tmp/ptypy/sim/npy.ptyd'
>>> DEFAULT.auto_center = False
```

Our defaults are now

```
>>> print u.verbose.report(DEFAULT, noheader=True)
* id3VSFF3C638           : ptypy.utils.parameters.Param(19)
  * positions_theory     : None
  * auto_center          : False
  * chunk_format         : .chunk%02d
  * min_frames           : 1
  * orientation          : None
  * num_frames           : None
  * energy               : None
  * center               : None
  * recipe               : ptypy.utils.parameters.Param(1)
    * base_path          : /tmp/ptypy/sim/
  * psize                : None
  * label                : None
  * load_parallel        : data
  * shape                : None
  * rebin                : None
  * experimentID         : None
  * version              : 0.1
  * save                 : None
  * dfile                : /tmp/ptypy/sim/npy.ptyd
  * distance             : None
```

The simplest subclass of PtyScan would look like this

```
>>> class NumpyScan(PtyScan):
>>>     # We overwrite the DEFAULT with the new DEFAULT.
>>>     DEFAULT = DEFAULT
>>>
>>>     def __init__(self, pars=None, **kwargs):
>>>         # In init we need to call the parent.
>>>         super(NumpyScan, self).__init__(pars, **kwargs)
```

Of course this class does nothing special beyond PtyScan.

An additional step of initialisation would be to retrieve the geometric information that we stored in `geofilepath` and update the input parameters with it.

We write a tiny file parser.

```
>>> def extract_geo(base_path):
>>>     out = {}
>>>     with open(base_path+'geometry.txt') as f:
>>>         for line in f:
>>>             key, value = line.strip().split()
>>>             out[key] = eval(value)
>>>     return out
```

We test it.

```
>>> print extract_geo(save_path)
{'distance': 0.15, 'energy': 0.0023319, 'shape': 256, 'psize': 2.4e-05}
```

That seems to work. We can integrate this parser into the initialisation as we assume that this small access can be done by all MPI nodes without data access problems. Hence, our subclass becomes

```
>>> class NumpyScan(PtyScan):
>>>     # We overwrite the DEFAULT with the new DEFAULT.
>>>     DEFAULT = DEFAULT
>>>
```

```
>>>     def __init__(self, pars=None, **kwargs):
>>>         p = DEFAULT.copy(depth=2)
>>>         p.update(pars)
>>>
>>>         with open(p.recipe.base_path+'geometry.txt') as f:
>>>             for line in f:
>>>                 key, value = line.strip().split()
>>>                 # we only replace Nones or missing keys
>>>                 if p.get(key) is None:
>>>                     p[key] = eval(value)
>>>
>>>         super(NumpyScan, self).__init__(p, **kwargs)
```

Good! Next, we need to implement how the class finds out about the positions in the scan. The method *load_positions()* can be used for this purpose.

```
>>> print PtyScan.load_positions.__doc__

        **Override in subclass for custom implementation**

        *Called in* :py:meth:`initialize`

        Loads all positions for all diffraction patterns in this scan.
        The positions loaded here will be available by all processes
        through the attribute ``self.positions``. If you specify position
        on a per frame basis in :py:meth:`load` , this function has no
        effect.

        If theoretical positions :py:data:`positions_theory` are
        provided in the initial parameter set :py:data:`DEFAULT`,
        specifyiing positions here has NO effect and will be ignored.

        The purpose of this function is to avoid reloading and parallel
        reads on files that may require intense parsing to retrieve the
        information, e.g. long SPEC log files. If parallel reads or
        log file parsing for each set of frames is not a time critical
        issue of the subclass, reimplementing this function can be ignored
        and it is recommended to only reimplement the :py:meth:`load`
        method.

        If `load_parallel` is set to `all` or common`, this function is
        executed by all nodes, otherwise the master node executes this
        function and braodcasts the results to other nodes.

        Returns
        -------
        positions : ndarray
            A (N,2)-array where *N* is the number of positions.

        Note
        ----
        Be aware that this method sets attribute :py:attr:`num_frames`
        in the following manner.

        * If ``num_frames == None`` : ``num_frames = N``.
        * If ``num_frames < N`` , no effect.
        * If ``num_frames > N`` : ``num_frames = N``.
```

The parser for the positions file would look like this.

```
>>> def extract_pos(base_path):
>>>     pos = []
>>>     files = []
```

```
>>>     with open(base_path+'positions.txt') as f:
>>>         for line in f:
>>>             fname, y, x = line.strip().split()
>>>             pos.append((eval(y), eval(x)))
>>>             files.append(fname)
>>>     return files, pos
```

And the test:

```
>>> files, pos = extract_pos(save_path)
>>> print files[:2]
['ccd/diffraction_0000.npy', 'ccd/diffraction_0001.npy']

>>> print pos[:2]
[(0.0014658, 0.0020175), (0.0018532, 0.0016686)]
```

```
>>> class NumpyScan(PtyScan):
>>>     # We overwrite the DEFAULT with the new DEFAULT.
>>>     DEFAULT = DEFAULT
>>>
>>>     def __init__(self,pars=None, **kwargs):
>>>         p = DEFAULT.copy(depth=2)
>>>         p.update(pars)
>>>
>>>         with open(p.recipe.base_path+'geometry.txt') as f:
>>>             for line in f:
>>>                 key, value = line.strip().split()
>>>                 # we only replace Nones or missing keys
>>>                 if p.get(key) is None:
>>>                     p[key]=eval(value)
>>>
>>>         super(NumpyScan, self).__init__(p, **kwargs)
>>>         # all input data is now in self.info
>>>
>>>     def load_positions(self):
>>>         # the base path is now stored in
>>>         base_path = self.info.recipe.base_path
>>>         with open(base_path+'positions.txt') as f:
>>>             for line in f:
>>>                 fname, y, x = line.strip().split()
>>>                 pos.append((eval(y),eval(x)))
>>>                 files.append(fname)
>>>         return np.asarray(pos)
```

One nice thing about rewriting `self.load_positions` is that the maximum number of frames will be set and we do not need to manually adapt *check()*

The last step is to overwrite the actual loading of data. Loading happens (MPI-compatible) in *load()*

```
>>> print PtyScan.load.__doc__

        **Override in subclass for custom implementation**

        Loads data according to node specific scanpoint indeces that have
        been determined by :py:class:`LoadManager` or otherwise

        Returns
        -------
        raw, positions, weight : dict
            Dictionaries whose keys are the given scan point `indices`
            and whose values are the respective frame / position according
            to the scan point index. `weight` and `positions` may be empty

        Note
```

```
        ----
        This is the *most* important method to change when subclassing
        :any:`PtyScan`. Most often it suffices to override the constructor
        and this method to createa subclass of suited for a specific
        experiment.
```

Load seems a bit more complex than `self.load_positions` for its return values. However, we can opt-out of providing weights (masks) and positions, as we have already adapted `self.load_positions` and there were no bad pixels in the (linear) detector

The final subclass looks like this.

```
>>> class NumpyScan(PtyScan):
>>>     # We overwrite the DEFAULT with the new DEFAULT.
>>>     DEFAULT = DEFAULT
>>>
>>>     def __init__(self,pars=None, **kwargs):
>>>         p = DEFAULT.copy(depth=2)
>>>         p.update(pars)
>>>
>>>         with open(p.recipe.base_path+'geometry.txt') as f:
>>>             for line in f:
>>>                 key, value = line.strip().split()
>>>                 # we only replace Nones or missing keys
>>>                 if p.get(key) is None:
>>>                     p[key]=eval(value)
>>>
>>>         super(NumpyScan, self).__init__(p, **kwargs)
>>>         # all input data is now in self.info
>>>
>>>     def load_positions(self):
>>>         # the base path is now stored in
>>>         pos=[]
>>>         base_path = self.info.recipe.base_path
>>>         with open(base_path+'positions.txt') as f:
>>>             for line in f:
>>>                 fname, y, x = line.strip().split()
>>>                 pos.append((eval(y),eval(x)))
>>>                 files.append(fname)
>>>         return np.asarray(pos)
>>>
>>>     def load(self, indices):
>>>         raw = {}
>>>         bp = self.info.recipe.base_path
>>>         for ii in indices:
>>>             raw[ii] = np.load(bp+'ccd/diffraction_%04d.npy' % ii)
>>>         return raw, {}, {}
```

Loading the data

With the subclass we create a scan only using defaults

```
>>> NPS = NumpyScan()
>>> NPS.initialize()
```

In order to process the data. We need to call *auto()* with the chunk size as arguments. It returns a data chunk that we can inspect with *ptypy.utils.verbose.report()*. The information is concatenated, but the length of iterables or dicts is always indicated in parantheses.

```
>>> print u.verbose.report(NPS.auto(80), noheader=True)
* id3VSFF2N0SO           : dict(2)
  * common               : ptypy.utils.parameters.Param(9)
```

```
    * distance           : 0.15
    * center             : [array = [ 128.  128.]]
    * energy             : 0.0023319
    * psize              : [array = [  2.40000000e-05   2.40000000e-05]]
    * label              : None
    * shape              : [array = [256 256]]
    * version            : 0.1
    * experimentID       : None
    * weight2d           : [256x256 bool array]
  * iterable             : list(80)
    * id3VSFF2OGSO       : dict(4)
      * index            : 0
      * data             : [256x256 int32 array]
      * mask             : [256x256 bool array]
      * position         : [array = [ 0.0014658  0.0020175]]
    * id3VSFF2N15G       : dict(4)
      * index            : 1
      * data             : [256x256 int32 array]
      * mask             : [256x256 bool array]
      * position         : [array = [ 0.0018532  0.0016686]]
    * id3VSFF2NEH8       : dict(4)
      * index            : 2
      * data             : [256x256 int32 array]
      * mask             : [256x256 bool array]
      * position         : [array = [-0.0017546  0.0011135]]
    * id3VSFF2NCK0       : dict(4)
      * index            : 3
      * data             : [256x256 int32 array]
      * mask             : [256x256 bool array]
      * position         : [array = [-0.0014226  0.0015149]]
    * id3VSFF2LFK8       : dict(4)
      * index            : 4
      * data             : [256x256 int32 array]
      * mask             : [256x256 bool array]
      * position         : [array = [-0.002074    0.00013049]]
    * ...                : ....


>>> print u.verbose.report(NPS.auto(80), noheader=True)
* id3VSFF2O1VO           : dict(2)
  * common               : ptypy.utils.parameters.Param(9)
    * distance           : 0.15
    * center             : [array = [ 128.  128.]]
    * energy             : 0.0023319
    * psize              : [array = [  2.40000000e-05   2.40000000e-05]]
    * label              : None
    * shape              : [array = [256 256]]
    * version            : 0.1
    * experimentID       : None
    * weight2d           : [256x256 bool array]
  * iterable             : list(34)
    * id3VSFF2OH5G       : dict(4)
      * index            : 80
      * data             : [256x256 int32 array]
      * mask             : [256x256 bool array]
      * position         : [array = [-0.0021597 -0.0012469]]
    * id3VSFF2ND5G       : dict(4)
      * index            : 81
      * data             : [256x256 int32 array]
      * mask             : [256x256 bool array]
      * position         : [array = [-0.0023717  -0.00077061]]
    * id3VSFF2LCB8       : dict(4)
      * index            : 82
```

```
      * data             : [256x256 int32 array]
      * mask             : [256x256 bool array]
      * position         : [array = [-0.0024801  -0.00026067]]
    * id3VSFF2OGK0       : dict(4)
      * index            : 83
      * data             : [256x256 int32 array]
      * mask             : [256x256 bool array]
      * position         : [array = [-0.0024801   0.00026067]]
    * id3VSFF2O1N0       : dict(4)
      * index            : 84
      * data             : [256x256 int32 array]
      * mask             : [256x256 bool array]
      * position         : [array = [-0.00051848 -0.0024393 ]]
    * ...                : ....
```

We observe the second chunk was not 80 frames deep but 34 as we only had 114 frames of data.

So where is the *.ptyd* data-file? As default, PtyScan does not actually save data. We have to manually activate it in in the input paramaters.

```
>>> data = NPS.DEFAULT.copy(depth=2)
>>> data.save = 'append'
>>> NPS = NumpyScan(pars=data)
>>> NPS.initialize()
WARNING root - File /tmp/ptypy/sim/npy.ptyd already exist. Renamed to /tmp/ptypy/sim/npy.pty
```

```
>>> for i in range(50):
>>>     msg = NPS.auto(20)
>>>     if msg == NPS.EOS:
>>>         break
```

We can analyse the saved `npy.ptyd` with `h5info()`

```
>>> from ptypy.io import h5info
>>> print h5info(NPS.info.dfile)
File created : Mon Jun 22 11:27:49 2015
 * chunks [dict 6]:
     * 0 [dict 4]:
         * data [20x256x256 int32 array]
         * indices [list = [0.000000, 1.000000, 2.000000, 3.000000,  ...]]
         * positions [20x2 float64 array]
         * weights [array = []]
     * 1 [dict 4]:
         * data [20x256x256 int32 array]
         * indices [list = [20.000000, 21.000000, 22.000000, 23.000000,  ...]]
         * positions [20x2 float64 array]
         * weights [array = []]
     * 2 [dict 4]:
         * data [20x256x256 int32 array]
         * indices [list = [40.000000, 41.000000, 42.000000, 43.000000,  ...]]
         * positions [20x2 float64 array]
         * weights [array = []]
     * 3 [dict 4]:
         * data [20x256x256 int32 array]
         * indices [list = [60.000000, 61.000000, 62.000000, 63.000000,  ...]]
         * positions [20x2 float64 array]
         * weights [array = []]
     * 4 [dict 4]:
         * data [20x256x256 int32 array]
         * indices [list = [80.000000, 81.000000, 82.000000, 83.000000,  ...]]
         * positions [20x2 float64 array]
         * weights [array = []]
     * 5 [dict 4]:
         * data [14x256x256 int32 array]
```

```
         * indices [list = [100.000000, 101.000000, 102.000000, 103.000000,  ...]]
         * positions [14x2 float64 array]
         * weights [array = []]
 * info [dict 20]:
     * auto_center [scalar = False]
     * center [array = [128 128]]
     * chunk_format [string = ".chunk%02d"]
     * dfile [string = "/tmp/ptypy/sim/npy.ptyd"]
     * distance [scalar = 0.15]
     * energy [scalar = 0.0023319]
     * experimentID [None]
     * label [None]
     * load_parallel [string = "data"]
     * min_frames [scalar = 1]
     * num_frames [None]
     * orientation [None]
     * positions_scan [114x2 float64 array]
     * positions_theory [None]
     * psize [scalar = 2.4e-05]
     * rebin [scalar = 1]
     * recipe [Param 1]:
         * base_path [string = "/tmp/ptypy/sim/"]
     * save [string = "append"]
     * shape [array = [256 256]]
     * version [string = "0.1"]
 * meta [dict 9]:
     * center [array = [ 128.  128.]]
     * distance [scalar = 0.15]
     * energy [scalar = 0.0023319]
     * experimentID [None]
     * label [None]
     * psize [array = [  2.40000000e-05   2.40000000e-05]]
     * shape [array = [256 256]]
     * version [string = "0.1"]
     * weight2d [256x256 bool array]

None
```

Listing the new subclass

In order to make the subclass available in your local PtyPy, navigate to `[ptypy_root]/ptypy/experiment` and paste the content into a new file `user.py`:

```
$ touch [ptypy_root]/ptypy/experiment/user.py
```

Append the following lines into `[ptypy_root]/ptypy/experiment.__init__.py`:

```
from user import NumpyScan
PtyScanTypes.update({'numpy':NumpyScan})
```

Now, your new subclass will be used whenever you pass `'numpy'` for the *scan.data.source* parameter. All special parameters of the class should be passed via the dict *scan.data.recipe*.

1.4 Parameter tree structure

1.4.1 .verbose_level

.verbose_level (*int*)
(0) Verbosity level

B.2. Parameter tree and module reference

.io.rfile

`.io.`**`rfile`** (*str*)
(6) Reconstruction file name (or format string)

Reconstruction file name or format string (constructed against runtime dictionary)

default = `recons/%(run)s/%(run)s_%(engine)s_%(iterations)04d.ptyr`

.io.autosave

`.io.`**`autosave`** (*Param*)
(7) Auto-save options

default = `None`

`.io.autosave.`**`active`** (*bool*)
(8) Activation switch

If `True` the current reconstruction will be saved at regular intervals. **unused**

default = `TRUE`

`.io.autosave.`**`interval`** (*int*)
(9) Auto-save interval

If `>0` the current reconstruction will be saved at regular intervals according to the pattern in `paths.autosave`. If `<=0` not automatic saving

default = `10 (>-1)`

`.io.autosave.`**`rfile`** (*str*)
(10) Auto-save file name (or format string)

Auto-save file name or format string (constructed against runtime dictionary)

default = `dumps/%(run)s/%(run)s_%(engine)s_%(iterations)04d.ptyr`

.io.interaction

`.io.`**`interaction`** (*Param*)
(11) Server / Client parameters

If `None` or `False` is passed here in script instead of a Param, it translates to `active=False` i.e. no ZeroMQ interaction server.

default = `None`

`.io.interaction.`**`active`** (*bool*)
(12) Activation switch

Set to `False` for no ZeroMQ interaction server

default = `TRUE`

`.io.interaction.`**`address`** (*str*)
(13) The address the server is listening to.

Wenn running ptypy on a remote server, it is the servers network address.

default = `tcp://127.0.0.1`

`.io.interaction.`**`port`** (*int*)
(14) The port the server is listening to.

Make sure to pick an unused port with a few unused ports close to it.

default = `5560`

Verbosity level for information logging.

- `0`: Only errors
- `1`: Warning
- `2`: Process Information
- `3`: Object Information
- `4`: Debug

default = `1 (>0, <4)`

1.4.2 .data_type

`.data_type` (*str*)
(1) Reconstruction floating number precision

Reconstruction floating number precision (`'single'` or `'double'`)

default = `single`

1.4.3 .run

`.run` (*str*)
(2) Reconstruction identifier

Reconstruction run identifier. If `None`, the run name will be constructed at run time from other information.

default = `None`

1.4.4 .dry_run

`.dry_run` (*bool*)
(3) Dry run switch

Run everything skipping all memory and cpu-heavy steps (and file saving). **NOT IMPLEMENTED**

default = `FALSE`

1.4.5 .io

`.io` (*Param*)
(4) Global parameters for I/O

default = `None`

.io.home

`.io.`**`home`** (*dir*)
(5) Base directory for all I/O

home is the root directory for all input/output operations. All other path parameters that are relative paths will be relative to this directory.

default = `./`

.scan.tags

.scan.**tags** (*str*)
(24) Comma seperated string tags describing the data input

[deprecated?]

default = None

.scan.if_conflict_use_meta

.scan.**if_conflict_use_meta** (*bool*)
(25) Give priority to metadata relative to input parameters

[deprecated, use *scan.geometry.precedence* insteead]

default = TRUE

.scan.data

.scan.**data** (*Param*)
(26) Data preparation parameters

default = None

.scan.data.**recipe** (*ext*)
(27) Data preparation recipe container

default = None

.scan.data.**source** (*str*)
(28) Describes where to get the data from.

Accepted values are:

- 'file': data will be read from a .ptyd file.
- any valid recipe name: data will be prepared using the recipe.
- 'sim' : data will be simulated according to parameters in simulation

default = None

.scan.data.**dfile** (*str*)
(29) Prepared data file path

If source was None or 'file', data will be loaded from this file and processing as well as saving is deactivated. If source is the name of an experiment recipe or path to a file, data will be saved to this file

default = None

.scan.data.**label** (*str*)
(30) The scan label

Unique string identifying the scan

default = None

.scan.data.**shape** (*int*, *tuple*)
(31) Shape of the region of interest cropped from the raw data.

Cropping dimension of the diffraction frame Can be None, (dimx, dimy), or dim. In the latter case shape will be (dim, dim).

default = None

.scan.data.**save** (*str*)
(32) Saving mode

.io.interaction.**connections** (*int*)
(15) Number of concurrent connections on the server

A range [port : port+connections] of ports adjacent *port* will be opened on demand for connecting clients.

default = 10

.io.autoplot

.io.**autoplot** (*Param*)
(16) Plotting client parameters

In script you may set this parameter to None or False for no automatic plotting.

default = None

.io.autoplot.**imfile** (*str*)
(17) Plot images file name (or format string)

default = plots/%(run)s/%(run)s_%(engine)s_%(iterations)04d.png

.io.autoplot.**interval** (*int*)
(18) Number of iterations between plot updates

Requests to the server will happen with this iteration intervals. Note that this will work only if interaction.polling_interval is smaller or equal to this number. If interval =0 plotting is disabled which should be used, when ptypy is run on a cluster.

default = 1 (>-1)

.io.autoplot.**threaded** (*bool*)
(19) Live plotting switch

If True, a plotting client will be spawned in a new thread and connected at initialization. If False, the master node will carry out the plotting, pausing the reconstruction. This option should be set to True when ptypy is run on an isolated workstation.

default = TRUE

.io.autoplot.**layout** (*str*, *Param*)
(20) Options for default plotter or template name

Flexible layout for default plotter is not implemented yet. Please choose one of the templates 'default','‘black_and_white','nearfield', ``'minimal' or 'weak'

default = None

.io.autoplot.**dump** (*bool*)
(21) Switch to dump plots as image files

default = FALSE

.io.autoplot.**make_movie** (*bool*)
(22) Produce reconstruction movie after the reconstruction.

Switch to request the production of a movie from the dumped plots at the end of the reconstruction.

default = FALSE

1.4.6 .scan

.scan (*Param*)
(23) Scan parameters

This categrogy specifies defaults for all scans. Scan-specific parameters are stored in scans.scan_%%

default = None

default = None

.scan.data.**chunk_format** (*str*)
(41) Appendix to saved files if save == 'link'

default = .chunk%02d

.scan.data.**auto_center** (*bool*)
(42) Determine if center in data is calculated automatically

- False, no automatic centering
- None, only if center is None
- True, it will be enforced

default = None

.scan.data.**load_parallel** (*str*)
(43) Determines what will be loaded in parallel

Choose from None, 'data', 'common', 'all'

default = data

.scan.data.**positions_theory** (*ndarray*)
(44) Theoretical positions for this scan

If provided, experimental positions from *PtyScan* subclass will be ignored. If data preparation is called from Ptycho instance, the calculated positions from the *ptypy.core.xy.from_pars()* dict will be inserted here

default = None

.scan.data.**experimentID** (*str*)
(45) Name of the experiment

If None, a default value will be provided by the recipe. **unused**

default = None

.scan.sharing

.scan.**sharing** (*Param*)
(46) Scan sharing options

default = None

.scan.sharing.**object_share_with** (*str*)
(47) Label or index of scan to share object with.

Possible values:

- None: Do not share
- *(string)*: Label of the scan to share with
- *(int)*: Index of scan to share with

default = None

.scan.sharing.**object_share_power** (*float*)
(48) Relative power for object sharing

default = 1 (>0.0)

.scan.sharing.**probe_share_with** (*str*)
(49) Label or index of scan to share probe with.

Possible values:

Mode to use to save data to file.

- None: No saving
- 'merge': attemts to merge data in single chunk **[not implemented]**
- 'append': appends each chunk in master *.ptyd file
- 'link': appends external links in master *.ptyd file and stores chunks separately in the path given by the link. Links file paths are relative to master file.

default = None

.scan.data.**center** (*tuple*)
(33) Center (pixel) of the optical axes in raw data

If None, this parameter will be set by *auto_center* or elsewhere

default = None

.scan.data.**psize** (*float, tuple*)
(34) Detector pixel size

Dimensions of the detector pixels (in meters)

default = None (>0.0)

.scan.data.**distance** (*float*)
(35) Sample-to-detector distance

In meters.

default = None (>0.0)

.scan.data.**rebin** (*int*)
(36) Rebinning factor

Rebinning factor for the raw data frames. 'None' or 1 both mean *no binning*

default = None (>1, <8)

.scan.data.**orientation** (*int, tuple*)
(37) Data frame orientation

- None or 0: correct orientation
- 1: invert columns (numpy.flip_lr)
- 2: invert columns, invert rows
- 3: invert rows (numpy.flip_ud)
- 4: transpose (numpy.transpose)
- 4+i: tranpose + other operations from above

Alternatively, a 3-tuple of booleans may be provided (do_transpose, do_flipud, do_fliplr)

default = None

.scan.data.**energy** (*float*)
(38) Photon energy of the incident radiation

default = None (>0.0)

.scan.data.**min_frames** (*int*)
(39) Minimum number of frames loaded by each node

default = 1

.scan.data.**num_frames** (*int*)
(40) Maximum number of frames to be prepared

If *positions_theory* are provided, num_frames will be ovverriden with the number of positions available

.scan.geometry.**propagation** (*str*)
(58) Propagation type

Either “farfield” or “nearfield”

default = farfield

.scan.xy

.scan.**xy** (*Param*)
(59) Parameters for scan patterns

These parameters are useful in two cases:

- When the experimental positions are not known (no encoders)
- When using the package to simulate data.

In script an array of shape *(N,2)* may be passed here instead of a Param or dictionary as an **override**

default = None

.scan.xy.**model** (*str*)
(60) Scan pattern type

The type must be one of the following:

- None: positions are read from data file.
- 'raster': raster grid pattern
- 'round': concentric circles pattern
- 'spiral': spiral pattern

In script an array of shape *(N,2)* may be passed here instead

default = None (>0.0)

.scan.xy.**spacing** (*float*, *tuple*)
(61) Pattern spacing

Spacing between scan positions. If the model supports asymmetric scans, a tuple passed here will be interpreted as *(dy,dx)* with *dx* as horizontal spacing and *dy* as vertical spacing. If None the value is calculated from *extent* and *steps*

default = 1.50E-06 (>0.0)

.scan.xy.**steps** (*int*, *tuple*)
(62) Pattern step count

Number of steps with length *spacing* in the grid. A tuple *(ny,nx)* provided here can be used for a different step in vertical (*ny*) and horizontal direction (*nx*). If None the, step count is calculated from *extent* and *spacing*

default = 10 (>0)

.scan.xy.**extent** (*float*, *tuple*)
(63) Rectangular extent of pattern

Defines the absolut maximum extent. If a tuple *(ly,lx)* is provided the extent may be rectangular rather than square. All positions outside of *extent* will be discarded. If None the extent will is *spacing* times *steps*

default = 1.50E-05 (>0.0)

.scan.xy.**offset** (*float*, *tuple*)
(64) Offset of scan pattern relative to origin

If tuple, the offset may differ in *x* and *y*. Please not that the offset will be included when removing scan points outside of *extend*.

- None: Do not share
- *(string)*: Label of the scan to share with
- *(int)*: Index of scan to share with

default = None

.scan.sharing.**probe_share_power** (*float*)
(50) Relative power for probe sharing

default = 1 (>0.0)

.scan.geometry

.scan.**geometry** (*Param*)
(51) Physical parameters

All distances are in meters. Other units are specified in the documentation strings. These paramters have very low priority in the *Ptycho* construction process and can usually left out in script if either *scan.data* ot the invoked preparation subclass provide enough geometric information. You can change this behavior with the *precedence* parameter.

default = None

.scan.geometry.**precedence** (*str*)
(52) Where geometry parameters take precence over others

Possible options if parameters are not None:

- None: Fill only missing parameters (default) at the very last moment making meta data from *PtyScan* the default source of geometric information.
- 'meta': Overwrite meta after data was loaded, does not affect data preparation.
- 'data': Overwrite entries in *scan.data*. This affects data preparation too.

default = None

.scan.geometry.**energy** (*float*)
(53) Energy (in keV)

If None, uses *lam* instead.

default = 6.2 (>0.0)

.scan.geometry.**lam** (*float*)
(54) Wavelength

Used only if *energy* is None

default = None (>0.0)

.scan.geometry.**distance** (*float*)
(55) Distance from object to detector

default = 7.19 (>0.0)

.scan.geometry.**psize** (*float*)
(56) Pixel size in Detector plane

default = 0.000172 (>0.0)

.scan.geometry.**resolution** (*float*)
(57) Pixel size in Sample plane

This parameter is used only for simulations

default = None (>0.0)

.scan.illumination.recon.**layer** (*float*)
(73) Layer (mode) of storage data to load

`None` means all layers, choose `0` for main mode

default = `None`

.scan.illumination.**stxm** (*Param*)
(74) Parameters to initialize illumination from diffraction data

default = `None`

.scan.illumination.stxm.**label** (*str*)
(75) Scan label of diffraction that is to be used for probe estimate

`None`, own scan label is used

default = `None`

.scan.illumination.**aperture** (*Param*)
(76) Beam aperture parameters

default = `None`

.scan.illumination.aperture.**form** (*str*)
(77) One of None, 'rect' or 'circ'

One of:

- `None` : no aperture, this may be useful for nearfield
- `'rect'` : rectangular aperture
- `'circ'` : circular aperture

default = `circ (>0.0)`

.scan.illumination.aperture.**diffuser** (*float*)
(78) Noise in the transparen part of the aperture

Can be either:

- `None` : no noise
- `2-tuple` : noise in phase (amplitude (rms), minimum feature size)
- `4-tuple` : noise in phase & modulus (rms, mfs, rms_mod, mfs_mod)

default = `None (>0.0)`

.scan.illumination.aperture.**size** (*float*)
(79) Aperture width or diameter

May also be a tuple *(vertical,horizontal)* in case of an asymmetric aperture

default = `None (>0.0)`

.scan.illumination.aperture.**edge** (*int*)
(80) Edge width of aperture (in pixels!)

default = `2 (>0)`

.scan.illumination.aperture.**central_stop** (*float*)
(81) size of central stop as a fraction of aperture.size

If not None: places a central beam stop in aperture. The value given here is the fraction of the beam stop compared to *size*

default = `None (>0.0, <1.0)`

.scan.illumination.aperture.**offset** (*float, tuple*)
(82) Offset between center of aperture and optical axes

May also be a tuple (vertical,horizontal) for size in case of an asymmetric offset

default = `0`

.scan.xy.**jitter** (*float, tuple*)
(65) RMS of jitter on sample position

Only use in simulation. Adds a random jitter to positions.

default = `0`

.scan.xy.**count** (*int*)
(66) Number of scan points

Only return return positions up to number of *count*.

default = `None`

.scan.illumination

.scan.**illumination** (*Param*)
(67) Illumination model (probe)

In script, you may pass directly a three dimensional numpy.ndarray here instead of a *Param*. This array will be copied to the storage instance with no checking whatsoever. Used in *~ptypy.core.illumination*

default = `None (>0.0)`

.scan.illumination.**model** (*str*)
(68) Type of illumination model

One of:

- `None` : model initialitziation defaults to flat array filled with the specified number of photons
- `'recon'` : load model from previous reconstruction, see *recon* Parameters
- `'stxm'` : Estimate model from autocorrelation of mean diffraction data
- <*resource*> : one of ptypys internal image resource strings
- <*template*> : one of the templates inillumination module

In script, you may pass a numpy.ndarray here directly as the model. It is considered as incoming wavefront and will be propagated according to *propagation* with an optional *aperture* applied before

default = `None`

.scan.illumination.**photons** (*int*)
(69) Number of photons in the incident illumination

A value specified here will take precedence over calculated statistics from the loaded data.

default = `None (>0)`

.scan.illumination.**recon** (*Param*)
(70) Parameters to load from previous reconstruction

default = `None`

.scan.illumination.recon.**rfile** (*str*)
(71) Path to a `.ptyr` compatible file

default = `\*.ptyr`

.scan.illumination.recon.**ID** (*NoneType*)
(72) ID (label) of storage data to load

`None` means any ID

default = `None`

.scan.sample

`.scan.`**`sample`** (*Param*)
(92) Initial object modelization parameters

In script, you may pass a numpy.array here directly as the model. This array will be passed to the storage instance with no checking whatsoever. Used in *~ptypy.core.sample*

default = `None (>0.0)`

`.scan.sample.`**`model`** (*str*)
(93) Type of initial object model

One of:

- `None` : model initialitziation defaults to flat array filled *fill*
- `'recon'` : load model from STXM analysis of diffraction data
- `'stxm'` : Estimate model from autocorrelation of mean diffraction data
- **<***resource***>** : one of ptypys internal model resource strings
- **<***template***>** : one of the templates in sample module

In script, you may pass a numpy.array here directly as the model. This array will be processed according to *process* in order to *simulate* a sample from e.g. a thickness profile.

default = `None`

`.scan.sample.`**`fill`** (*float*, *complex*)
(94) Default fill value

default = `1`

`.scan.sample.`**`recon`** (*Param*)
(95) Parameters to load from previous reconstruction

default = `None`

`.scan.sample.recon.`**`rfile`** (*str*)
(96) Path to a `.ptyr` compatible file

default = `\*.ptyr`

`.scan.sample.recon.`**`ID`** (*NoneType*)
(97) ID (label) of storage data to load

`None` is any ID

default = `None`

`.scan.sample.recon.`**`layer`** (*float*)
(98) Layer (mode) of storage data to load

`None` is all layers, choose `0` for main mode

default = `None`

`.scan.sample.`**`stxm`** (*Param*)
(99) STXM analysis parameters

default = `None`

`.scan.sample.stxm.`**`label`** (*str*)
(100) Scan label of diffraction that is to be used for probe estimate

`None`, own scan label is used

default = `None`

default = `0`

`.scan.illumination.`**`propagation`** (*Param*)
(83) Parameters for propagation after aperture plane

Propagation to focus takes precedence to parallel propagation if *foccused* is not `None`

default = `None`

`.scan.illumination.propagation.`**`parallel`** (*float*)
(84) Parallel propagation distance

If `None` or `0` : No parallel propagation

default = `None`

`.scan.illumination.propagation.`**`focussed`** (*float*)
(85) Propagation distance from aperture to focus

If `None` or `0` : No focus propagation

default = `None`

`.scan.illumination.propagation.`**`antialiasing`** (*float*)
(86) Antialiasing factor

Antialiasing factor used when generating the probe. (numbers larger than 2 or 3 are memory hungry) **[Untested]**

default = `1`

`.scan.illumination.propagation.`**`spot_size`** (*float*)
(87) Focal spot diameter

If not `None`, this parameter is used to generate the appropriate aperture size instead of `size`

default = `None (>0.0)`

`.scan.illumination.`**`diversity`** (*Param*)
(88) Probe mode(s) diversity parameters

Can be `None` i.e. no diversity

default = `None`

`.scan.illumination.diversity.`**`noise`** (*tuple*)
(89) Noise in the generated modes of the illumination

Can be either:

- `None` : no noise
- `2-tuple` : noise in phase (amplitude (rms), minimum feature size)
- `4-tuple` : noise in phase & modulus (rms, mfs, rms_mod, mfs_mod)

default = `None`

`.scan.illumination.diversity.`**`power`** (*tuple*, *float*)
(90) Power of modes relative to main mode (zero-layer)

default = `0.1`

`.scan.illumination.diversity.`**`shift`** (*float*)
(91) Lateral shift of modes relative to main mode

[not implemented]

default = `None`

.scan.sample.diversity.**power** (*tuple*, *float*)
(111) Power of modes relative to main mode (zero-layer)

default = 0.1

.scan.sample.diversity.**shift** (*float*)
(112) Lateral shift of modes relative to main mode

[not implemented]

default = None

.scan.coherence

.scan.**coherence** (*Param*)
(113) Coherence parameters

default = None (>0.0)

.scan.coherence.**num_probe_modes** (*int*)
(114) Number of probe modes

default = 1 (>0)

.scan.coherence.**num_object_modes** (*int*)
(115) Number of object modes

default = 1 (>0)

.scan.coherence.**spectrum** (*list*)
(116) Amplitude of relative energy bins if the probe modes have a different energy

default = None (>0.0)

.scan.coherence.**object_dispersion** (*str*)
(117) Energy dispersive response of the object

One of:

- None or 'achromatic': no dispersion
- 'linear': linear response model
- 'irregular': no assumption

[not implemented]

default = None

.scan.coherence.**probe_dispersion** (*str*)
(118) Energy dispersive response of the probe

One of:

- None or 'achromatic': no dispersion
- 'linear': linear response model
- 'irregular': no assumption

[not implemented]

default = None

1.4.7 .scans

.**scans** (*Param*)
(119) Param container for instances of *scan* parameters

If not specified otherwise, entries in *scans* will use parameter defaults from *scan*

.scan.sample.**process** (*Param*)
(101) Model processing parameters

Can be None, i.e. no processing

default = None

.scan.sample.process.**offset** (*tuple*)
(102) Offset between center of object array and scan pattern

default = (0,0) (>0.0)

.scan.sample.process.**zoom** (*tuple*)
(103) Zoom value for object simulation.

If None, leave the array untouched. Otherwise the modeled or loaded image will be resized using zoom().

default = None (>0.0)

.scan.sample.process.**formula** (*str*)
(104) Chemical formula

A Formula compatible with a cxro database query,e.g. 'Au' or 'NaCl' or 'H2O'

default = None

.scan.sample.process.**density** (*float*)
(105) Density in [g/ccm]

Only used if *formula* is not None

default = 1

.scan.sample.process.**thickness** (*float*)
(106) Maximum thickness of sample

If None, the absolute values of loaded source array will be used

default = 1.00E-06

.scan.sample.process.**ref_index** (*complex*)
(107) Assigned refractive index

If None, treat source array as projection of refractive index. If a refractive index is provided the array's absolute value will be used to scale the refractive index.

default = 0.5+0.j (>0.0)

.scan.sample.process.**smoothing** (*int*)
(108) Smoothing scale

Smooth the projection with gaussian kernel of width given by *smoothing_mfs*

default = 2 (>0)

.scan.sample.**diversity** (*Param*)
(109) Probe mode(s) diversity parameters

Can be None i.e. no diversity

default = None

.scan.sample.diversity.**noise** (*tuple*)
(110) Noise in the generated modes of the illumination

Can be either:

- None : no noise
- 2-tuple : noise in phase (amplitude (rms), minimum feature size)
- 4-tuple : noise in phase & modulus (rms, mfs, rms_mod, mfs_mod)

default = None

.engine.DM.**alpha** (*int*)
(129) Difference map parameter

default = 1 (>0)

.engine.DM.**probe_update_start** (*int*)
(130) Number of iterations before probe update starts

default = 2 (>0)

.engine.DM.**update_object_first** (*bool*)
(131) If False update object before probe

default = TRUE (>0.0)

.engine.DM.**overlap_converge_factor** (*float*)
(132) Threshold for interruption of the inner overlap loop

The inner overlap loop refines the probe and the object simultaneously. This loop is escaped as soon as the overall change in probe, relative to the first iteration, is less than this value.

default = 0.05 (>0.0)

.engine.DM.**overlap_max_iterations** (*int*)
(133) Maximum of iterations for the overlap constraint inner loop

default = 10 (>0)

.engine.DM.**probe_inertia** (*float*)
(134) Weight of the current probe estimate in the update

default = 0.001 (>0.0)

.engine.DM.**object_inertia** (*float*)
(135) Weight of the current object in the update

default = 0.1 (>0.0)

.engine.DM.**fourier_relax_factor** (*float*)
(136) If rms error of model vs diffraction data is smaller than this fraction, Fourier constraint is met

Set this value higher for noisy data

default = 0.01 (>0.0)

.engine.DM.**obj_smooth_std** (*int*)
(137) Gaussian smoothing (pixel) of the current object prior to update

If None, smoothing is deactivated. This smoothing can be used to reduce the amplitude of spurious pixels in the outer, least constrained areas of the object.

default = 20 (>0)

.engine.ML

.engine.**ML** (*Param*)
(138) Maximum Likelihood parameters

default = None

.engine.ML.**type** (*str*)
(139) Likelihood model. One of 'gaussian', 'poisson' or 'euclid'

[only 'gaussian' is implemented for now]

default = gaussian (>0.0)

default = None

.scans.scan_00

.scans.**scan_00** (*scan*)
(120) Default first scans entry

If only a single scan is used in the reconstruction, this entry may be left unchanged. If more than one scan is used, please make an entry for each scan. The name *scan_00* is an arbitrary choice and may be set to any other string.

default = None

1.4.8 .engine

.engine (*Param*)
(121) Reconstruction engine parameters

default = None

.engine.common

.engine.**common** (*Param*)
(122) Parameters common to all engines

default = None

.engine.common.**name** (*str*)
(123) Name of engine.

Dependent on the name given here, the default parameter set will be a superset of *common* and parameters to the entry of the same name.

default = DM

.engine.common.**numiter** (*int*)
(124) Total number of iterations

default = 20 (>0)

.engine.common.**numiter_contiguous** (*int*)
(125) Number of iterations without interruption

The engine will not return control to the caller until this number of iterations is completed (not processing server requests, I/O operations, ...)

default = 1 (>0)

.engine.common.**probe_support** (*float*)
(126) Fraction of valid probe area (circular) in probe frame

default = 0.7 (>0.0)

.engine.common.**clip_object** (*tuple*)
(127) Clip object amplitude into this intrervall

default = None (>0.0)

.engine.DM

.engine.**DM** (*Param*)
(128) Parameters for Difference map engine

default = None

1.5 ptypy package

1.5.1 Subpackages

ptypy.core package

Submodules

ptypy.core.classes module

Container management.

This module defines flexible containers for the various quantities needed for ptychographic reconstructions.

Container class A high-level container that keeps track of sub-containers (Storage) and Views onto them. A container can copy itself to produce a buffer needed for calculations. Some basic Mathematical operations are implemented at this level as in place operations. In General, operations on arrays should be done using the Views, which simply return numpyarrays.

Storage class The sub-container, wrapping a numpy array buffer. A Storage defines a system of coordinate (for now only a scaled translation of the pixel coordinates, but more complicated affine transformation could be implemented if needed). The sub-class DynamicStorage can adapt the size of its buffer (cropping and/or padding) depending on the Views.

View class A low-weight class that contains all information to access a 2D piece of a Storage within a Container. The basic idea is that the View access is controlled by a physical position and its frame, such that one is not bothered by memory/array addresses when accessing data.

This file is part of the PTYPY package.

> **copyright** Copyright 2014 by the PTYPY team, see AUTHORS.
>
> **license** GPLv2, see LICENSE for details.

class ptypy.core.classes.**Container**(*ptycho=None*, *ID=None*, *data_type='complex'*, ***kwargs*)

Bases: *ptypy.core.classes.Base*

High-level container class.

Container can be seen as a "super-numpy-array" which can contain multiple sub-containers of type *Storage*, potentially of different shape, along with all *View* instances that act on these Storages to extract data from the internal data buffer *Storage.data*.

Typically there will be five such base containers in a *Ptycho* reconstruction instance:

- *Cprobe*, Storages for the illumination, i.e. **probe**,
- *Cobj*, Storages for the sample transmission, i.e. **object**,
- *Cexit*, Storages for the **exit waves**,
- *Cdiff*, Storages for **diffraction data**, usually one per scan,
- *Cmask*, Strorages for **masks** (and weights), usually one per scan,

A container can conveniently duplicate all its internal *Storage* instances into a new Container using *copy()*. This feature is intensively used in the reconstruction engines where buffer copies are needed to temporarily store results. These copies are referred by the "original" container through the property *copies()* and a copy refers to its original through the attribute `original` In order to reduce the number of *View* instances, Container copies do not hold views and use instead the Views held by the original container

Variables

.engine.ML.**floating_intensities**(*bool*)
(140) If True, allow for adaptative rescaling of the diffraction pattern intensities (to correct for incident beam intensity fluctuations).

default = FALSE

.engine.ML.**intensity_renormalization**(*float*)
(141) A rescaling of the intensity so they can be interpreted as Poisson counts.

default = 1 (>0.0)

.engine.ML.**reg_del2**(*bool*)
(142) Whether to use a Gaussian prior (smoothing) regularizer.

default = TRUE (>0.0)

.engine.ML.**reg_del2_amplitude**(*float*)
(143) Amplitude of the Gaussian prior if used.

default = 0.01 (>0.0)

.engine.ML.**smooth_gradient**(*float*)
(144) Smoothing preconditioner. If 0, not used, if > 0 gaussian filter if < 0 Hann window.

default = 0 (>0.0)

.engine.ML.**scale_precond**(*bool*)
(145) Whether to use the object/probe scaling preconditioner.

This parameter can give faster convergence for weakly scattering samples.

default = FALSE (>0.0)

.engine.ML.**scale_probe_object**(*float*)
(146) Relative scale of probe to object.

default = 1 (>0.0)

.engine.ML.**probe_update_start**(*int*)
(147) Number of iterations before probe update starts

default = 0

1.4.9 .engines

.engines(*Param*)
(148) Container for instances of "engine" parameters

All engines registered in this structure will be executed sequentially.

default = None

.engines.engine_00

.engines.**engine_00**(*engine*)
(149) Default first engines entry

Default first engine is difference map (DM)

default = None

dtype
Property that returns numpy dtype of all internal data buffers

fill(*fill=0.0*)
Fill all storages with scalar value *fill*

formatted_report(*table_format=None*, *offset=8*, *align='right'*, *separator=' : '*, *include_header=True*)
Returns formatted string and a dict with the respective information

Parameters

- **table_format** (*list*) – List of (*item*,*length*) pairs where item is name of the info to be listed in the report and length is the column width. The following items are allowed:
  - *memory*, for memory usage of the storages and total use
  - *shape*, for shape of internal storages
  - *dimensions*, is `shape \* psize`
  - *psize*, for pixel size of storages
  - *views*, for number of views in each storage
- **offset** (*int, optional*) – First column width
- **separator** (*str, optional*) – Column separator
- **align** (*str, optional*) – Column alignment, either `'right'` or `'left'`
- **include_header** (*bool*) – Include a header if True

Returns

- **fstring** (*str*) – Formatted string
- **dct** (*dict*) – Dictionary containing with the respective info to the keys in *table_format*

See also:

Storage.formatted_report()

info()
Return the total buffer space for this container in bytes and storage info

Returns

- **space** (*int*) – Accumulated memory usage of all data buffers in this Container
- **fstring** (*str*) – Formatted string

Note: May get **deprecated** in future. Use formatted_report instead.

See also:

report(), *formatted_report()*

nbytes
Return total number of bytes used by numpy array buffers in this container. This is not the actual size in memory of the whole contianer, as it does not include the views nor dictionary overhead.

new_storage(*ID=None*, ***kwargs*)
Create and register a storage object.

Parameters

- **ID** (*str*) – An ID for the storage. If None, a new ID is created. An error will be raised if the ID already exists.
- **kwargs** – Arguments for new storage creation. See doc for *Storage*.

- **original** (Container) – If self is copy of a Container, this attribute refers to the original Container. Otherwise it is None.
- ***data_type*** (*str*) – Either "single" or "double"

Parameters

- **ID** (*str or int*) – A unique ID, managed by the parent
- **ptycho** (Ptycho) – The instance of Ptycho associated with this pod.
- **data_type** (*str or numpy.dtype*) – data type - either a numpy.dtype object or 'complex' or 'real' (precision is taken from ptycho.FType or ptycho.CType)

S
A property that returns the internal dictionary of all *Storage* instances in this *Container*

Sp
A property that returns the internal dictionary of all *Storage* instances in this *Container* as a *Param*

V
A property that returns the internal dictionary of all *View* instances in this *Container*

Vp
A property that returns the internal dictionary of all *View* instances in this *Container* as a *Param*

__getitem__(*view*)
Access content through view.

Parameters **view** (View) – A valid *View* object.

__setitem__(*view*, *newdata*)
Set content given by view.

Parameters

- **view** (View) – A valid *View* for this object
- **newdata** (*array_like*) – The data to be stored 2D.

allreduce(*op=None*)
Performs MPI parallel `allreduce` with a sum as reduction for all *Storage* instances held by *self*

Parameters

- **c** (Container) – Input
- **op** – Reduction operation. If `None` uses sum.

See also:

ptypy.utils.parallel.allreduce(), *Storage.allreduce()*

clear()
Reduce / delete all data in attached storages

copies
Property that returns list of all copies of this *Container*

copy(*ID=None*, *fill=None*)
Create a new *Container* matching self.

The copy does not manage views.

Parameters **fill** (*scalar or None*) – If None (default), copy content. If scalar, initializes to this value

delete_copy(*copyIDs=None*)
Delet a copy or all copies of this container from owner instance.

Parameters **copyIDS** (*str*) – ID of copy to be deleted. If None, deletes *all* copies

Returns The view to internal data buffe correspondieng to View *v*

Return type ndarray

__setitem__(*v*, *newdata*)
Storage[v] = newdata

Set internal data buffer to *newdata* for the region of view *v*.

Parameters

- **v** (View) – A View for this storage
- **newdata** (*ndarray*) – Two-dimensional array that fits the view's shape

allreduce(*op=None*)
Performs MPI parallel `allreduce` with a default sum as reduction operation for internal data buffer `self.data`

Parameters

- **c** (Container) – Input
- **op** – Reduction operation. If `None` uses sum.

See also:

ptypy.utils.parallel.allreduce(), *Container.allreduce()*

center
Return the position of the origin relative to the upper-left corner of the storage, in pixel coordinates

copy(*owner=None*, *ID=None*, *fill=None*)
Return a copy of this storage object.

Note: the returned copy has the same container as self.

Parameters

- **ID** (*str or int*) – A unique ID, managed by the parent
- **fill** (*scalar or None*) – If float, set the content to this value. If None, copy the current content.

data = None
Three dimensional array as data buffer

dtype

fill(*fill=None*)
Fill managed buffer.

Parameters **fill** (*scalar, numpy array or None.*) – Fill value to use. If fill is a numpy array, it is cast as self.dtype and self.shape is updated to reflect the new buffer shape. If fill is None, use default value (self.fill_value).

fill_value = None
Default fill value

formatted_report(*table_format=None*, *offset=8*, *align='right'*, *separator=' : '*, *include_header=True*)
Returns formatted string and a dict with the respective information

Parameters

- **table_format** (*list, optional*) – List of (*item*,*length*) pairs where item is name of the info to be listed in the report and length is the column width. The following items are allowed:
  - *memory*, for memory usage of the storages and total use
  - *shape*, for shape of internal storages

reformat(*AlsoInCopies=False*)
Reformats all storages in this container.

Parameters **AlsoInCopies** (*bool*) – If True, also reformat associated copies of this container

report()
Returns a formatted string giving a report on all storages in this container.

size
Return total number of pixels in this container.

storages
A property that returns the internal dictionary of all *Storage* instances in this *Container*

views
A property that returns the internal dictionary of all *View* instances in this *Container*

views_in_storage(*s*, *active=True*)
Return a list of views on *Storage* *s*.

Parameters

- **s** (Storage) – The storage to look for.
- **active** (*True or False*) – If True (default), return only active views.

class ptypy.core.classes.**Storage**(*container*, *ID=None*, *data=None*, *shape=(1, 1, 1)*, *fill=0.0*, *psize=None*, *origin=None*, *layermap=None*, *padonly=False*, ***kwargs*)
Bases: *ptypy.core.classes.Base*

Inner container handling acces to data arrays.

This class essentially manages access to an internal numpy array buffer called *data*

- It returns a view to coordinates given (slicing)
- It contains a physical coordinate grid

Parameters

- **container** (Container) – The container instance
- **ID** (*None,str or int*) – A unique ID, managed by the parent, if None ID is generated by parent.
- **data** (*ndarray or None*) – A numpy array to use as initial buffer.
- **shape** (*3-tuple*) – The shape of the buffer
- **fill** (*float or complex*) – The default value to fill storage with, will be converted to data type of owner.
- **psize** (*float or 2-tuple of float*) – The physical pixel size.
- **origin** (*2-tuple of int*) – The physical coordinates of the [0,0] pixel (upper-left corner of the storage buffer).
- **layermap** (*list or None*) – A list (or 1D numpy array) mapping input layer indices to the internal buffer. This may be useful if the buffer contains only a portion of a larger dataset (as when using distributed data with MPI). If None, provide direct access. to the 3d internal data.
- **padonly** (*bool*) – If True, reformat() will enlarge the internal buffer if needed, but will not shrink it.

__getitem__(*v*)
Storage[v]

zoom_to_psize (*new_psize*, ***kwargs*)
Changes pixel size and zooms the data buffer along last two axis accordingly, updates all attached views and reformats if neccessary. **untested!!**

Parameters **new_psize** (*scalar or array_like*) – new pixel size

class ptypy.core.classes.**View** (*container*, *ID=None*, *accessrule=None*, ***kwargs*)
Bases: *ptypy.core.classes.Base*

A "window" on a Container.

A view stores all the slicing information to extract a 2D piece of Container.

Note: The final structure of this class is yet up to debate and the constructor signature may change. Especially since "DEFAULT_ACCESSRULE" is yet so small, its contents could be incorporated in the constructor call.

Parameters

- **container** (Container) – The Container instance this view applies to.
- **ID** (*str or int*) – ID for this view. Automatically built from ID if None.
- **accessrule** (*dict*) – All the information necessary to access the wanted slice. Maybe subject to change as code evolve. See keyword arguments Almost all keys of accessrule will be available as attributes in the constructed View instance.

Keyword Arguments

- **storageID** (*str*) – ID of storage, If the Storage does not exist it will be created! (*default* is None)
- **shape** (*int or tuple of int*) – Shape of the view in pixels (*default* is None)
- **coord** (*2-tuple of float,*) – Physical coordinates [meter] of the center of the view.
- **psize** (*float or tuple of float*) – Pixel size [meters]. Required for storage initialization, See DEFAULT_PSIZE
- **layer** (*int*) – Index of the third dimension if applicable. (*default* is 0)
- **active** (*bool*) – Whether this view is active (*default* is True)

DEFAULT_ACCESSRULE = Param({'layer': 0, 'coord': None, 'psize': 1.0, 'shape': None, 'storageID': None

active = None
Active state. If False this view will be ignored when resizing the data buffer of the associated *Storage*.

coord
The View's physical coordinate (meters)

data
The view content in data buffer of associated storage.

dcoord
Center coordinate (index) in data buffer.

dhigh
High side of the View's data range.

dlayer = None
The "layer" i.e. first axis index in Storage data buffer

dlow
Low side of the View's data range.

– *dimensions*, is shape * psize
– *psize*, for pixel size of storages
– *views*, for number of views in each storage

- **offset** (*int, optional*) – First column width
- **separator** (*str, optional*) – Column separator.
- **align** (*str, optional*) – Column alignment, either 'right' or 'left'.
- **include_header** (*bool, optional*) – Include a header if True.

Returns

- **fstring** (*str*) – Formatted string
- **dct** (*dict*) – Dictionary containing with the respective info to the keys in *table_format*

get_view_coverage()
Creates an array in the shape of internal buffer where the value of each pixel represents the number of views that cover that pixel

Returns view coverage in the shape of internal buffer

Return type ndarray

grids()

Returns **x, y** – grids in the shape of internal buffer

Return type ndarray

origin
Return the physical position of the upper-left corner of the storage.

psize

Returns The pixel size.

reformat(*newID=None*)
Crop or pad if required.

Parameters **newID** (*str or int*) – If None (default) act on self. Otherwise create a copy of self before doing the cropping and padding.

Returns **s** – returns new Storage instance in same *Container* if *newId* is not None.

Return type *Storage*

report()

Returns a formatted string giving a report on this storage.

Return type str

update()
Update internal state, including all views on this storage to ensure consistency with the physical coordinate system.

update_views(*v=None*)
Update the access information for a given view.

Parameters **v** (*View or None*) – The view object to update. If None, loop through all views. Apart from that, no check is done, not even whether the view is actually on self. Use cautiously.

views
Return all the views that refer to this storage.

geometry = None
Geo instance with propagators

mask
Convenience property that links to slice of masking *Storage*. Equivalent to `self.ma_view.data`.

object
Convenience property that links to slice of object *Storage*. Usually equivalent to `self.ob_view.data`.

pr_view = None
A reference to the (pr)obe-view. (ob)ject-, (ma)sk- and (di)ff-view are accessible in the same manner (`self.xx_view`).

probe
Convenience property that links to slice of probe *Storage*. Equivalent to `self.pr_view.data`.

class `ptypy.core.classes.`**Base**(*owner=None*, *ID=None*, *BeOwner=True*)
Bases: `object`

Ptypy Base class to support some kind of hierarchy, conversion to and from dictionarys as well as a 'cross-node' ID managament of python objects

Parameters

- **owner** (*Other subclass of Base or Base*) – Owner gives IDs to other ptypy objects that refer to him as owner. Owner also keeps a reference to these objects in its internal _pool where objects are key-sorted according to their ID prfix
- **ID** (*None, str or int*) –
- **BeOwner** (*bool*) – Set to *False* if this instance is not intended to own other ptypy objects.

calc_mem_usage()

ptypy.core.data module

data - Diffraction data access

This module defines a PtyScan, a container to hold the experimental data of a ptychography scan. Instrument-specific reduction routines should inherit PtyScan to prepare data for the Ptycho Instance in a uniform format.

The experiment specific child class only needs to overwrite 2 functions of the base class:

For the moment the module contains two main objects: PtyScan, which holds a single ptychography scan, and DataSource, which holds a collection of datascans and feeds the data as required.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

`ptypy.core.data.`**GENERIC = Param({'positions_theory': None, 'auto_center': None, 'chunk_format': '.chu**
Default data parameters. See *scan.data* and a short listing below

- *positions_theory* = None, see *positions_theory*
- *auto_center* = None, see *auto_center*
- *chunk_format* = ".chunk%02d", see *chunk_format*
- *min_frames* = 1, see *min_frames*
- *orientation* = None, see *orientation*
- *num_frames* = None, see *num_frames*

pod
Returns first *POD* in the `self.pods` dict. This is a common call in the code and has therefore found its way here. May return `None` if there is no pod connected.

pods = None
Volatile dictionary for all *POD*s that connect to this view

psize
Pixel size of the View.

shape
Two dimensional shape of View.

slice
Returns a slice-tuple according to `self.layer`, `self.dlow` and `self.dhigh`. Please note, that this may not always makes sense

sp
The subpixel difference (meters) between physical coordinate and data coordinate.

storage = None
The *Storage* instance that this view applies to by default.

storageID = None
The storage ID that this view will be forward to if applied to a *Container*.

class `ptypy.core.classes.`**POD**(*ptycho=None*, *ID=None*, *views=None*, *geometry=None*, ***kwargs*)
Bases: *ptypy.core.classes.Base*

POD : Ptychographic Object Descriptor

A POD brings together probe view, object view and diff view. It also gives access to "exit", a (coherent) exit wave, and to propagation objects to go from exit to diff space.

Parameters

- **ptycho** (Ptycho) – The instance of Ptycho associated with this pod.
- **ID** (*str or int*) – The pod ID, If None it is managed by the ptycho.
- **views** (*dict or Param*) – The views. See *DEFAULT_VIEWS*.
- **geometry** (Geo) – Geometry class instance and attached propagator

DEFAULT_VIEWS = {'exit': None, 'probe': None, 'obj': None, 'mask': None, 'diff': None}
Default set of *View*s used by a POD

active
Convenience property that describes whether this pod is active or not. Equivalent to `self.di_view.active`

bw
Convenience property that returns backward propagator of attached Geometry instance. Equivalent to `self.geometry.propagator.bw`.

diff
Convenience property that links to slice of diffraction *Storage*. Equivalent to `self.di_view.data`.

exit
Convenience property that links to slice of exit wave *Storage*. Equivalent to `self.pr_view.data`.

fw
Convenience property that returns forward propagator of attached Geometry instance. Equivalent to `self.geometry.propagator.fw`.

This method is supposed to return the number of accessible frames for preparation and should determine if data acquistion for this scan is finished. Its main purpose is to allow for a data acquisition scheme, where the number of frames is not known when *PtyScan* is constructed, i.e. a data stream or an on-the-fly reconstructions.

Note: If *num_frames* is set on `__init__()` of the subclass, this method can be left as is.

Parameters

- **frames** (*int or None*) – Number of frames requested.
- **start** (*int or None*) – Scanpoint index to start checking from.

Returns

- **frames_accessible** (*int*) – Number of frames readable.
- **end_of_scan** (*int or None*) – is one of the following, - 0, end of the scan is not reached - 1, end of scan will be reached or is - None, can't say

correct (*raw*, *weights*, *common*)
Override in subclass for custom implementation

Place holder for dark and flatfield correction. If *load* already provides data in the form of photon counts, and no frame specific weight is needed, this method may be left as is

May get *merged* with *load* in future.

Returns **data,weights** – Flat and dark-corrected data dictionaries. These dictionaries must have the same keys as the input *raw* and contain corrected frames (*data*) and statistical weights (*weights*) which are zero for invalid or masked pixel other the number of detector counts that correspond to one photon count

Return type dict

end_of_scan

frames_accessible

get_data_chunk (*chunksize*, *start=None*)
This function prepares a container that is compatible to data package This function is called from the auto() function.

info = None
Param container that stores all input parameters.

initialize()
Begins the Data preparation and intended as the first method that does read-write acces on (large) data. Does the following:

- Creates a *.ptyd data file at location specified by `dfile` (master node only)
- Calls *load_weight()*, *load_positions()* *load_common()* (master node only for `load_parallel==None` or `load_parallel=='data'`)
- Sets *num_frames* if needed
- Calls *post_initialize()*

load (*indices*)
Override in subclass for custom implementation

Loads data according to node specific scanpoint indeces that have been determined by `LoadManager` or otherwise

Returns **raw, positions, weight** – Dictionaries whose keys are the given scan point *indices* and whose values are the respective frame / position according to the scan point index. *weight* and *positions* may be empty

- *energy* = `None`, see *energy*
- *center* = `None`, see *center*
- *recipe* = `dict`, see *recipe*
- *label* = `None`, see *label*
- *psize* = `None`, see *psize*
- *load_parallel* = `"data"`, see *load_parallel*
- *shape* = `None`, see *shape*
- *rebin* = `None`, see *rebin*
- *experimentID* = `None`, see *experimentID*
- *version* = `"0.1"`
- *save* = `None`, see *save*
- *dfile* = `None`, see *dfile*
- *distance* = `None`, see *distance*

class `ptypy.core.data.`**PtyScan** (*pars=None*, ***kwargs*)
Bases: `object`

PtyScan: A single ptychography scan, created on the fly or read from file.

BASECLASS

Objectives:

- Stand alone functionality
- Can produce .ptyd data formats
- Child instances should be able to prepare from raw data
- On-the-fly support in form of chunked data.
- mpi capable, child classes should not worry about mpi

Class creation with minimum set of parameters, see *GENERIC* Please note that class creation is not meant to load data.

Call *initialize* to begin loading and data file creation.

CODES = {'msgEOS': 'End of scan reached', 'msg1': 'Scan unfinished. More frames available after a pause

DEFAULT = Param({'positions_theory': None, 'auto_center': None, 'min_frames': 1, 'orientation': None, '

EOS = 'msgEOS'

WAIT = 'msg1'

abort

auto (*frames*, *chunk_form='dp'*)
Repeated calls to this function will process the data

Parameters **frames** (*int*) – Number of frames to process.

Returns one of the following - None, if scan's end is not reached, but no data could be prepared yet - False, if scan's end is reached - a data package otherwise

Return type variable

check (*frames=None*, *start=None*)
Override in subclass for custom implementation

This method checks how many frames the preparation routine may process, starting from frame *start* at a request of *frames*.

If *load_parallel* is set to *all* or common', this function is executed by all nodes, otherwise the master node executes this function and braodcasts the results to other nodes.

Returns **weight2d** – A twodimensional array with a shape compatible to the raw diffraction data frames

Return type ndarray

Note: For now, weights will be converted to a mask, `mask = weight2d > 0` for use in reconstruction algorithms. It is planned to use a general weight instead of a mask in future releases.

min_frames = None
Minimum number of frames to prepare / load with call of *auto()*

num_frames = None
Total number of frames to prepare / load. Set by *num_frames*

post_init()
Placeholder. Called at the end of construction by all processes.

post_initialize()
Placeholder. Called at the end of *initialize()* by all processes.

Use this method to benefit from 'hard-to-retrieve but now available' information after initialize.

report(*what=None*, *shout=True*)
Make a report on internal structure

return_chunk_as(*chunk*, *kind='dp'*)
Returns the loaded data chunk *chunk* in the format *kind* For now only kind=='dp' (data package) is valid.

ptypy.core.data.**PTYD = {'info': {}, 'chunks': {}, 'meta': {}}**
Basic Structure of a .ptyd datafile

class ptypy.core.data.**PtydScan**(*pars=None*, *source=None*, ***kwargs*)
Bases: *ptypy.core.data.PtyScan*

PtyScan provided by native "ptyd" file format.

PtyScan provided by native "ptyd" file format.

Parameters

- **source** – Explicit source file. If not None or 'file', the data may get processed depending on user input
- **pars** – Input like PtyScan

DEFAULT = Param({'positions_theory': None, 'auto_center': None, 'min_frames': 1, 'orientation': None, '

check(*frames=None*, *start=None*)
Implementation of the check routine for a .ptyd file format

See also:

PtyScan.check()

load(*indices*)
Load from ptyd. Due to possible chunked data, slicing frames is non-trivial

load_positions()

load_weight()

class ptypy.core.data.**MoonFlowerScan**(*pars=None*, ***kwargs*)
Bases: *ptypy.core.data.PtyScan*

Test Ptyscan class producing a romantic ptychographic dataset of a moon illuminating flowers.

Return type dict

Note: This is the *most* important method to change when subclassing *PtyScan*. Most often it suffices to override the constructor and this method to createa subclass of suited for a specific experiment.

load_common()
Override in subclass for custom implementation

Called in initialize()

Loads anything and stores that in a dict. This dict will be available to all processes after *initialize()* through the attribute `common`

The purpose of this method is the same as *load_weight()* and *load_positions()* except for that the contents of `common` have no built-in effect of the behavior in the processing other than the user specifies it in py:meth:*load*

If *load_parallel* is set to *all* or common', this function is executed by all nodes, otherwise the master node executes this function and braodcasts the results to other nodes.

Returns **common**

Return type dict

load_positions()
Override in subclass for custom implementation

Called in initialize()

Loads all positions for all diffraction patterns in this scan. The positions loaded here will be available by all processes through the attribute `self.positions`. If you specify position on a per frame basis in *load()*, this function has no effect.

If theoretical positions `positions_theory` are provided in the initial parameter set *DEFAULT*, specifyiing positions here has NO effect and will be ignored.

The purpose of this function is to avoid reloading and parallel reads on files that may require intense parsing to retrieve the information, e.g. long SPEC log files. If parallel reads or log file parsing for each set of frames is not a time critical issue of the subclass, reimplementing this function can be ignored and it is recommended to only reimplement the *load()* method.

If *load_parallel* is set to *all* or common', this function is executed by all nodes, otherwise the master node executes this function and braodcasts the results to other nodes.

Returns **positions** – A (N,2)-array where *N* is the number of positions.

Return type ndarray

Note: Be aware that this method sets attribute *num_frames* in the following manner.

- If `num_frames == None`: `num_frames = N`.
- If `num_frames < N`, no effect.
- If `num_frames > N`: `num_frames = N`.

load_weight()
Override in subclass for custom implementation

Called in initialize()

Loads a common (2d)-weight for all diffraction patterns. The weight loaded here will be available by all processes through the attribute `self.weight2d`. If a *per-frame-weight* is specified in *load()*, this function has no effect.

The purpose of this function is to avoid reloading and parallel reads. If that is not critical to the implementation, reimplementing this function in a subclass can be ignored.

energy
Property to get and set the energy

lam
Property to get and set the wavelength

lz
Retrieves product of wavelength and propagation distance

propagator
Retrieves propagator, createas propagator instance if necessary.

psize
Property to get and set the pixel size in the propagated plane

resolution
Property to get and set the pixel size in source plane

shape
Property to get and set the *shape* i.e. the frame dimensions

update(*update_propagator=True*)
Update the internal pixel sizes, giving precedence to the sample pixel size (resolution) if self.psize_is_fixed is True.

class ptypy.core.geometry.**BasicNearfieldPropagator**(*geo_dct=None*, *ffttype='scipy'*, ***kwargs*)

Bases: object

Basic two step (i.e. two ffts) Nearfield Propagator.

Parameters

- **geo_pars** (*Param or dict*) – Parameter dictionary as in *DEFAULT*.
- **ffttype** (*str or other*) – Type of CPU-based FFT implementation. One of
  - 'numpy' for numpy.fft.fft2
  - 'scipy' for scipy.fft.fft2
  - 2 or 4-tupel of (forward_fft2(),invese_fft2(), [scaling,inverse_scaling])

DEFAULT = Param({'origin': 'fftshift', 'distance': 7.0, 'center': 'fftshift', 'precedence': None, 'lam': None,

bw(*W*)
computes backward propagated wavefront of input wavefront W

fw(*W*)
computes forward propagated wavefront of input wavefront W

update(*geo_pars=None*, ***kwargs*)
update internal p dictionary. Recompute all internal array buffers

class ptypy.core.geometry.**BasicFarfieldPropagator**(*geo_pars=None*, *ffttype='numpy'*, ***kwargs*)

Bases: object

Basic single step Farfield Propagator.

Includes quadratic phase factors and arbitrary origin in array.

Be aware though, that if the origin is not in the center of the frame, coordinates are rolled peridically, just like in the conventional fft case.

Parameters

- **geo_pars** (*Param or dict*) – Parameter dictionary as in *DEFAULT*.
- **ffttype** (*str or other*) – Type of CPU-based FFT implementation. One of
  - 'numpy' for numpy.fft.fft2

Parent pars are for the

DEFAULT = Param({'positions_theory': None, 'auto_center': None, 'min_frames': 1, 'orientation': None, '

RECIPE = Param({'photons': 100000000.0, 'psf': 0.0, 'density': 0.2})

load(*indices*)

load_positions()

load_weight()

ptypy.core.geometry module

Geometry manager

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

ptypy.core.geometry.**DEFAULT = Param({'origin': 'fftshift', 'distance': 7.0, 'center': 'fftshift', 'precedence**
Default geometry parameters. See also *scan.geometry* and an unflattened representation below

- *origin* = "fftshift"
- *distance* = 7.0, see *distance*
- *center* = "fftshift"
- *precedence* = None, see *precedence*
- *lam* = None, see *lam*
- *energy* = 6.2, see *energy*
- *psize* = 0.000172, see *psize*
- *propagation* = "farfield", see *propagation*
- *shape* = 256
- *misfit* = 0
- *resolution* = None, see *resolution*

class ptypy.core.geometry.**Geo**(*owner=None*, *ID=None*, *pars=None*, ***kwargs*)
Bases: *ptypy.core.classes.Base*

Hold and keep consistant the information about experimental parameters.

Keeps also reference to the Propagator and updates this reference when resolution, pixel size, etc. is changed in *Geo*. Reference to io.paths.recons.

Variables **interact** (*bool (True)*) – If set to True, changes to properties like *energy()*, *lam()*, *shape()* or *psize()* will cause a call to *update()*

Parameters

- **owner** (*Base or subclass*) – Instance of a subclass of *Base* or None. That is usually *Ptycho* for a Geo instance.
- **ID** (*str or int*) – Identifier. Use ID=None for automatic ID assignment.
- **pars** (*dict or Param*) – The configuration parameters. See *Geo.DEFAULT*. Any other kwarg will update internal p dictionary if the key exists in *Geo.DEFAULT*.

DEFAULT = Param({'origin': 'fftshift', 'distance': 7.0, 'center': 'fftshift', 'precedence': None, 'lam': None,

distance
Propagation distance in meters

–*diffuser* = None, see *diffuser*

–*offset* = 0.0, see *offset*

–*size* = None, see *size*

•*model* = None, see *model*

ptypy.core.illumination.**init_storage**(*storage*, *pars*, *energy=None*, ***kwargs*)
Initializes *Storage* *storage* with parameters from *pars*

Parameters

- **storage** (*ptypy.core.Storage*) – A *Storage* instance in the *probe* container of *Ptycho*
- **pars** (Param) – Parameter structure for creating a probe / illumination. See *DEFAULT* Also accepted as argument:
  - string giving the filename of a previous reconstruction to extract storage from.
  - string giving the name of an available TEMPLATE
  - FIXME: document other string cases.
  - numpy array: interpreted as initial illumination.
- **energy** (*float, optional*) – Energy associated with this storage. If None, tries to retrieve the energy from the already initialized ptypy network.

ptypy.core.illumination.**aperture**(*A*, *grids=None*, *pars=None*, ***kwargs*)
Creates an aperture in the shape and dtype of *A* according to x,y-grids *grids*. Keyword Arguments may be any of *DEFAULT*.aperture.

Parameters

- **A** (*ndarray*) – Model array (at least 2-dimensional) to place aperture on top.
- **pars** (*dict or ptypy.utils.Param*) – Parameters, see *DEFAULT*.aperture
- **grids** (*ndarray*) – Cartesion coordinate grids, if None, they will be created with `grids = u.grids(sh[-2:],psize=(1.0,1.0))`

Returns **ap** – Aperture array (complex) in shape of A

Return type ndarray

ptypy.core.manager module

Scan management

The main task of this module is to prepare the data structure for reconstruction, taking a data feed and connecting individual diffraction measurements to the other containers. The way this connection is done is defined by the user through a model definition. The connections are described by the POD objects. This module also takes care of initializing containers according to user-defined rules.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

class ptypy.core.manager.**ModelManager**(*ptycho*, *pars=None*, *scans=None*, ***kwargs*)
Bases: object

Manage ptypy objects creation and update.

The main task of ModelManager is to follow the rules for a given reconstruction model and create:

•the probe, object, exit, diff and mask containers

– ‘scipy’ for scipy.fft.fft2

– 2 or 4-tupel of (forward_fft2(),invese_fft2(), [scaling,inverse_scaling])

DEFAULT = Param({‘origin’: ‘fftshift’, ‘distance’: 7.0, ‘center’: ‘fftshift’, ‘precedence’: None, ‘lam’: None,

bw(*W*)
computes backward propagated wavefront of input wavefront W

fw(*W*)
computes forward propagated wavefront of input wavefront W

update(*geo_pars=None*, ***kwargs*)
update internal p dictionary. Recompute all internal array buffers

ptypy.core.illumination module

This module generates the probe

@author: Bjoern Enders

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

ptypy.core.illumination.**DEFAULT = Param({‘diversity’: Param({‘shift’: None, ‘noise’: None, ‘power’: 1**
Default illumination parameters. See *scan.illumination* and a short listing below

•*diversity* = Param, see *diversity*

–*shift* = None, see *shift*

–*noise* = None, see *noise*

–*power* = 1.0, see *power*

•*recon* = Param, see *recon*

–*layer* = None, see *layer*

–*ID* = None, see *ID*

–*rfile* = "*.ptyr", see *rfile*

•*photons* = None, see *photons*

•*propagation* = Param, see *propagation*

–*antialiasing* = None, see *antialiasing*

–*focussed* = None, see *focussed*

–*spot_size* = None, see *spot_size*

–*parallel* = None, see *parallel*

•*stxm* = Param, see *stxm*

–*label* = None, see *label*

•*ovveride* = None

•*aperture* = Param, see *aperture*

–*rotate* = 0.0

–*form* = "circ", see *form*

–*central_stop* = None, see *central_stop*

–*edge* = 2, see *edge*

- *xy* = Param, see *xy*
- *illumination* = Param, see *illumination*

collect_diff_mask_meta(*label=None*, *filename=None*, *save=False*, *dtype=None*, ***kwargs*)
DEPRECATED attempt to save diffraction data

Parameters

- **label** (*str*) – ptypy label of the scan to save if None, tries to save ALL diffraction data
- **filename** (*str*) – override the file path to write to will change *data_filename* in *scan_info* dict
- **other kwargs are added to 'scan_info' key in the '.h5' file** (*all*) –

make_datasource(*data_pars=None*)
This function will make a static datasource from parameters in the self.scans dict. For any additional file in data.filelist it will create a new entry in self.scans with generic parameters given by the current model.

new_data()
Get all new diffraction patterns and create all views and pods accordingly.

prepare_scan(*label=None*)
Prepare scan specific parameters and create a label if necessary

ptypy.core.model module

Data sharing models.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

ptypy.core.model.**parse_model**(*pars*, *sharing_dct*)
This factory function takes a model description in the input parameters and return an object that can be called with a scan_label (or index) and a diffraction pattern index, and returns probe and object ids.

ptypy.core.paths module

Path manager

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

ptypy.core.paths.**DEFAULT = Param({'recon': 'recons/%(run)s/%(run)s_%(engine)s.ptyr', 'autoplot': 'plo**
Default path parameters. See io.paths and a short listing below

class ptypy.core.paths.**Paths**(*io=None*)
Bases: object

Path managing class

Parameters **io** (*Param or dict*) – Parameter set to pick path info from. See *io*

DEFAULT = Param({'recon': 'recons/%(run)s/%(run)s_%(engine)s.ptyr', 'autoplot': 'plots/%(run)s/%(ru

auto_file(*runtime=None*)
File path for autosave file

- the views
- the PODs

A ptychographic problem is defined by the combination of one or multiple scans. ModelManager uses encapsulate scan-specific elements in .scans und .scans_pars

Note: This class is densely connected to *Ptycho* the seperation in two classes is more history than reason and these classes may get merged in future reeases

Parameters

- **ptycho** (Ptycho) – The parent Ptycho object
- **pars** (*dict or Param*) – Input parameters (see *DEFAULT*) If None uses defaults
- **scans** (*dict or Param*) – Scan-specific parameters, Values should be dict Param that follow the structure of *pars*. If None, tries in ptycho.p.scans else becomes empty dict

DEFAULT = Param({'sharing': Param({'probe_share_with': None, 'probe_share_power': 1, 'object_share_
Default scan parameters. See *scan* and a short listing below

- *sharing* = Param, see *sharing*
  - *probe_share_with* = None, see *probe_share_with*
  - *probe_share_power* = 1, see *probe_share_power*
  - *object_share_power* = 1, see *object_share_power*
  - *object_share_with* = None, see *object_share_with*
- *tags* = "", see *tags*
- *geometry* = Param, see *geometry*
  - *origin* = "fftshift"
  - *distance* = 7.0, see *distance*
  - *center* = "fftshift"
  - *precedence* = None, see *precedence*
  - *lam* = None, see *lam*
  - *energy* = 6.2, see *energy*
  - *psize* = 0.000172, see *psize*
  - *propagation* = "farfield", see *propagation*
  - *shape* = 256
  - *misfit* = 0
  - *resolution* = None, see *resolution*
- *coherence* = Param, see *coherence*
  - *object_dispersion* = None, see *object_dispersion*
  - *energies* = [1.0]
  - *spectrum* = [1.0], see *spectrum*
  - *num_object_modes* = 1, see *num_object_modes*
  - *num_probe_modes* = 1, see *num_probe_modes*
  - *probe_dispersion* = None, see *probe_dispersion*
- *sample* = Param, see *sample*

containers
Dict of all *Container* instances in the pool of self

finalize()
Cleanup

init_communication()
Called on __init__ if `level>=3`.

Initializes ZeroMQ communication on the master node and spawns an optional plotting client.

init_data(*print_stats=True*)
Called on __init__ if `level>=2`.

Creates a datasource and calls for `ModelManager.new_data()` Prints statistics on the ptypy structure if `print_stats=True`

init_engine(*label=None*, *epars=None*)
Called on __init__ if `level>=4`.

Initializes engine with label *label* from paramaters and lists it internally in `self.engines` which is an ordered dictinoary.

Parameters

- **label** (*str*) – Label of engine which is to be created from parameter set of the same label in input parameter tree. If `None`, an engine is created for each available parameter set in input parameter tree sorted by label.
- **epars** (*Param or dict*) – Set of engine parameters. The created engine is listed as *auto00*, *auto01* , etc in `self.engines`

init_structures()
Called on __init__ if `level>=1`.

Prepare everything for reconstruction. Creates attributes `modelm` and the containers `probe` for illumination, `obj` for the samples, `exit` for the exit waves, `diff` for diffraction data and `mask` for detectors masks

classmethod load_run(*runfile*, *load_data=True*)
Load a previous run.

Parameters

- **runfile** (*str*) – file dump of Ptycho class
- **load_data** (*bool*) – If *True* also load data (thus regenerating pods & views for 'minimal' dump

Returns **P** – Ptycho instance with `level==2`

Return type *Ptycho*

p = None
Reference for parameter tree, with which this instance was constructed.

plot_overview(*fignum=100*)
plots whole the first four layers of every storage in probe, object % diff

pods
Dict of all *POD* instances in the pool of self

print_stats(*table_format=None*, *detail='summary'*)
Calculates the memrory usage and other info of ptycho instance

run(*label=None*, *epars=None*, *engine=None*)
Called on __init__ if `level>=5`.

Start the reconstruction with at least one engine. As a consequence, `self.runtime` will be filled with content.

get_path(*path*, *runtime*)

plot_file(*runtime=None*)
File path for plot file

recon_file(*runtime=None*)
File path for reconstruction file

run(*run*)
Determine run name

ptypy.core.ptycho module

ptycho - definition of the upper-level class Ptycho.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

class `ptypy.core.ptycho.`**Ptycho**(*pars=None*, *level=2*, ***kwargs*)
Bases: *ptypy.core.classes.Base*

Ptycho : A ptychographic data holder and reconstruction manager.

This is the highest level class. It organizes and contains the data, manages the reconstruction engines, and interacts with the outside world.

If MPI is enabled, this class acts both as a manager (rank = 0) and a worker (any rank), and most information exists on all processes. In principle the only part that is divided between processes is the diffraction data.

By default Ptycho is instantiated once per process, but it can also be used as a managed container to load past runs.

Variables

- **CType,FType** (*numpy.dtype*) – numpy dtype for arrays. *FType* is for data, i.e. real-valued arrays, *CType* is for complex-valued arrays
- **interactor** (ptypy.io.interaction.Server) – ZeroMQ interaction server for communication with e.g. plotting clients
- **runtime** (Param) – Runtime information, e.g. errors, iteration etc.
- **modelm** (ModelManager) – THE managing instance for *POD*, *View* and *Geo* instances
- **probe,obj,exit,diff,mask** (Container) – Container instances for illuminations, samples, exit waves, diffraction data and detector masks / weights

Parameters

- **pars** (Param) – Input parameters, subset of the *ptypy parameter tree*
- **level** (*int*) – Determines how much is initialized.
  - <=0 : empty ptypy structure
  - **1** [reads parameters, configures interaction server,] see *init_structures()*
  - **2** [also configures Containers, initializes Modelmanager] see *init_data()*
  - **3** [also initializes ZeroMQ-communication] *init_communication()*
  - **4** [also initializes reconstruction engines,] see *init_engine()*
  - **>=4** [also and starts reconstruction] see *run()*

DEFAULT = Param({'engine': Param({'Dummy': Param({'itertime': 2.0}), 'DM': Param({'overlap_max_it
Default ptycho parameters which is the trunk of the default *ptypy parameter tree*

–*probes* = None

ptypy.core.sample module

This module generates a sample

@author: Bjoern Enders

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

ptypy.core.sample.**DEFAULT = Param({'diversity': Param({'shift': None, 'noise': None, 'power': 1.0}), 'pr**

Default sample parameters. See *scan.sample* and a short listing below

- *diversity* = Param, see *diversity*
  - –*shift* = None, see *shift*
  - –*noise* = None, see *noise*
  - –*power* = 1.0, see *power*
- *process* = Param, see *process*
  - –*density* = None, see *density*
  - –*offset* = 0, see *offset*
  - –*zoom* = None, see *zoom*
  - –*thickness* = None, see *thickness*
  - –*ref_index* = (0.5+0j), see *ref_index*
  - –*formula* = None, see *formula*
  - –*smoothing* = 2, see *smoothing*
- *recon* = Param, see *recon*
  - –*layer* = None, see *layer*
  - –*rfile* = "*.ptyr", see *rfile*
  - –*ID* = None, see *ID*
- *stxm* = Param, see *stxm*
  - –*label* = None, see *label*
- *override* = None
- *model* = None, see *model*
- *fill* = 1.0, see *fill*

ptypy.core.sample.**init_storage**(*storage*, *sample_pars=None*, *energy=None*)

Initializes a storage as sample transmission.

Parameters

- **storage** (Storage) – The object *Storage* to initialize
- **sample_pars** (Param) – Parameter structure that defines how the sample is created. See *DEFAULT* for the parameters.
- **energy** (*float, optional*) – Energy associated in the experiment for this sample object. If None, the ptypy structure is searched for the appropriate *Geo* instance: storage.views[0].pod.geometry.energy

Parameters

- **label** (*str, optional*) – Engine label of engine to run. If None all available engines are run in the order they were stored in self.engines. If the engine is not yet created, *init_engine()* is called for that label.
- **epars** (*dict or Param, optional*) – Engine parameter set. An engine is created from this set, using *init_engine()* and run immediately afterwards. For parameters see *engine*
- **engine** (*BaseEngine, optional*) – An engine instance that should be a subclass of BaseEngine or have the same methods.

save_run(*alt_file=None*, *kind='minimal'*, *force_overwrite=True*)

Save run to file.

As for now, diffraction / mask data is not stored

Parameters

- **alt_file** (*str*) – Alternative filepath, will override io.save_file
- **kind** (*str*) – Type of saving, one of:
  - *'minimal'*, only initial parameters, probe and object storages and runtime information is saved.
  - *'full_flat'*, (almost) complete environment

ptypy.core.ptycho.**DEFAULT_io = Param({'autoplot': Param({'threaded': True, 'layout': 'default', 'dump**

Default io parameters. See *io* and a short listing below

- *autoplot* = Param, see *autoplot*
  - –*threaded* = True, see *threaded*
  - –*layout* = "default", see *layout*
  - –*dump* = True, see *dump*
  - –*imfile* = "plots/%(run)s/%(run)s_%(engine)s_%(iteration)04d.png", see *imfile*
  - –*interval* = 1, see *interval*
  - –*make_movie* = False, see *make_movie*
- *home* = "./", see *home*
- *interaction* = Param, see *interaction*
  - –*address* = "tcp://127.0.0.1", see *address*
  - –*poll_timeout* = 10
  - –*connections* = 10, see *connections*
  - –*pingtimeout* = 10
  - –*pinginterval* = 2
  - –*port* = 5560, see *port*
- *rfile* = "recons/%(run)s/%(run)s_%(engine)s.ptyr", see *rfile*
- *autosave* = Param, see *autosave*
  - –*objects* = None
  - –*rfile* = "dumps/%(run)s/%(run)s_%(engine)s_%(iteration)04d.ptyr", see *rfile*
  - –*interval* = 10, see *interval*

- **nth** (*int, optional*) – Number of points in first shell

Returns **pos** – A (N,2)-array of positions.

Return type ndarray

Examples

```
>>> from ptypy.core import xy
>>> from matplotlib import pyplot as plt
>>> pos = xy.round_scan()
>>> plt.plot(pos[:,1],pos[:,0],'o-');plt.show()
```

ptypy.core.xy.**raster_scan**(*ny=10*, *nx=10*, *dy=1.5e-06*, *dx=1.5e-06*)
Generates a raster scan.

Parameters

- **nx** (*ny,*) – Number of steps in *y* (vertical) and *x* (horizontal) direction *x* is the fast axis
- **dx** (*dy,*) – Step size (grid spacinf) in *y* and *x*

Returns **pos** – A (N,2)-array of positions. It is `N = (nx+1)*(nx+1)`

Return type ndarray

Examples

```
>>> from ptypy.core import xy
>>> from matplotlib import pyplot as plt
>>> pos = xy.raster_scan()
>>> plt.plot(pos[:,1],pos[:,0],'o-');plt.show()
```

ptypy.core.xy.**spiral_scan**(*dr=1.5e-06*, *r=7.5e-06*, *maxpts=None*)
Generates a spiral scan.

Parameters

- **r** (*float*) – Number of radial steps from center, `nr + 1` shells will be made
- **dr** (*float*) – Step size (shell spacing)
- **nth** (*int, optional*) – Number of points in first shell

Returns **pos** – A (N,2)-array of positions. It is

Return type ndarray

Examples

```
>>> from ptypy.core import xy
>>> from matplotlib import pyplot as plt
>>> pos = xy.spiral_scan()
>>> plt.plot(pos[:,1],pos[:,0],'o-');plt.show()
```

Module contents

Core functionalities of the package.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

ptypy.core.sample.**simulate**(*A*, *pars*, *energy*, *fill=1.0*, *prefix=''*, ***kwargs*)
Simulates a sample object into model numpy array *A*

Parameters

- **A** (*ndarray*) – Numpy array as buffer. Must be at least twodimensional
- **pars** (Param) – Simulation parameters. See *DEFAULT* .simulate

ptypy.core.save_load module

ptypy.core.save_load.**link**(*pool*, *replace_objects_only=False*, *preserve_input_pool=True*)
Reverse operation to unlink.

ptypy.core.save_load.**unlink**(*obj*)
Looks into all references of obj. Places references in a pool with string labels resembling their python ID plus an object dependent prefix. Reduces all objects and dicts to dictionaries. Reduces all tupels to list. All references are replaced by the labels in the pool.

Original references in 'obj' are preserved

ptypy.core.save_load.**to_h5**(*filename*, *obj*)

ptypy.core.save_load.**from_h5**(*filename*)

ptypy.core.xy module

This module generates the scan patterns.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

ptypy.core.xy.**DEFAULT = Param({'count': None, 'jitter': None, 'spacing': 1.5e-06, 'steps': 10, 'extent': 1.5e**
Default pattern parameters. See *scan.xy* and a short listing below

- *count* = `None`, see *count*
- *jitter* = `None`, see *jitter*
- *spacing* = `1.5e-06`, see *spacing*
- *steps* = `10`, see *steps*
- *extent* = `1.5e-05`, see *extent*
- *offset* = `0`, see *offset*
- *override* = `None`
- *model* = `None`, see *model*

ptypy.core.xy.**from_pars**(*xypars=None*)
Creates position array from parameter tree *pars*. See *DEFAULT*

Parameters **pars** (Param) – Input parameters

Returns **ndarray pos** A numpy.ndarray of shape `(N,2)` for *N* positios

ptypy.core.xy.**round_scan**(*dr=1.5e-06*, *nr=5*, *nth=5*, *bullseye=True*)
Generates a round scan

Parameters

- **nr** (*int*) – Number of radial steps from center, `nr + 1` shells will be made
- **dr** (*float*) – Step size (shell spacing)

engine_finalize()
Delete temporary containers.

engine_initialize()
Prepare for ML reconstruction.

engine_iterate(*num=1*)
Compute *num* iterations.

engine_prepare()
last minute initialization, everything, that needs to be recalculated, when new data arrives

ptypy.engines.base module

Base engine. Used to define reconstruction parameters that are shared by all engines.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

class ptypy.engines.base.**BaseEngine**(*ptycho*, *pars=None*)
Bases: object

Base reconstruction engine.

In child classes, overwrite the following methods for custom behavior :

engine_initialize engine_prepare engine_iterate engine_finalize

Base reconstruction engine.

Parameters

- **ptycho** (Ptycho) – The parent *Ptycho* object.
- **pars** (*Param or dict*) – Initialization parameters

DEFAULT = Param({'probe_support': 0.8, 'probe_update_start': 0, 'probe_center_tol': None, 'numiter': 1

engine_finalize()
Engine-specific finalization. Used to wrap-up engine-specific stuff. Called at the end of self.finalize()

engine_initialize()
Engine-specific initialization. Called at the end of self.initialize().

engine_iterate(*num*)
Engine single-step iteration. All book-keeping is done in self.iterate(), so this routine only needs to implement the "core" actions.

engine_prepare()
Engine-specific preparation. Last-minute initialization providing up-to-date information for reconstruction. Called at the end of self.prepare()

finalize()
Clean up after iterations are done

initialize()
Prepare for reconstruction.

iterate(*num=None*)
Compute one or several iterations.

num : None,int number of iterations. If None or num<1, a single iteration is performed

prepare()
Last-minute preparation before iterating.

license GPLv2, see LICENSE for details.

ptypy.engines package

Submodules

ptypy.engines.DM module

Difference Map reconstruction engine.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

class ptypy.engines.DM.**DM**(*ptycho_parent*, *pars=None*)
Bases: *ptypy.engines.base.BaseEngine*

Difference map reconstruction engine.

DEFAULT = Param({'overlap_max_iterations': 10, 'fourier_relax_factor': 0.05, 'overlap_converge_factor':

center_probe()

engine_finalize()
try deleting ever helper contianer

engine_initialize()
Prepare for reconstruction.

engine_iterate(*num=1*)
Compute *num* iterations.

engine_prepare()
last minute initialization, everything, that needs to be recalculated, when new data arrives

object_update()
DM object update.

overlap_update()
DM overlap constraint update.

probe_update()
DM probe update.

ptypy.engines.ML module

Maximum Likelihood reconstruction engine.

TODO:

- Implement other regularizers

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

class ptypy.engines.ML.**ML**(*ptycho_parent*, *pars=None*)
Bases: *ptypy.engines.base.BaseEngine*

Maximum likelihood reconstruction engine.

DEFAULT = Param({'floating_intensities': False, 'reg_del2_amplitude': 0.01, 'smooth_gradient': 0, 'intensit

class ptypy.engines.**BaseEngine**(*ptycho*, *pars=None*)
Bases: object

Base reconstruction engine.

In child classes, overwrite the following methods for custom behavior :

engine_initialize engine_prepare engine_iterate engine_finalize

Base reconstruction engine.

Parameters

- **ptycho** (Ptycho) – The parent *Ptycho* object.
- **pars** (*Param or dict*) – Initialization parameters

DEFAULT = Param({'probe_support': 0.8, 'probe_update_start': 0, 'probe_center_tol': None, 'numiter': 1

engine_finalize()
Engine-specific finalization. Used to wrap-up engine-specific stuff. Called at the end of self.finalize()

engine_initialize()
Engine-specific initialization. Called at the end of self.initialize().

engine_iterate(*num*)
Engine single-step iteration. All book-keeping is done in self.iterate(), so this routine only needs to implement the "core" actions.

engine_prepare()
Engine-specific preparation. Last-minute initialization providing up-to-date information for reconstruction. Called at the end of self.prepare()

finalize()
Clean up after iterations are done

initialize()
Prepare for reconstruction.

iterate(*num=None*)
Compute one or several iterations.

num : None,int number of iterations. If None or num<1, a single iteration is performed

prepare()
Last-minute preparation before iterating.

ptypy.experiment package

Submodules

ptypy.experiment.I13 module

Scan loading recipe for the I13 beamline, Diamond.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

class ptypy.experiment.I13.**I13Scan**(*pars=None*, ***kwargs*)
Bases: *ptypy.core.data.PtyScan*

I13 (Diamond Light Source) data preparation class.

DEFAULT = Param({'positions_theory': None, 'auto_center': False, 'chunk_format': '.chunk%02d', 'min_

ptypy.engines.utils module

Engine-specific utilities. This could be compiled, or GPU accelerated.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

ptypy.engines.utils.**Cdot**(*c1*, *c2*)
Compute the dot product on two containers *c1* and *c2*. No check is made to ensure they are of the same kind.

Parameters **c1, c2** (Container) – Input

Returns The dot product (*scalar*)

ptypy.engines.utils.**Cnorm2**(*c*)
Computes a norm2 on whole container *c*.

Parameters **c** (Container) – Input

Returns The norm2 (*scalar*)

See also:

ptypy.utils.math_utils.norm2()

ptypy.engines.utils.**basic_fourier_update**(*diff_view*, *pbound=None*, *alpha=1.0*, *LL_error=True*)
Fourier update one a single view using its associated pods. Updates on all pods' exit waves.

Parameters

- **diff_view** (View) – View to diffraction data
- **alpha** (*float, optional*) – Mixing between old and new exit wave. Valid interval `[0, 1]`
- **pbound** (*float, optional*) – Power bound. Fourier update is bypassed if the quadratic deviation between diffraction data and *diff_view* is below this value. If `None`, fourier update always happens.
- **LL_error** (*bool*) – If `True`, calculates log-likehood and puts it in the last entry of the returned error vector, else puts in `0.0`

Returns

error – 1d array, `error = np.array([err_fmag, err_phot, err_exit]).`

- *err_fmag*, Fourier magnitude error; quadratic deviation from root of experimental data
- *err_phot*, quadratic deviation from experimental data (photons)
- *err_exit*, quadratic deviation of exit waves before and after Fourier iteration

Return type ndarray

Module contents

Engines module.

Implements the difference map (DM) and maximum likelihood (ML) reconstruction algorithms for ptychography.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

•bool if the end of scan was reached (None if this routine doesn't know)

correct (*raw*, *weights*, *common*)
Apply (eventual) corrections to the raw frames. Convert from "raw" frames to usable data. :param raw: :param weights: :param common: :return:

load (*indices*)
Load frames given by the indices.

Parameters **indices** –

Returns

load_common ()
Load scanning positions, dark, white field and distortion files.

ptypy.experiment.ID16Anfp.**undistort** (*frame*, *delta*)
Frame distortion correction (linear interpolation) Any value outside the frame is replaced with a constang value (mean of the complete frame)

Parameters

- **frame** (*ndarray*) – the input frame data
- **delta** (*2-tuple*) – containing the horizontal and vertical displacements respectively.

Returns The corrected frame of same dimension and type as frame.

Return type ndarray

ptypy.experiment.optiklabor module

Created on Nov 22 2013

@author: Bjeorn Enders

class ptypy.experiment.optiklabor.**FliSpecScanMultexp** (*pars=None*, ***kwargs*)
Bases: *ptypy.core.data.PtyScan*

check (*frames_requested*, *start=0*)

correct (*raw*, *weights*, *common*)

load (*indices*)

load_common ()

ptypy.experiment.plugin module

Scan loading plugin. Meant to make easier user-generated, problem-specific data preparation.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

ptypy.experiment.plugin.**makeScanPlugin** (*pars=None*)
Factory wrapper that provides a PtyScan object.

ptypy.experiment.spec module

Utility module to read spec files Adapted from spec_read.m by Andreas Menzel (PSI)

This file is part of the PTYPY package.

check (*frames*, *start=0*)
Returns the number of frames available from starting index *start*, and whether the end of the scan was reached.

Parameters

- **frames** – Number of frames to load
- **start** – starting point

Returns (frames_available, end_of_scan)

•the number of frames available from a starting point *start*

•bool if the end of scan was reached (None if this routine doesn't know)

correct (*raw*, *weights*, *common*)
Apply (eventual) corrections to the frames. Convert from "raw" frames to usable data. :param raw: :param weights: :param common: :return:

load (*indices*)
Load frames given by the indices.

Parameters **indices** –

Returns

load_common ()
Load scanning positions and mask file.

load_positions ()
Load the positions and return as an (N,2) array

ptypy.experiment.ID16Anfp module

Scan loading recipe for the ID16A beamline at ESRF - near-field ptycho setup.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

class ptypy.experiment.ID16Anfp.**ID16AScan** (*pars=None*, ***kwargs*)
Bases: *ptypy.core.data.PtyScan*

Subclass of PtyScan for ID16A beamline (specifically for near-field ptychography).

Create a PtyScan object that will load ID16A data.

Parameters

- **pars** – preparation parameters
- **kwargs** – Additive parameters

check (*frames*, *start=0*)
Returns the number of frames available from starting index *start*, and whether the end of the scan was reached.

Parameters

- **frames** – Number of frames to load
- **start** – starting point

Returns (frames_available, end_of_scan)

•the number of frames available from a starting point *start*

ptypy.io.h5rw module

Wrapper to store nearly anything in an hdf5 file.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

ptypy.io.h5rw.**h5write**(*filename*, *{'var1'=..., 'var2'=..., ...}*)
h5write(filename, var1=..., var2=..., ...) h5write(filename, dict, var1=..., var2=...)

Writes variables var1, var2, ... to file filename. The key-value arguments have precedence on the provided dictionnary.

supported variable types are: * scalars * numpy arrays * strings * lists * dictionaries

(if the option UNSUPPORTED is equal to 'pickle', any other type is pickled and saved. UNSUPPORTED = 'ignore' silently eliminates unsupported types. Default is 'fail', which raises an error.)

The file mode can be chosen according to the h5py documentation. It defaults to overwriting an existing file.

ptypy.io.h5rw.**h5append**(*filename*, *{'var1'=..., 'var2'=..., ...}*)
h5append(filename, var1=..., var2=..., ...) h5append(filename, dict, var1=..., var2=...)

Appends variables var1, var2, ... to file filename. The key-value arguments have precedence on the provided dictionnary.

supported variable types are: * scalars * numpy arrays * strings * lists * dictionaries

(if the option UNSUPPORTED is equal to 'pickle', any other type is pickled and saved. UNSUPPORTED = 'ignore' silently eliminates unsupported types. Default is 'fail', which raises an error.)

The file mode can be chosen according to the h5py documentation. It defaults to overwriting an existing file.

ptypy.io.h5rw.**h5read**(*filename*)
h5read(filename, s1, s2, ...) h5read(filename, (s1,s2, ...))

Read variables from a hdf5 file created with h5write and returns them as a dictionary.

If specified, only variable named s1, s2, ... are loaded.

Variable names support slicing and group access. For instance, provided that the file contains the appropriate objects, the following syntax is valid:

a = h5read('file.h5', 'myarray[2:4]') a = h5read('file.h5', 'adict.thekeyIwant')

Another way of slicing, is with the slice keyword argument, which will take the provided slice object and apply it on the last variable name:

a = h5read('file.h5', 'array1', 'array2', slice=slice(1,2)) # Will read array2[1:2]

h5read(filename_with_wildcard, ... , doglob=True) Reads sequentially all globbed filenames.

ptypy.io.h5rw.**h5info**(*filename*)
Prints out a tree structure of given h5 file.

ptypy.io.imageIO module

Wrapper over Python Imaging Library to read images (+ metadata) and write images.

ptypy.io.imageIO.**imread**(*filename*, *doglob=None*, *roi=None*)

d,meta = imread(filename) reads array in image file and returns it as a numpy array, along with metadata (from header). d and meta are lists if filename is a list of file names.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

class ptypy.experiment.spec.**SpecInfo**(*spec_filename*)
Bases: object

parse(*rehash=False*)
Parse the spec dat-file and extract information.

class ptypy.experiment.spec.**SpecScan**
Bases: object

ptypy.experiment.spec.**verbose**(*n*, *s*)
This function should be replaced by the real verbose class after import. It is here for convenience since this module has no other external dependencies.

Module contents

Beamline-specific data preparation modules.

Currently available:

- I13DLS

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

ptypy.io package

Submodules

ptypy.io.edfIO module

IO module for handling the file format EDF used by the ESRF beamline id19 Provides read, write and readHeader functions.

translated to python: April 2010, Dieter Hahn

Among other, contains the functions readData writeData WHICH IS NOT YET IMPLEMENTED

ptypy.io.edfIO.**edfread**(*filename*, *doglob=None*, *roi=None*)

d,meta = edfread(filename) reads array in edf file filename and returns it as a numpy array, along with metadata (from header). d and meta are lists if filename is a list of file names.

... = edfread(filename, doglob=True) reads all matching files if filename contains unix-style wildcards, and returns lists.

... = edfread(filename, doglob=False) ignores wildcards

... = edfread(filename, doglob=None) [default] behaves like doglob=True, except that it returns a list only if filename contains wildcards, while doglob=True always returns a list, even if there is only one match.

... = edfread(filename, roi=(RowFrom, RowTo, ColumnFrom, ColumnTo)) returns a region of interest (applied on all files if gobbing or if filename is a list)

make_ID_pool()

process_requests(*tinterval=0.001*)
Give permission to the serving thread to send objects and wait for it to complete (safer!)

register(*obj*, *name*)
Exposes the content of an object for transmission and interaction. For now this is equivalent to Interactor.object[name] = obj, but maybe use weakref in the future?

send_error(*error_message*)
Queue an ERROR message for all connected clients.

send_warning(*warning_message*)
Queue a warning message for all connected clients.

stop()

class ptypy.io.interaction.**Client**(*pars=None*, ***kwargs*)
Bases: object

Basic but complete client to interact with the server.

Parameters **pars** (*dict or Param*) – Parameter set for the client, see *DEFAULT*

Keyword Arguments

- **address** (*str*) – Primary address of the remote server.
- **port** (*int*) – Primary port of the remote server.
- **poll_timeout** (*float*) – Network polling interval (in milliseconds!).
- **pinginterval** (*float*) – Interval to check pings (in seconds).

DEFAULT = Param({'connections': 10, 'poll_timeout': 100, 'pinginterval': 1, 'port': 5560, 'address': 'tcp://
Default parameters, see also *io.interaction*

- *connections* = 10, see *connections*
- *poll_timeout* = 100
- *pinginterval* = 1
- *port* = 5560, see *port*
- *address* = "tcp://127.0.0.1", see *address*

activate()
Create the thread.

avail()
Queries the server for the name of objects available. ! Synchronous call !

do(*execstr*, *timeout=0*, *tag=None*)
Modify and object using an exec string. This function returns the "ticket number" which identifies the object once it will have been transmitted. If timeout > 0 and the requested object has been transmitted within timeout seconds, return a tuple (ticket, data).

flush()
Delete all stored data (and accompanying status).

get(*evalstr*, *timeout=0*, *tag=None*)
Requests an object (or part of it) using an eval string. This function returns the "ticket number" which identifies the object once it will have been transmitted. If timeout > 0 and the requested object has been transmitted within timeout seconds, return a tuple (ticket, data).

get_now(*evalstr*)
Synchronous get. May be dangerous, but should be safe for small objects like parameters.

newdata(*ticket*)
Meant to be replaced, e.g. to send signals to a GUI.

... = imread(filename, doglob=True) reads all matching files if filename contains unix-style wildcards, and returns lists.

... = imread(filename, doglob=False) ignores wildcards

... = imread(filename, doglob=None) [default] behaves like doglob=True, except that it returns a list only if filename contains wildcards, while doglob=True always returns a list, even if there is only one match.

... = imread(filename, roi=(RowFrom, RowTo, ColumnFrom, ColumnTo)) returns a region of interest (applied on all files if gobbing or if filename is a list)

ptypy.io.image_read module

ptypy.io.image_read.**image_read**(*filename*, **args*, ***kwargs*)
Attempts to import image data from any file.

ptypy.io.interaction module

Interaction module

Provides the server and a basic client to interact with it.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

class ptypy.io.interaction.**Server**(*pars=None*, ***kwargs*)
Bases: object

Main server class.

Interaction server, meant to run asynchronously with process 0 to manage client requests.

Parameters **pars** (*dict or Param*) – Parameter set for the server.

Keyword Arguments

- **address** (*str*) – Primary address
- **port** (*int*) – Primary port
- **connections** (*int*) – number of ports to open on demand *behind* the primary port.
- **poll_timeout** (*float*) – Network polling interval (in milliseconds!).
- **pinginterval** (*float*) – Interval to check pings (in seconds).
- **pingtimeout** (*float*) – Ping time out: client disconnected after this period (in seconds).

DEFAULT = Param({'poll_timeout': 10, 'connections': 10, 'pingtimeout': 10, 'pinginterval': 2, 'port': 5560,
Default parameters, see also *io.interaction*

- *poll_timeout* = 10
- *connections* = 10, see *connections*
- *pingtimeout* = 10
- *pinginterval* = 2
- *port* = 5560, see *port*
- *address* = "tcp://127.0.0.1", see *address*

activate()
This needs to be run for the thread to initialize. Returns port where the server listens or None if Server failed

ptypy.io.rawIO module

read .raw images from FLI camera Created in Nov 2013 TODO: add masking function to mask out hot/dead pixels of detector @author: Bjoern Enders

ptypy.io.rawIO.**rawread**(*filename*, *doglob=None*, *roi=None*)

d,meta = rawread(filename) reads array in CDF file filename and returns it as a numpy array, along with metadata (from header). d and meta are lists if filename is a list of file names.

... = rawread(filename, doglob=True) reads all matching files if filename contains unix-style wildcards, and returns lists.

... = rawread(filename, doglob=False) ignores wildcards

... = rawread(filename, doglob=None) [default] behaves like doglob=True, except that it returns a list only if filename contains wildcards, while doglob=True always returns a list, even if there is only one match.

... = rawread(filename, roi=(RowFrom, RowTo, ColumnFrom, ColumnTo)) returns a region of interest (applied on all files if gobbing or if filename is a list)

Module contents

Input/Output utilities.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

ptypy.resources package

Module contents

ptypy.resources.**flower_obj**(*shape=None*)

ptypy.resources.**moon_pr**(*shape=None*)

ptypy.simulations package

Submodules

ptypy.simulations.detector module

Detector module

Note that "dark current" and consequently exposure time is not used. Modern detectors are insensitive to dark current. Scale the incoming Intensity accordingly to match the number of photons in your acquisition.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

ptypy.simulations.detector.**shot**(*I*, *exp=0.1*, *flux=100000.0*, *sensitivity=1.0*, *dark_c=None*, *io_noise=0.0*, *full_well=1023*, *el_per_ADU=1.0*, *offset=50.0*)

I : intensity distribution flux : overall photon photons per seconds coming in exp : exposition time in sec

p = None
sanity check for port range: if str(p.port_range)==p.port_range:

from ptypy.utils import str2range p.port_range = str2range(p.port_range)

poll(*ticket=None*, *tag=None*)
Returns true if the transaction for a given ticket is completed. If ticket and tag are None, returns true only if no transaction is pending

set(*varname*, *varvalue*, *timeout=0*, *tag=None*)
Sets an object named varname to the value varvalue.

stop()

stop_server()
Send a SHUTDOWN command - is this a good idea?

unexpected_ticket(*ticket*)
Used to deal with warnings sent by the server.

wait(*ticket=None*, *tag=None*, *timeout=None*)
Blocks and return True only when the transaction for a given ticket is completed. If ticket is None, returns only when no more transaction are pending. If timeout is a positive number, wait will return False after timeout seconds if the ticket(s) had not been processed yet.

ptypy.io.json_rw module

Wrapper to store nearly anything in a file using JSON encoding.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

ptypy.io.json_rw.**jwrite**(*filename*, *{'var1'=..., 'var2'=..., ...}*)
jwrite(filename, var1=..., var2=..., ...) jwrite(filename, dict, var1=..., var2=...)

Writes variables var1, var2, ... to file filename using JSON encoding. The key-value arguments have precedence on the provided dictionary.

All JSON supported types are supported, as well as numpy arrays, which are replaced by a base64 encoding.

ptypy.io.json_rw.**jread**(*filename*, **args*, ***kwargs*)
h5read(filename) h5read(filename, s1, s2, ...) h5read(filename, (s1,s2, ...))

Read variables from a JSON file created with jwrite and returns them as a dictionary.

The following features are meant to have an interface identical to h5read. Note however that no time of memory is saved by "loading" only a subset of variables or slices.

If specified, only variable named s1, s2, ... are loaded.

Variable names support slicing and group access. For instance, provided that the file contains the appropriate objects, the following syntax is valid:

a = jread('file.h5', 'myarray[2:4]') a = jread('file.h5', 'adict.thekeyIwant')

Another way of slicing is with the slice keyword argument which will take the provided slice object and apply it on the last variable name:

a = jread('file.json', 'array1', 'array2', slice=slice(1,2)) # Will read array2[1:2]

jread(filename_with_wildcard, ... , doglob=True) Reads sequentially all globbed filenames.

Module contents

Ptychographic simulation module.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

ptypy.utils package

Submodules

ptypy.utils.array_utils module

utility functions to manipulate/reshape numpy arrays.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

`ptypy.utils.array_utils.`**`grids`**(*sh*, *psize=None*, *center='geometric'*, *FFTlike=True*)

`q0,q1,... = grids(sh)` returns centered coordinates for a N-dimensional array of shape sh (pixel units)

`q0,q1,... = grids(sh,psize)` gives the coordinates scaled according to the given pixel size psize.

`q0,q1,... = grids(sh,center='fftshift')` gives the coordinates shifted according to fftshift convention for the origin

`q0,q1,... = grids(sh,psize,center=(c0,c1,c2,...))` gives the coordinates according scaled with psize having the origin at (c0,c1,..)

Parameters

- **sh** (*tuple of int*) – The shape of the N-dimensional array
- **psize** (*float or tuple of float*) – Pixel size in each dimensions
- **center** (*tupel of int*) – Tuple of pixel, or use `center='fftshift'` for fftshift-like grid and `center='geometric'` for the matrix center as grid origin
- **FFTlike** (*bool*) – If False, grids ar not bound by the interval [-sh//2:sh//2[

Returns The coordinate grids

Return type ndarray

`ptypy.utils.array_utils.`**`switch_orientation`**(*A*, *orientation*, *center=None*)

Switches orientation of Array *A* along the last two axes *(-2,-1)*

Parameters

- **A** (*array-like*) – input array, must be at least twodimensional
- **orientation** (*tuple*) – 3-tuple of booleans (transpose,flipud,fliplr)
- **center** (*tuple, optional*) – move this coordinate alomg with the transformations

Returns

- **out** (*ndarray*) – A view of *A* where either rows, columns and axis may be reversed
- **center** (*tuple*) – new center

io_noise : readout noise rms full_well : electron capacity of each pixel el_per_ADU : conversion effficiency of electrons to digitally counted units dark_curr : electrons per second per pixel on average

class `ptypy.simulations.detector.`**`Detector`**(*pars=None*)

Bases: `object`

`filter`(*intensity_stack*, *convert_dtype=False*)

`ptypy.simulations.detector.`**`conv`**(*A*, *inp*, ***kwargs*)

`ptypy.simulations.detector.`**`fill2D`**(*imA*, *imB*, *offset*)

fill array imA with imB

ptypy.simulations.ptysim_utils module

Utilities for the simulation package.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

`ptypy.simulations.ptysim_utils.`**`exp_positions`**(*positions*, *drift=0.0*, *scale=0.0*, *noise=0.0*)

Takes postion array and messes around with it

ptypy.simulations.simscan module

Simulation of ptychographic datasets.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

class `ptypy.simulations.simscan.`**`SimScan`**(*pars=None*, *scan_pars=None*, *recipe_pars=None*, ***kwargs*)

Bases: *`ptypy.core.data.PtyScan`*

Simulates a ptychographic scan and acts as Virtual data source. The simulation is carried out in the following way.

TO BE FILLED WITH CONTENT

Parameters

- **pars** (Param) – PtyScan parameters including a *recipe*. See *`data.GENERIC`*.
- **scan_pars** (*Param or dict*) – Parameters for a single scan. Use this especially to set positions, illumination and sample.
- **recipe_pars** (*Param or dict*) – Equivalent alternative to *pars['recipe']*.

`RECIPE` = Param({'pos_noise': 1e-10, 'plot': True, 'verbose_level': 1, 'psf': None, 'pos_drift': 0, 'frame_siz

`load`(*indices*)

Load data, weights and positions from internal dictionarys

`manipulate_ptycho`(*ptycho*)

Overwrite in child class for inline manipulation of the ptycho instance that is created by the Simulation

•**if *hplanes* is scalar and positiv :** pads symmetrically with a fill specified with 'fillpar' and 'filltype' look at function pad_lr() for detail.

•**if *hplanes* is tupel :** function pads /crops at (begin, end) according to the tupel.

Examples

```
>>> import numpy as np
>>> from ptypy.utils import crop_pad_axis
>>> A=np.ones((8,9))
```

Add a total of 2 rows, one at top, one at bottom.

```
>>> B=crop_pad_axis(A,2,0)
```

That is the same as

```
>>> B = crop_pad_axis(A,(1,1),0))
```

Crop 3 columns on the left side and pad 2 columns on the right:

```
>>> B = crop_pad_axis(A,(-3,2),1)
```

Crop one plane on low-side and high-side (total of 2) of Volume V:

```
>>> V=np.random.rand(3,5,5)
>>> B=crop_pad_axis(V,-2,0)
```

Mirror volume 3 planes on low side of row axis, crop 2 planes on high side:

```
>>> B=crop_pad_axis(V,(3,-2),1,filltype='mirror')
```

ptypy.utils.array_utils.**crop_pad**(*A*, *hplane_list*, *axes=None*, *cen=None*, *fillpar=0.0*, *filltype='scalar'*)

Crops or pads a volume array *A* with a number of hyperplanes according to parameters in *hplanes* Wrapper for crop_pad_axis.

Parameters

- **A** (*array-like*) – input array of any shape
- **hplane_list** (*array-like, list*) – list of scalars or tupels counting the number of hyperplanes to crop / pad see crop_pad_axis() for detail. If N=len(hplane_list) has less entries than dimensions of A, the last N axes are used
- **axes** (*tuple, list*) – tuple / list of axes to be used for cropping / padding, has to be same length as hplanes
- **cen** – center of array, padding/cropping occurs at cen + A.shape / 2

Returns Cropped or padded array.

Return type array-like

See also:

crop_pad_axis()

Examples

```
>>> import numpy as np
>>> from ptypy.utils import crop_pad
>>> V=np.random.rand(3,5,5)
```


ptypy.utils.array_utils.**mirror**(*A*, *axis=-1*)

Mirrors array *A* along one axis *axis*

Parameters

- **A** (*array-like*) – Input array
- **axis** (*int, optional*) – Axis along which the array will be mirrored

Returns A view to the mirrored array.

Return type array-like

ptypy.utils.array_utils.**crop_pad_symmetric_2d**(*A*, *newshape*, *center=None*)

Crops or pads Array *A* symmetrically along the last two axes *(-2,-1)* around center *center* to a new shape *newshape*.

Parameters

- **A** (*array-like*) – Input array, must be at least two-dimensional.
- **newshape** (*tuple, array_like*) – New shape (for the last two axes). Must have at least 2 entries.
- **center** (*tuple, array_like, optional*) – This coordinate tuple marks the center for cropping / padding If None, `np.array(A.shape[-2:]) // 2` will be the center

Returns

- **out** (*ndarray*) – Cropped or padded array with shape `A.shape[:-2]+newshape[-2:]`
- **center** (*tuple*) – Center in returned array, should be in the actual center of array.

See also:

crop_pad_axis(), *crop_pad()*

ptypy.utils.array_utils.**crop_pad_axis**(*A*, *hplanes*, *axis=-1*, *roll=0*, *fillpar=0.0*, *filltype='scalar'*)

Crops or pads a volume array *A* at beginning and end of axis *axis* with a number of hyperplanes specified by *hplanes*

Parameters

- **A** (*array-like*) – Input array, should be at least one-dimensional
- **hplanes** (*int or tuple of int*) – Number of hyperplanes to add or remove in total. If tuple of 2 ints, this value is interpreted as *(begin, end)*. See the `Note` for more information.
- **axis** (*int, optional*) – axis to be used for cropping / padding
- **roll** (*int,*) – Roll array backwards by this number prior to padding / cropping. The roll is reversed afterwards
- **filltype** (*fillpar,*) – See *pad_lr* for explqanation

Returns Cropped / padded array

Return type array-like

See also:

pad_lr(), *crop_pad()*

Note:

•**if *hplanes* is scalar and negativ :** crops symmetrically, low-index end of axis is preferred if hplane is odd,

- **cen_new** (*array_like*) – Desired new center position in shiftzoomed array

See also:

c_affine_transform()

Returns

Return type Shifted and zoomed array

ptypy.utils.array_utils.**c_zoom**(*c*, **args*, ***kwargs*)
complex input

Zoom an array.

The array is zoomed using spline interpolation of the requested order.

Parameters

- **input** (*ndarray*) – The input array.
- **zoom** (*float or sequence, optional*) – The zoom factor along the axes. If a float, *zoom* is the same for each axis. If a sequence, *zoom* should contain one value for each axis.
- **output** (*ndarray or dtype, optional*) – The array in which to place the output, or the dtype of the returned array.
- **order** (*int, optional*) – The order of the spline interpolation, default is 3. The order has to be in the range 0-5.
- **mode** (*str, optional*) – Points outside the boundaries of the input are filled according to the given mode ('constant', 'nearest', 'reflect' or 'wrap'). Default is 'constant'.
- **cval** (*scalar, optional*) – Value used for points outside the boundaries of the input if `mode='constant'`. Default is 0.0
- **prefilter** (*bool, optional*) – The parameter prefilter determines if the input is pre-filtered with *spline_filter* before interpolation (necessary for spline interpolation of order > 1). If False, it is assumed that the input is already filtered. Default is True.

Returns **zoom** – The zoomed input. If *output* is given as a parameter, None is returned.

Return type ndarray or None

ptypy.utils.array_utils.**c_affine_transform**(*c*, **args*, ***kwargs*)
complex input

Apply an affine transformation.

The given matrix and offset are used to find for each point in the output the corresponding coordinates in the input by an affine transformation. The value of the input at those coordinates is determined by spline interpolation of the requested order. Points outside the boundaries of the input are filled according to the given mode.

Parameters

- **input** (*ndarray*) – The input array.
- **matrix** (*ndarray*) – The matrix must be two-dimensional or can also be given as a one-dimensional sequence or array. In the latter case, it is assumed that the matrix is diagonal. A more efficient algorithms is then applied that exploits the separability of the problem.
- **offset** (*float or sequence, optional*) – The offset into the array where the transform is applied. If a float, *offset* is the same for each axis. If a sequence, *offset* should contain one value for each axis.
- **output_shape** (*tuple of ints, optional*) – Shape tuple.

Pads 4 planes of zeros on the last axis (2 on low side and 2 on high side) and pads 3 planes of zeros on the second last axis (2 on low side and 1 on high side):

```
>>> B=crop_pad(V,[3,4])
```

Also equivalent:

```
>>> B=crop_pad(V,[(2,1),(2,2)])
>>> B=crop_pad(V,[(2,1),(2,2)], axes=[-2,-1],fillpar=0.0,filltype='scalar')
```

None-peripheral cropping /padding:

```
>>> from ptypy.utils import grids
>>> fftshift_grid = grids((6,4),center = 'fft')
>>> cropped_grid=crop_pad(V,[-2,4],cen='fft')
```

Note that cropping/ padding now occurs at the start and end of fourier coordinates useful for cropping /padding high frequencies in fourier space.

ptypy.utils.array_utils.**pad_lr**(*A*, *axis*, *l*, *r*, *fillpar=0.0*, *filltype='scalar'*)
Pads ndarray *A* orthogonal to *axis* with *l* layers (pixels,lines,planes,...) on low side an *r* layers on high side.

Parameters

- **A** (*array-like*) – Input array
- **axis,l,r** (*int*) – Pads orthogonal to *axis* on with *l* layers (pixels,lines, planes,...) on low side an *r* layers on high side.
- **fillpar** (*scalar*) – Scalar fill parameter for `filltype=scalar` fill.
- **filltype** (*str*) –
  - *'scalar'*, uniformly pad with fillpar
  - *'mirror'*, mirror *A*
  - *'periodic'*, periodic fill
  - *'custom'*, pad according arrays found in *fillpar*

Returns Padded array

Return type ndarray

See also:

crop_pad_axis(), *crop_pad()*, *crop_pad_symmetric_2d()*

ptypy.utils.array_utils.**zoom**(*c*, **arg*, ***kwargs*)
Wrapper to dynamically decide whether to use the complex overloaded zoom function or the original one. For parameters see *c_zoom*. Use this function instead of the aforementioned to profit from a faster execution in case of float arrays.

See also:

c_zoom()

ptypy.utils.array_utils.**shift_zoom**(*c*, *zoom*, *cen_old*, *cen_new*, ***kwargs*)
Move array from center *cen_old* to *cen_new* and perform a zoom *zoom*.

This function wraps scipy.ndimage.affine_transfrom or the complex overloaded version *c_affine_transform*, which the user is refered to for keyword args.

Parameters

- **c** (*ndarray*) – Array to shiftzoom. Can be float or complex
- **zoom** (*float*) – Zoom factor
- **cen_old** (*array_like*) – Center in input array *c*

See also:

rebin()

ptypy.utils.embedded_shell module

Tool to embed an ipython shell for debugging.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

ptypy.utils.math_utils module

Numerical util functions.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

ptypy.utils.math_utils.**smooth_step**(*x*, *mfs*)
Smoothed step function with fwhm *mfs* Evaluates the error function *scipy.special.erf*.

ptypy.utils.math_utils.**abs2**(*A*)
Squared absolute value (for real array *A*)

ptypy.utils.math_utils.**norm2**(*A*)
Squared norm

ptypy.utils.math_utils.**norm**(*A*)
Norm.

ptypy.utils.math_utils.**delxb**(*a*, *axis=-1*)
Backward first order derivative for finite difference calculation.

Note: The first element along the derivative direction is set to 0.

Pixel units are used ($\Delta x = \Delta h = 1$).

Parameters

- **a** (*ndarray*) – Input array.
- **axis** (*int, Default=-1, optional*) – Which direction used for the derivative.

Returns **out** – Derived array.

Return type ndarray

ptypy.utils.math_utils.**delxc**(*a*, *axis=-1*)
Central first order derivative for finite difference calculation.

Note: Forward and backward derivatives are used for first and last elements along the derivative direction.

Pixel units are used ($\Delta x = \Delta h = 1$).

Parameters

- **a** (*nd-numpy-array*) – Input array.
- **axis** (*int, Default=-1, optional*) – Which direction used for the derivative.

- **output** (*ndarray or dtype, optional*) – The array in which to place the output, or the dtype of the returned array.
- **order** (*int, optional*) – The order of the spline interpolation, default is 3. The order has to be in the range 0-5.
- **mode** (*str, optional*) – Points outside the boundaries of the input are filled according to the given mode ('constant', 'nearest', 'reflect' or 'wrap'). Default is 'constant'.
- **cval** (*scalar, optional*) – Value used for points outside the boundaries of the input if `mode='constant'`. Default is 0.0
- **prefilter** (*bool, optional*) – The parameter prefilter determines if the input is pre-filtered with *spline_filter* before interpolation (necessary for spline interpolation of order > 1). If False, it is assumed that the input is already filtered. Default is True.

Returns **affine_transform** – The transformed input. If *output* is given as a parameter, None is returned.

Return type ndarray or None

ptypy.utils.array_utils.**rebin**(*a*, **args*, ***kwargs*)
Rebin ndarray data into a smaller ndarray of the same rank whose dimensions are factors of the original dimensions.

Note: eg. An array with 6 columns and 4 rows can be reduced to have 6,3,2 or 1 columns and 4,2 or 1 rows.

Parameters

- **a** (*ndarray*) – Input array.
- **axis** (*int, Default=-1, optional*) – The laplacian is computed along the provided axis or list of axes, or all axes if None

Returns **out** – Rebined array.

Return type ndarray

Examples

```
>>> import ptypy
>>> import numpy as np
>>> a=np.random.rand(6,4)
>>> b=ptypy.utils.rebin(a,3,2)
a.reshape(args[0],factor[0],args[1],factor[1],).sum(1).sum(2)*( 1./factor[0]/factor[1
>>> a2=np.random.rand(6)
>>> b2=ptypy.utils.rebin(a2,2)
a.reshape(args[0],factor[0],).sum(1)*( 1./factor[0])
```

ptypy.utils.array_utils.**rebin_2d**(*A*, *rebin=1*)
Rebins array *A* symmetrically along the last 2 axes with factor *rebin*.

Parameters

- **A** (*array-like*) – input array, must be at least two-dimensional.
- **rebin** (*int*) – rebin factor, `rebin=2` means that a square of 4 pixels will be binned into one.

Returns **out** – rebinned array. Note that data type casting follows numpy rules, so that boolean arrays are converted to int.

Return type ndarray

ptypy.utils.math_utils.**gf**(*c*, **arg*, ***kwargs*)
Wrapper for scipy.ndimage.gaussian_filter, that determines whether original or the complex function shall be used.

See also:

c_gf()

ptypy.utils.math_utils.**cabs2**(*A*)
Squared absolute value (for complex array *A*)

ptypy.utils.math_utils.**gf_2d**(*c*, *sigma*, ***kwargs*)
Gaussian filter along the last 2 axes

See also:

gf(), *c_gf()*

ptypy.utils.math_utils.**c_gf**(*c*, **args*, ***kwargs*)
complex input

Multidimensional Gaussian filter.

Parameters

- **input** (*array_like*) – Input array to filter.
- **sigma** (*scalar or sequence of scalars*) – Standard deviation for Gaussian kernel. The standard deviations of the Gaussian filter are given for each axis as a sequence, or as a single number, in which case it is equal for all axes.
- **order** (*{0, 1, 2, 3} or sequence from same set, optional*) – The order of the filter along each axis is given as a sequence of integers, or as a single number. An order of 0 corresponds to convolution with a Gaussian kernel. An order of 1, 2, or 3 corresponds to convolution with the first, second or third derivatives of a Gaussian. Higher order derivatives are not implemented
- **output** (*array, optional*) – The *output* parameter passes an array in which to store the filter output.
- **mode** (*{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional*) – The *mode* parameter determines how the array borders are handled, where *cval* is the value when mode is equal to 'constant'. Default is 'reflect'
- **cval** (*scalar, optional*) – Value to fill past edges of input if *mode* is 'constant'. Default is 0.0

Returns **gaussian_filter** – Returned array of same shape as *input*.

Return type ndarray

Notes

The multidimensional filter is implemented as a sequence of one-dimensional convolution filters. The intermediate arrays are stored in the same data type as the output. Therefore, for output types with a limited precision, the results may be imprecise because intermediate results may be stored with insufficient precision.

ptypy.utils.misc module

Miscellaneous utility functions with little dependencies from other modules

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

Returns **out** – Derived array.

Return type nd-numpy-array

ptypy.utils.math_utils.**delxf**(*a*, *axis=-1*, *out=None*)
Forward first order derivative for finite difference calculation.

Note: The last element along the derivative direction is set to 0.

Pixel units are used ($\Delta x = \Delta h = 1$).

Parameters

- **a** (*ndarray*) – Input array.
- **axis** (*int, Default=-1, optional*) – Which direction used for the derivative.
- **out** (*ndarray, Default=None, optional*) – Array in wich the resault is written (same size as `a`).

Returns **out** – Derived array.

Return type ndarray

ptypy.utils.math_utils.**ortho**(*modes*)
Orthogonalize the given list of modes or ndarray along first axis. **specify procedure**

Parameters **modes** (*array-like or list*) – List equally shaped arrays or array of higher dimension

Returns

- **amp** (*vector*) – relative power of each mode
- **nplist** (*list*) – List of modes, sorted in descending order

ptypy.utils.math_utils.**gauss_fwhm**(*x*, *fwhm=1.0*, *off=0.0*)
Evaluates gaussian with full width half maximum

Parameters

- **x** (*ndarray*) – input array
- **fwhm** (*float,optional*) – Full width at half maximum
- **off** (*float, optional*) – Offset / shift

See also:

gaussian()

ptypy.utils.math_utils.**gaussian**(*x*, *std=1.0*, *off=0.0*)
Evaluates gaussian standard normal

$$g(x) = \frac{1}{\mathrm{std}\sqrt{2\pi}} \exp\left(-\frac{(x-\mathrm{off})^2}{2\mathrm{std}^2}\right)$$

Parameters

- **x** (*ndarray*) – input array
- **std** (*float,optional*) – Standard deviation
- **off** (*float, optional*) – Offset / shift

See also:

gauss_fwhm(), *smooth_step()*

```
>>> expect3( (1.0, 3.0,4.0) )
array([ 1.,  3.,  4.])
```

ptypy.utils.misc.**keV2m**(*keV*)
Convert photon energy in keV to wavelength (in vacuum) in meters.

ptypy.utils.misc.**keV2nm**(*keV*)
Convert photon energy in keV to wavelength (in vacuum) in nanometers.

ptypy.utils.misc.**nm2keV**(*nm*)
Convert wavelength in nanometers to photon energy in keV.

ptypy.utils.misc.**clean_path**(*filename*)
Makes path absolute and create all sub directories if needed.

Parameters **filename** (*str*) – A filename.

ptypy.utils.misc.**unique_path**(*filename*)
Makes path absolute and unique

Parameters **filename** (*str*) – A filename.

ptypy.utils.parallel module

Utility functions and classes to support MPI computing.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

ptypy.utils.parallel.**barrier**()
Wrapper for comm.Barrier.

class ptypy.utils.parallel.**LoadManager**
Bases: object

Keeps track of the amount of data managed by each process and helps keeping it balanced.

Note: A LoadManager should always be given the full range of ids, as the *all-knowing* power stems from the fact that it always knows how many processes there are, what rank it has and the full range of ids to distribute. Hence, please *"assign"* always the same list across all nodes.

---

assign(*idlist=None*)
Subdivide the provided list of ids into contiguous blocks to balance the load among the processes.

The elements of idlist are used as keys to subsequently identify the rank of a given id, through the dict self.rank_of. No check is done whether ids passed are unique or even hashable (they better are)

If idlist is None, increase by one the load of the least busy process. *rank_of* is not updated in this case.

Parameters **idlist** (*list*) – List of objects that can also be keys in a *dict*, i.e. hashable objects.

Returns **R** – A nested list, (a list of lists) such that R[rank]=list of elements of *idlist* managed by process of given *rank*.

Return type list

load = None
Represents the number of elements assigned to this process.

rank_of = None
Rank of specific *id*, i.e. element of *idlist*.

license GPLv2, see LICENSE for details.

ptypy.utils.misc.**str2int**(*A*)
Transforms numpy array *A* of strings to np.uint8 and back

Examples

```
>>> from ptypy.utils import str2int
>>> A=np.array('hallo')
>>> A
array('hallo', dtype='|S5')
>>> str2int(A)
array([104,  97, 108, 108, 111], dtype=uint8)
>>> str2int(str2int(A))
array('hallo', dtype='|S5')
```

ptypy.utils.misc.**str2range**(*s*)
Generates an index list from string input *s*

Examples

```
>>> # Same as range(1,4,2)
>>> str2range('1:4:2')
>>> # Same as range(1,2)
>>> str2range('1')
```

ptypy.utils.misc.**complex_overload**(*func*)
Overloads function specified only for floats in the following manner

$$\text{complex\_overload}\{f\}(c) = f(\Re(c)) + \mathrm{i}f(\Im(c))$$

ptypy.utils.misc.**expect2**(*a*)
Generates 1d numpy array with 2 entries generated from multiple inputs (tuples, arrays, scalars). Main puprose of this function is to circumvent debugging of input as very often a 2-vector is requred in scripts

Examples

```
>>> from ptypy.utils import expect2
>>> expect2( 3.0 )
array([ 3.,  3.])
>>> expect2( (3.0,4.0) )
array([ 3.,  4.])
>>> expect2( (1.0, 3.0,4.0) )
array([ 1.,  3.])
```

ptypy.utils.misc.**expect3**(*a*)
Generates 1d numpy array with 3 entries generated from multiple inputs (tuples, arrays, scalars). Main puprose of this function is to circumvent debugging of input.

Examples

```
>>> from ptypy.utils import expect3
>>> expect3( 3.0 )
array([ 3.,  3.,  3.])
>>> expect3( (3.0,4.0) )
array([ 3.,  4.,  4.])
```

ptypy.utils.parallel.**bcast_dict**(*dct*, *keys='all'*, *source=0*)
Broadcasts or scatters a dict *dct* from `rank==source`. If value is a numpy ndarray, *comm.Bcast* is used instead *comm.bcast*, such that transfer is accelerated.

Fills dict *dct* in place for receiving nodes, although this is a bit inconsistent compared to *gather_dict*

Parameters

- **keys** (*list or 'all'*) – List of keys whose values are accepted at each node. In case of `keys=all`, every node accepts all items and *bcast_dict* acts as broadcast.
- **source** (*int*) – Rank of node / process which broadcasts / scatters.

Returns **dct** – A smaller dictionary with values to *keys* if that key was in source dictionary.

Return type dict

Note: There is no guarantee that each *key,value* pair is accepted at other nodes, except for `keys='all'`. Also in this implementation the input *dct* from source is *completely* transmitted to every node, potentially creating a large overhead for may nodes and huge dictionarys

Deleting reference in input dictionary may result in data loss at `rank==source`

See also:

gather_dict()

ptypy.utils.parallel.**gather_dict**(*dct*, *target=0*)
Gathers broadcasted or scattered dict *dct* at rank *target*.

Parameters

- **dct** (*dict*) – Input dictionary. Remains unaltered
- **target** (*int*) – Rank of process where the *dct*'s are gathered

Returns **out** – Gathered dict at `rank==target`, Empty dict at `rank!=target`

Return type dict

Note: If the same *key* exists on different nodes, the corresponding values will be consecutively overidden in the order of the ranks at the gathering node without complain or notification.

See also:

`broadcast_dict()`

ptypy.utils.parallel.**MPIrand_normal**(*loc=0.0*, *scale=1.0*, *size=1*)
Wrapper for `np.random.normal` for same random sample across all nodes. *See numpy/scipy documentation below.*

normal(loc=0.0, scale=1.0, size=None)

Draw random samples from a normal (Gaussian) distribution.

The probability density function of the normal distribution, first derived by De Moivre and 200 years later by both Gauss and Laplace independently [1], is often called the bell curve because of its characteristic shape (see the example below).

The normal distributions occurs often in nature. For example, it describes the commonly occurring distribution of samples influenced by a large number of tiny, random disturbances, each with its own unique distribution [2].

Parameters

- **loc** (*float*) – Mean ("centre") of the distribution.

[1] P. R. Peebles Jr., "Central Limit Theorem" in "Probability, Random Variables and Random Signal Principles", 4th ed., 2001, pp. 51, 51, 125.

reset()
Resets *LoadManager* to initial state.

ptypy.utils.parallel.**allreduce**(*a*, *op=None*)
Wrapper for comm.Allreduce, always in place.

Parameters

- **a** (*numpy-ndarray*) – The array to operate on.
- **op** (*operation*) – MPI operation to execute. None or one of MPI.BAND, MPI.BOR, MPI.BXOR, MPI.LAND, MPI.LOR, MPI.LXOR, MPI.MAX, MPI.MAXLOC, MPI.MIN, MPI.MINLOC, MPI.OP_NULL, MPI.PROD, MPI.REPLACE or MPI.SUM. If None, uses MPI.SUM.

Note: *Explanation* : If process #1 has ndarray `a` and process #2 has ndarray `b`. After calling allreduce, the new arrays after allreduce are `a'=op(a,b)` and `b'=op(a,b)` on process #1 and #2 respectively

---

ptypy.utils.parallel.**send**(*data*, *dest=0*, *tag=0*)
Wrapper for *comm.Send* and *comm.send*. If data is a *numpy.ndarray*, a header will be sent first with *comm.send* that contains information on array shape and data type. Afterwards the array will be sent with *comm.Send*. If data is not a *numpy.ndarray*, the whole object will be pickled and sent with *comm.send* in one go.

Parameters

- **data** (*ndarray or other*) – Object to send
- **dest** (*int*) – The rank of the destination node / process. Defaults to 0 (master).
- **tag** (*int*) – Defaults to 0.

See also:

receive(), *bcast()*

ptypy.utils.parallel.**receive**(*source=None*, *tag=0*)
Wrapper for *comm.Recv*. Probes first with *comm.recv*. If the unpickled is a 3-tuple with first entry=='npy', prepares buffer and waits with *comm.Recv*

Parameters

- **source** (*int or None*) – The rank of the node / process sending data. If None, this is set to MPI.ANY_SOURCE
- **tag** (*int*) – Not really useful here - defaults to *0* all the time

Returns **out**

Return type ndarray or other

ptypy.utils.parallel.**bcast**(*data*, *source=0*)
Wrapper for *comm.bcast* and *comm.Bcast*. If data is a *numpy.ndarray*, a header will be broadcasted first with *comm.bcast* that contains information on array shape and data type. Afterwards the array will be sent with *comm.Bcast*. If data is not a *numpy.ndarray*, the whole object will be pickled and broadcasted with *comm.bcast* in one go.

Parameters

- **data** (*ndarray or other*) – Object to send
- **source** (*int*) – The rank of the source node / process. Defaults to 0 (master).
- **tag** (*int*) – Defaults to 0.

See also:

receive(), *send()*

- **low** (*float, optional*) – Lower boundary of the output interval. All values generated will be greater than or equal to low. The default value is 0.
- **high** (*float*) – Upper boundary of the output interval. All values generated will be less than high. The default value is 1.0.
- **size** (*int or tuple of ints, optional*) – Shape of output. If the given size is, for example, (m,n,k), m*n*k samples are generated. If no shape is specified, a single sample is returned.

Returns **out** – Drawn samples, with shape *size*.

Return type ndarray

See also:

randint() Discrete uniform distribution, yielding integers.

random_integers() Discrete uniform distribution over the closed interval `[low, high]`.

random_sample() Floats uniformly distributed over `[0, 1)`.

random() Alias for *random_sample*.

rand() Convenience function that accepts dimensions as input, e.g., `rand(2,2)` would generate a 2-by-2 array of floats, uniformly distributed over `[0, 1)`.

Notes

The probability density function of the uniform distribution is

$$p(x) = \frac{1}{b-a}$$

anywhere within the interval `[a, b)`, and zero elsewhere.

Examples

Draw samples from the distribution:

```
>>> s = np.random.uniform(-1,0,1000)
```

All values are within the given interval:

```
>>> np.all(s >= -1)
True
>>> np.all(s < 0)
True
```

Display the histogram of the samples, along with the probability density function:

```
>>> import matplotlib.pyplot as plt
>>> count, bins, ignored = plt.hist(s, 15, normed=True)
>>> plt.plot(bins, np.ones_like(bins), linewidth=2, color='r')
>>> plt.show()
```

ptypy.utils.parallel.**MPInoise2d**(*sh*, *rms=1.0*, *mfs=2*, *rms_mod=None*, *mfs_mod=2*)

Creates complex-valued statistical noise in the shape of *sh* consistent across all nodes.

Parameters

- **sh** (*tuple*) – Output shape.
- **rms** (*float or None*) – Root mean square of noise in phase. If None, only ones are returned.

- **scale** (*float*) – Standard deviation (spread or "width") of the distribution.
- **size** (*tuple of ints*) – Output shape. If the given shape is, e.g., `(m, n, k)`, then `m * n * k` samples are drawn.

See also:

scipy.stats.distributions.norm() probability density function, distribution or cumulative density function, etc.

Notes

The probability density for the Gaussian distribution is

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

where μ is the mean and σ the standard deviation. The square of the standard deviation, σ^2, is called the variance.

The function has its peak at the mean, and its "spread" increases with the standard deviation (the function reaches 0.607 times its maximum at $x + \sigma$ and $x - \sigma$ [2]). This implies that *numpy.random.normal* is more likely to return samples lying close to the mean, rather than those far away.

References

Examples

Draw samples from the distribution:

```
>>> mu, sigma = 0, 0.1 # mean and standard deviation
>>> s = np.random.normal(mu, sigma, 1000)
```

Verify the mean and the variance:

```
>>> abs(mu - np.mean(s)) < 0.01
True
```

```
>>> abs(sigma - np.std(s, ddof=1)) < 0.01
True
```

Display the histogram of the samples, along with the probability density function:

```
>>> import matplotlib.pyplot as plt
>>> count, bins, ignored = plt.hist(s, 30, normed=True)
>>> plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) *
...                np.exp( - (bins - mu)**2 / (2 * sigma**2) ),
...          linewidth=2, color='r')
>>> plt.show()
```

ptypy.utils.parallel.**MPIrand_uniform**(*low=0.0*, *high=1.0*, *size=1*)

Wrapper for `np.random.uniform` for same random sample across all nodes. *See numpy/scipy documentation below.*

uniform(low=0.0, high=1.0, size=1)

Draw samples from a uniform distribution.

Samples are uniformly distributed over the half-open interval `[low, high)` (includes low, but excludes high). In other words, any value within the given interval is equally likely to be drawn by *uniform*.

Parameters

- **out** (Param) – The Param structure built from a. No copy is done if the input is already a Param.

ptypy.utils.plot_client module

Client tools for plotting.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

class ptypy.utils.plot_client.**MPLClient**(*client_pars=None*, *autoplot_pars=None*, *layout_pars=None*, *in_thread=False*, *is_slave=False*)

Bases: *ptypy.utils.plot_client.MPLplotter*

DEFAULT = Param({'pr': Param({'layers': None, 'clims': [None, [-3.141592653589793, 3.141592653589793

loop_plot()
Plot forever.

class ptypy.utils.plot_client.**MPLplotter**(*pars=None*, *probes=None*, *objects=None*, *runtime=None*, *in_thread=False*)

Bases: object

Plotting Client for Ptypy, using matplotlib.

Create a client and attempt to connect to a running reconstruction server.

DEFAULT = Param({'pr': Param({'layers': None, 'clims': [None, [-3.141592653589793, 3.141592653589793

static create_plot_from_tile_list(*fignum=1*, *num_shape_list=[(4, (2, 2))]*, *figsize=(8, 8)*)

draw()

plot_all(*blocking=False*)

plot_error()

plot_storage(*storage*, *plot_pars*, *title=''*, *typ='obj'*)

save(*pattern*, *count=0*)

update_plot_layout(*plot_template=None*)
Generate the plot layout.

class ptypy.utils.plot_client.**PlotClient**(*client_pars=None*, *in_thread=False*)

Bases: object

A client that connects and continually gets the data required for plotting. Note: all data is transferred as soon as the server provides it. This might be a waste of bandwidth if all that is required is a client that plots "on demand"...

This PlotClient doesn't actually plot.

Create a client and attempt to connect to a running reconstruction server.

ACTIVE = 1

DATA = 2

STOPPED = 0

disconnect()
Disconnect from reconstruction server.

get_data()
Thread-safe way to copy data buffer. :return:

- **mfs** (*float*) – Minimum feature size [in pixel] of noise in phase.
- **rms_mod** (*float or None*) – Root mean square of noise in amplitude / modulus.
- **mfs_mod** – Minimum feature size [in pixel] of noise in amplitude / modulus.

Returns **noise** – Numpy array filled with noise

Return type ndarray

See also:

MPIrand_uniform(), *MPIrand_normal()*

ptypy.utils.parameters module

Parameter definition.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

class ptypy.utils.parameters.**Param**(*__d__=None*, ***kwargs*)

Bases: dict

Convenience class: a dictionary that gives access to its keys through attributes.

Note: dictionaries stored in this class are also automatically converted to Param objects: >>> p = Param() >>> p.x = {} >>> p Param({})

While dict(p) returns a dictionary, it is not recursive, so it is better in this case to use p.todict(). However, p.todict does not check for infinite recursion. So please don't store a dictionary (or a Param) inside itself.

BE: Please note also that the recursive behavior of the update function will create new references. This will lead inconsistency if other objects refer to dicts or Params in the updated Param instance.

A Dictionary that enables access to its keys as attributes. Same constructor as dict.

__dir__()
Defined to include the keys when using dir(). Useful for tab completion in e.g. ipython. If you do not wish the dict key's be displayed as attributes (although they are still accessible as such) set the class attribute *_display_items_as_attributes* to False. Default is True.

copy(*depth=0*)

Returns Param A (recursive) copy of P with depth *depth*

update(*__d__=None*, *in_place_depth=0*, *Convert=False*, ***kwargs*)
Update Param - almost same behavior as dict.update, except that all dictionaries are converted to Param if *Convert* is set to True, and update may occur in-place recursively for other Param instances that self refers to.

Parameters

- **Convert** (*bool*) – If True, convert all dict-like values in self also to Param. *WARNING* This mey result in misdirected references in your environment
- **in_place_depth** (*int*) – Counter for recursive in-place updates If the counter reaches zero, the Param to a key is replaced instead of updated

ptypy.utils.parameters.**asParam**(*obj*)
Convert the input to a Param.

Parameters

- **a** (*dict_like*) – Input structure, in any format that can be converted to a Param.
- **Returns** –

- **fignum** (*int, optional*) – Number of the figure.
- **slices** (*tuple of slices or string of numpy index expression, optional*) – Determines what part of Storage buffer is displayed, i.e. which layers and which region-of-interest. Layers are subplotted horizontically next to each other. Figsize is (6,6*layers)
- **modulus** (*str, optional*) – One of *sqrt*, *log* or *linear* to apply to modulus of array buffer. Useful to reduce dynamic range for diffraction images.
- **si_axes** (*str, optional*) – One of 'x','xy','y' or None, determins which axes display length units instead of pixel units
- **mask** (*ndarray, scalar or None*) – Bool array of valid pixel data for rescaling.

Returns **fig**

Return type maplotlib.pyplot.figure

See also:

imsave(), *Storage*

ptypy.utils.plot_utils.**imsave**(*a*, *filename=None*, *vmin=None*, *vmax=None*, *cmap=None*)

Take array *a* and transform to *PIL.Image* object that may be used by *pyplot.imshow* for example. Also save image buffer directly without the sometimes unnecessary Gui-frame and overhead.

Parameters

- **a** (*ndarray*) – Two dimensional array. Can be complex, in which case the amplitude will be optionally clipped by *vmin* and *vmax* if set.
- **filename** (*str, optionsl*) – File path to save the image buffer to. Use '*.png' or '*.png' as image formats.
- **vmin, vmax** (*float, optional*) – Value limits ('clipping') to fit the color scale. If not set, color scale will span from minimum to maximum value in array
- **cmap** (*str, optional*) – Name of the colormap for colorencoding.

Returns **im** – a *PIL.Image* object.

Return type PIL.Image

See also:

complex2rgb()

Examples

```
>>> from ptypy.utils import imsave
>>> from matplotlib import pyplot as plt
>>> from ptypy.resources import flower_obj
>>> a = flower_obj(512)
>>> pil = imsave(a)
>>> plt.imshow(pil)
>>> plt.show()
```

converts array a into, and returns a PIL image and displays it.

```
>>> pil = imsave(a, /tmp/moon.png)
```

returns the image and also saves it to filename

```
>>> imsave(a, vmin=0, vmax=0.5)
```

clips the array to values between 0 and 0.5.

start()
This needs to be run for the thread to initialize.

status
True only if new data were acquired since last time checked.

stop()
Stop loop plot

ptypy.utils.plot_client.**spawn_MPLClient**(*client_pars*, *autoplot_pars*)

A function that creates and runs a silent instance of MPLClient.

ptypy.utils.plot_utils module

Plotting utilities.

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

ptypy.utils.plot_utils.**pause**(*timeout=-1*, *message=None*)

Pause the execution of a script while leaving matplotlib figures responsive. *Gui aware*

Parameters

- **timeout** (*float, optional*) – By default, execution is resumed only after hitting return. If timeout >= 0, the execution is resumed after timeout seconds.
- **message** (*str, optional*) – Message to diplay on terminal while pausing

ptypy.utils.plot_utils.**rmphaseramp**(*a*, *weight=None*, *return_phaseramp=False*)

Attempts to remove the phase ramp in a two-dimensional complex array `a`.

Parameters

- **a** (*ndarray*) – Input image as complex 2D-array.
- **weight** (*ndarray, str, optional*) – Pass weighting array or use `'abs'` for a modulus-weighted phaseramp and `Non` for no weights.
- **return_phaseramp** (*bool, optional*) – Use True to get also the phaseramp array `p`.

Returns

- **out** (*ndarray*) – Modified 2D-array, `out=a*p`
- **p** (*ndarray, optional*) – Phaseramp if `return_phaseramp = True`, otherwise omitted

Examples

```
>>> b = rmphaseramp(image)
>>> b, p = rmphaseramp(image , return_phaseramp=True)
```

ptypy.utils.plot_utils.**plot_storage**(*S*, *fignum=100*, *modulus='linear'*, *slices=(slice(None, 1, None), slice(None, None, None), slice(None, None, None))*, *si_axes='x'*, *mask=None*, ***kwargs*)

Quickly display the data buffer of a *Storage* instance.

Keyword arguments are those of `PtyAxis`

Parameters

- **S** (Storage) – Storage instance

ptypy.utils.scripts.**hdr_image**(*img_list, exp_list, thresholds=[3000, 50000], dark_list=[], avg_type='highest', mask_list=[], ClipLongestExposure=False, ClipShortestExposure=False*)

Generate a high dynamic range image from a list of images *img_list* and corresponding exposure information in *exp_list*.

Parameters

- **img_list** (*list*) – Sequence of images (as 2d np.ndarray)
- **exp_list** (*list of float*) – Associated exposures to each element of above sequence *img_list*
- **thresholds** (*list, 2-tuple*) – Tuple of lower limit (noise floor) and upper limit (over-exposure) in the images.
- **dark_list** (*list*) – Single frame or sequence of dark images (as 2d np.array) of the same length as *img_list*. These frames are used for dark-field correction
- **avg_type** (*str*) – Type of combining all valid pixels:
  - *'highest'*, the next longest exposure is used to replace overexposed pixels.
  - *<other_string>*, overexposed pixels are raplaced by the pixel average of all other images with valid pixel values for that pixel.
- **mask_list** (*list*) – Single frame or sequence of 2d np.array. Provide additional masking (dead pixels, hot pixels)
- **ClipLongestExposure** (*bool*) – If True, also mask the noise floor in the longest exposure.
- **ClipShortestExposure** (*bool*) – if True, also mask the overexposed pixels in the shortest exposure.

Returns **out** – Combined hdr-image with shape of one of the frame in *img_list*

Return type ndarray

Examples

```
>>> from ptypy import io
>>> dark_list,meta=io.image_read('/path/to/dark/images/ccd*.raw')
>>> img_list,meta=io.image_read('/path/to/images/ccd*.raw')
>>> exp_list=[meta[j]['exposure_time__key'] for j in range(len(meta))]
>>> hdr,masks = hdr_image(img_list, exp_list, dark_list=dark_list)
```

ptypy.utils.scripts.**diversify**(*A, noise=None, shift=None, power=1.0*)

Add diversity to 3d numpy array *A*, *acts in-place*.

Parameters

- **noise** (*2-tuple or 4-tuple*) – For detailed description see *ptypy.utils.parallel.MPInoise2d*
- **power** (*float, tuple*) – Relative power of layers with respect to the first (0) layer. Can be scalar or tuple / array
- **shift** (*float, tuple*) – Relative shift of layers with respect to the first (0) layer. Can be scalar or tuple / array **not implemented yet**

See also:

ptypy.utils.parallel.MPInoise2d()

ptypy.utils.scripts.**cxro_iref**(*formula, energy, density=-1, npts=100*)

Query CXRO database for index of refraction values for a solid


```
>>> imsave(abs(a), cmap='gray')
```

uses a matplotlib colormap with name 'gray'

ptypy.utils.plot_utils.**imload**(*filename*)

Load an image and returns a numpy array. *May get deleted*

ptypy.utils.plot_utils.**complex2hsv**(*cin, vmin=None, vmax=None*)

Transforms a complex array into an RGB image, mapping phase to hue, amplitude to value and keeping maximum saturation.

Parameters

- **cin** (*ndarray*) – Complex input. Must be two-dimensional.
- **vmin, vmax** (*float*) – Clip amplitude of input into this interval.

Returns **rgb** – Three dimensional output.

Return type ndarray

See also:

complex2rgb(), *hsv2rgb()*, *hsv2complex()*

ptypy.utils.plot_utils.**complex2rgb**(*cin, **kwargs*)

Executes *complex2hsv* and then *hsv2rgb*

See also:

complex2hsv(), *hsv2rgb()*, *rgb2complex()*

ptypy.utils.plot_utils.**hsv2rgb**(*hsv*)

HSV (Hue,Saturation,Value) to RGB (Red,Green,Blue) transformation.

Parameters **hsv** (*array-like*) – Input must be two-dimensional. **First** axis is interpreted as hue,saturation,value channels.

Returns **rgb** – Three dimensional output. **Last** axis is interpreted as red, green, blue channels.

Return type ndarray

See also:

complex2rgb(), *complex2hsv()*, *rgb2hsv()*

ptypy.utils.plot_utils.**rgb2complex**(*rgb*)

Reverse to *complex2rgb*

ptypy.utils.plot_utils.**rgb2hsv**(*rgb*)

Reverse to *hsv2rgb*

ptypy.utils.plot_utils.**hsv2complex**(*cin*)

Reverse to *complex2hsv*

ptypy.utils.plot_utils.**franzmap**()

Set the default colormap to Franz's map and apply to current image if any.

ptypy.utils.scripts module

longer script-like functions

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

Parameters

- **listoffiles** (*list of str*) – A list of paths to files. Each file will be reinterpreted as a collection files in the same directory, e.g. only the first file in a series needs to be selected. All series for which a first file was given will be concatenated in the movie. May also be a textfile with a listing of frames in which case the creation of an intermediate file of frame paths will be skipped.
- **framefile** (*str*) – Filepath, the respective file will be created to store a list of all frames. This file will be used by mencoder later.
- **fps** (*scalar*) – Frames per second in final movie
- **bitrate** (*int*) – Encoding detail, determines video quality
- **codec** (*str*) – Defines the codec to Use
- **Encode** (*bool*) – If True, video will be encoded calling mencocder If False, mencoder will not be called, but the necessary command expression will be returned instead. Very well suited for a dry-run
- **RemoveImages** (*bool*) – If True, all images refered to by framefile are deleted except for the last frame.

Returns **cmd** – Command string for the shell to encode the video later manually.

Return type str

Examples

```
>>> from ptypy.utils import png2mpg
>>> png2mpg(['/path/to/image_000.png'])
```

The following happens: 1) search for files similar to image_*.png in '/path/to/' 2) found files get listed in a file '/path/to/frames.txt' 3) calls mencoder to use that file to encode a movie with the default args. 4) movie is in the same folder as 'frames.txt' and is called 'frames.mpg'

```
>>> png2mpg(['/path1/to/imageA_040.png','/path2/to/imageB_001.png'],framefile='./bana
```

Generates list file 'banana_text' in current folder. The list file contains in order every path compatible with the wildcards '/path1/to/imageA_*.png' and '/path2/to/imageB_*.png'

```
>>> str=png2mpg(['/path/to/image_000.png'],Encode=False)
```

Returns encoder argument string. Use os.system(encoderstring) for encoding manually later

ptypy.utils.scripts.**mass_center**(*A*, *axes=None*)

Calculates mass center of n-dimensional array *A* along tuple of axis *axes*.

Parameters

- **A** (*ndarray*) – input array
- **axes** (*list,tuple*) – Sequence of axes that contribute to distributed mass. If axes==None, all axes are considered.

Returns **mass** – Center of mass in pixel for each *axis* selected.

Return type 1darray

ptypy.utils.scripts.**phase_from_dpc**(*dpc_row*, *dpc_col*)

Implements fourier integration method for two diffential quantities.

Assumes 2 arrays of N-dimensions who contain differential quantities in X and Y, i.e. the LAST two dimensions (-2 & -1) in this case.

Parameters **dpc_col** (*dpc_row,*) – Differential information along 2nd last dimension and last dimension respectively. Must be the same shape

Parameters

- **formula** (*str*) – String representation of the Formula to use.
- **energy** (*float or (float,float)*) – Either a single energy (in eV) or the minimum/maximum bounds
- **density** (*None or float, optional*) – Density of the material [g/ccm]. If None or <0 the regular density at ambiente temperatures is used.
- **npts** (*int, optional*) – Number of points between the min and max energies.

Returns **energy, delta, beta** – Energy used and the respective *delta* and *beta* values.

Return type scalar or vector

ptypy.utils.scripts.**xradia_star**(*sh*, *spokes=48*, *std=0.5*, *minfeature=5*, *ringfact=2*, *rings=4*, *contrast=1.0*, *Fast=False*)

Creates an Xradia-like star pattern on an array of shape *sh* Works superb as test pattern in ptychography

requires scipy

Parameters

- **std** (*float*) – "Resolution" of xradia star, i.e. standard deviation of the errorfunction used for smoothing the edges (in pixel).
- **spokes** (*int, optional*) – Number of spokes
- **minfeature** (*float, optional*) – Spoke width at the smallest (inner) tip (in pixel).
- **ringfact** (*float*) – Increase in featuresize from ring to ring (factor). Determines position of the rings.
- **rings** (*int*) – Number of rings with spokes.
- **contrast** (*float*) – Minimum contrast, set to zero for a gradual increase of the profile from zero to 1, set to 1 for no gradient in profile
- **Fast** (*bool*) – If set to False, the error function is evaluated at the edges. Prefered choice when using fft, as edges are less prone to antiaaliasing in this case. If set to True, simple boolean comparison will be used instead to draw the edges and the result is later smoothed with a gaussian filter. Prefered choice for unpatient users, as this choice is roughly a factor 2 faster for larger arrays

Examples

```
>>> from ptypy.utils import xradia_star
>>> # Base configuration
>>> X1 = xradia_star(1024)
>>> # Few spokes single ring
>>> X2 = xradia_star(1024, 12, std=4, rings=1, minfeature=10, ringfact=10)
>>> # Very fine plus gradient
>>> X3 = xradia_star(1024, 64, std = 0.2, rings = 10, minfeature=1, contrast=0)
>>> from matplotlib import pyplot as plt
>>> ax=plt.subplot(131)
>>> ax.imshow(X1,cmap='gray')
>>> ax=plt.subplot(132)
>>> ax.imshow(X2,cmap='gray')
>>> ax=plt.subplot(133)
>>> ax.imshow(X3,cmap='gray')
>>> plt.show()
```

ptypy.utils.scripts.**png2mpg**(*listoffiles*, *framefile='frames.txt'*, *fps=5*, *bitrate=2000*, *codec='wmv2'*, *Encode=True*, *RemoveImages=False*)

Makes a movie (*.mpg) from a collection of *.png or *.jpeg frames. *Requires* binary of **mencoder** installed on system

ptypy.utils.validator module

Parameter validation. This module parses the file `resources/parameters_descriptions.csv` to extract the parameter defaults for PtyPy. It saves all parameters in the form of a *PDesc* object, which are flat listed in *parameter_descriptions* or in *entry_points_dct*, which only contains parameters with subparameters (children).

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

`ptypy.utils.validator.`**`create_default_template`**(*filename=None*, *user_level=0*, *doc_level=2*)

Creates a (descriptive) template for ptypy.

Parameters

- **filename** (*str*) – python file (.py) to generate, will be overriden if it exists
- **user_level** (*int*) – Filter parameters to display on those with less/equal user level
- **doc_level** (*int*) –
  - if 0, no comments.
  - if 1, *short_doc* as comment in script
  - if >2, *long_doc* and *short_doc* as comment in script

`ptypy.utils.validator.`**`make_sub_default`**(*entry_point*, *depth=1*)

Creates a default parameter structure, from the loaded parameter descriptions in this module

Parameters

- **entry_point** (*str*) – The node in the default parameter file
- **depth** (*int*) – The depth in the structure to which all sub nodes are expanded All nodes beyond depth will be returned as empty *Param* structure.

Returns **pars** – A parameter branch.

Return type *Param*

Examples

```
>>> from ptypy.utils import validator
>>> print validator.make_sub_default('.io')
```

`ptypy.utils.validator.`**`validate`**(*pars*, *entry_point*, *walk=True*, *raisecodes=[0, 4]*)

Check that the parameter structure *pars* matches the documented constraints at the given entry_point.

The function raises a RuntimeError if one of the code in the list *raisecodes* has been found. If raisecode is empty, the function will always return successfully but problems will be logged using logger.

Parameters

- **pars** (Param) – A parameter set to validate
- **entry_point** (*str*) – The node in the parameter structure to match to.
- **walk** (*bool*) – If `True` (*default*), navigate sub-parameters.
- **raisecodes** (*list*) – List of codes that will raise a RuntimeError.

class `ptypy.utils.validator.`**`PDesc`**(*description*, *parent=None*)

Bases: `object`

Small class to store all attributes of a ptypy parameter

Returns **out** – Integrated array of same shape as *dpc_row* and *dpc_col*.

Return type ndarray

`ptypy.utils.scripts.`**`radial_distribution`**(*A*, *radii=None*)

Returns radial mass distribution up to radii in *radii*

Parameters

- **A** (*ndarray*) – input array
- **radii** (*list,tuple*) – Sequence of radii to calculate enclosed mass. If *None*, the sequence defaults to `range(1,np.min(A.shape)/2)`

Returns **radii, masses** – Sequence of used *radii* and corresponding intergrated mass *masses*. Sequences have the same length

Return type list

`ptypy.utils.scripts.`**`stxm_analysis`**(*storage*, *probe=None*)

Performs a stxm analysis on a storage using the pods. This function is MPI compatible.

Parameters

- **storage** (*ptypy.core.Storage instance*) – A *Storage* instance to be analysed
- **probe** (*None, scalar or array*) –
  - If None, picks a probe from the first view's pod
  - If scalar, uses a Gaussian with probe as standard deviation
  - Else: attempts to use passed value directly as 2d-probe

Returns

- **trans** (*ndarray*) – Transmission of shape `storage.shape`.
- **dpc_row** (*ndarray*) – Differential phase contrast along row-coordinates, i.e. vertical direction (y-direction) of shape `storage.shape`.
- **dpc_col** (*ndarray*) – Differential phase contrast along column-coordinates, i.e. horizontal direction (x-direction) of shape `storage.shape`.

`ptypy.utils.scripts.`**`stxm_init`**(*storage*, *probe=None*)

Convnenience script that performs a STXM analyis for storage *storage* given a probe array *probe* and stores result back in storage

See also:

stxm_analysis()

`ptypy.utils.scripts.`**`load_from_ptyr`**(*filename*, *what='probe'*, *ID=None*, *layer=None*)

Convenience script to extract data from `*.ptyr`-file.

Parameters

- **filename** (*str*) – Full Path to a `*.ptyr` data file. No check on the file suffix is done. Any compatible hdf5 formatted file is allowed.
- **what** (*str*) – Type of container to retrieve. Only *'probe'* and *'obj'* makes sense. Default is *'probe'*
- **ID** (*str*) – ID of storage in chosen container. If `None` the first stored storage is choosen
- **layer** (*int, optional*) – If an interger, the data buffer of chosen storage gets sliced with *layer* for its first index

Returns **data** – If *layer* is provided, that layer `storaga,data[layer]` will be sliced from the 3d data buffer, else the whole buffer `storage.data` will be returned.

Return type ndarray

Module contents

Util sub-package

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

1.5.2 Module contents

Stores description list for validation and documentation.

check (*pars*, *walk*)
Check that input parameter pars is consistent with parameter description. If walk is True and pars is a Param object, checks are also conducted for all sub-parameters.

choices = None
If parameter is a list of choices, these are listed here.

default = None
Default value can be any type. None if unknown.

entry_point

is_evaluable

longdoc = None
Longer documentation, may contain *sphinx* inline markup.

lowlim = None
Lower limit of parameter, None if unknown

make_doc ()
Create documentation.

name = None
Name of parameter

parent = None
Parent parameter (`PDesc` type) if it has one.

set_default (*default=''*, *check=False*)
Sets default (str) and derives value (python type)

shortdoc = None
Short descriptive string of parameter

type = None
Type can be a comma-separated list of types

uplim = None
Upper limit of parameter, None if unknown

userlevel = None
User level, a higher level means a parameter that is less likely to vary or harder to understand.

ptypy.utils.verbose module

Verbose package, based on the standard logging library.

Use as: from verbose import logger logger.warn('This is a warning') logger.info('This is an information') ...

TODO: - Handlers for file and for interaction server - option for logging.disable to reduce call overhead

This file is part of the PTYPY package.

copyright Copyright 2014 by the PTYPY team, see AUTHORS.

license GPLv2, see LICENSE for details.

ptypy.utils.verbose.**set_level** (*level*)
Set verbosity level. Kept here for backward compatibility

ptypy.utils.verbose.**report** (*thing*, *depth=4*, *noheader=False*)
no protection for circular references

Acknowledgements

First of all, I would like to thank my academic supervisor, Franz Pfeiffer, for giving me the opportunity for a PhD thesis at his chair, for providing an excellent working environment and for guidance. Thank you for drawing me from the far north down to Munich, I really enjoyed the great atmosphere you created at the chair.

I would like to thank my ptychography supervisor, Pierre Thibault, for introducing and guiding me through my adventures in the high-dimensional space and Python. It was a really great experience working with and for a mind like you.

Thank you, Martin Dierolf, for the countless lessons in Linux, for always being an encouraging listener, colleague and friend. Thank you also for patiently hosting me on arrival and my boxes on departure.

Thanks Marco, for all the good times we had at conferences and for being a fun companion at work, on the running track and in the Schwasi. Also thank you for tracking down spelling mistakes in my thesis.

Beyond Pierre, Martin and Marco, I am very thankful to all the other *Ptychonauts* on board the small ptychography vessel: Irene Zanette, Richard Clare and the Master students Konstantin Böll, Benedikt Daurer and Klaus Wakoning. Thanks for all the fun beamtimes, the inspiring discussions and friendly atmosphere and for laughing at my jokes.

I would like to express my deep gratitude to all other members at the E17 chair. You were not only colleagues but also friends to me. Especially, I would like to thank my long-term office mates Kai, Michael and Andreas. I enjoyed every day in the office. Lorenz, Mathias, Sebastian A. & E., Simone, Dieter, Arne, Marian and Julia, I thank you for the coffee breaks. Thank you Marian and Julia, for company at the weekends. Thanks also to all the nice persons who shared the office temporarily with me, Alexander, Maite, Barbara and Klaus. Especially, I would like to thank Dieter R. especially for the fun dinner gatherings at his home.

Many thanks to the organisational backbone of the chair, Nelly, Klaus and Brunhilde who are always in a good spirits. Thanks Klaus for the runs we had.

I would like to thank Peter Cloetens for all the outstanding support during experiments at the ID22NI beamline.

I am also very thankful to the cSAXS staff, especially Manuel Guitar-Sicairos, Ana Diaz and Andreas Menzel. Thank you for all the inspiring discussions and the good times at your beamline.

Thanks to my furry, little, nut-addicted visitor who brightened the early morning hours with his presence.

I would like to acknowledge the TUM Graduate School for support and the DFG Cluster of Excellence Munich-Centre for Advanced Photonics (MAP) for funding.

I extend my gratitude to the examination committee.

Last but not least, I would like to thank my family and Juliane for unlimited support and for reminding me that work is not everything.