JOURNAL OF APPLIED CRYSTALLOGRAPHY
ISSN 1600-5767

Gwaihir: *Jupyter Notebook* graphical user interface for Bragg coherent diffraction imaging

David Simonne,[a,b]* Jérôme Carnis,[c] Clément Atlan,[b,d] Corentin Chatelier,[b,d] Vincent Favre-Nicolin,[d] Maxime Dupraz,[b,d] Steven J. Leake,[d] Edoardo Zatterin,[a] Andrea Resta,[a] Alessandro Coati[a] and Marie-Ingrid Richard[b,d]

[a]Synchrotron SOLEIL, L'Orme des Merisiers, Saint-Aubin, 91192 Gif-sur-Yvette, France, [b]Univ. Grenoble Alpes, CEA Grenoble, 17 rue des Martyrs, 38000 Grenoble, France, [c]Center for Free-Electron Laser Science CFEL, Deutsches Elektronen-Synchrotron DESY, Notkestraße 85, 22607 Hamburg, Germany, and [d]European Synchrotron Research Facility, 71 Avenue des Martyrs, 38000 Grenoble, France. *Correspondence e-mail: david.simonne@synchrotron-soleil.fr

Received 17 February 2022
Accepted 1 June 2022

Edited by S. Boutet, SLAC National Accelerator Laboratory, Menlo Park, USA

Keywords: X-ray diffraction; coherence; phase retrieval; *Jupyter Notebook*; graphical user interfaces.

Bragg coherent X-ray diffraction is a nondestructive method for probing material structure in three dimensions at the nanoscale, with unprecedented resolution in displacement and strain fields. This work presents *Gwaihir*, a user-friendly and open-source tool to process and analyze Bragg coherent X-ray diffraction data. It integrates the functionalities of the existing packages *bcdi* and *PyNX* in the same toolbox, creating a natural workflow and promoting data reproducibility. Its graphical interface, based on *Jupyter Notebook* widgets, combines an interactive approach for data analysis with a powerful environment designed to link large-scale facilities and scientists.

1. Introduction

Bragg coherent diffraction imaging (BCDI) (Robinson & Harder, 2009) is a powerful technique for the nondestructive characterization of material structure in three dimensions with unparalleled spatial and strain resolution of a few nanometres (Labat *et al.*, 2015; Cherukara *et al.*, 2018*a*) and 10^{-4}, respectively (Newton *et al.*, 2010; Lauraux *et al.*, 2020).

The BCDI method is reliant on the coherence of the light available and thus only began to be exploited at third-generation synchrotron sources (Miao *et al.*, 1999, 2000; Robinson *et al.*, 2001). Since then it has developed into a characterization tool for *in situ*/*operando* studies of materials structure (Ulvestad *et al.*, 2016; Kim *et al.*, 2019; Carnis *et al.*, 2021*b*) and will further benefit from source and beamline improvements as many synchrotrons have recently completed or are in the process of an upgrade from third- to fourth-generation synchrotrons.

BCDI relies on iterative algorithms to solve the phase lost during the measurement (Robinson & Harder, 2009). A 3D intensity distribution in the vicinity of a Bragg peak (stack of diffraction patterns forming a 3D reciprocal space map with the proper sampling) is collected from a sample illuminated with coherent light (Robinson *et al.*, 2005) and serves as input for phase retrieval. There are three main steps to complete to image the strain. The 3D map must first be pre-processed (removal of parasitic scattering intensities *etc.*; Öztürk *et al.*, 2017). It must then be inverted via phase retrieval (Miao & Sayre, 2000), the phase containing important information lost during the measurement. Finally, it is possible to extract meaningful results from the 3D complex images after post-processing (removal of phase offset, interpolation in a common orthonormal frame *etc.*).

Several software packages were developed to solve these steps but none offer a comprehensive pipeline from start to finish. For example, *PyNX* (Favre-Nicolin *et al.*, 2020*a*) focuses on the phase retrieval step, *bcdi* (Carnis *et al.*, 2021*a*) focuses on data pre-processing and post-processing, and *Cohere* (https://github.com/AdvancedPhotonSource/cohere) focuses on pre-processing and phase retrieval. Several graphical user interfaces (GUIs) also exist, such as *Cohere*, *Phasor* (https://github.com/DzhigaevD/phasor) and *Bonsu* (Newton *et al.*, 2012). Providing a workflow will reduce the time spent on data analysis for newcomers and improve result reproducibility by facilitating sharing while keeping track of analysis parameters and metadata.

Large-scale facilities and institutions seek ways to provide remote-access high-powered computing services to their users, which combine existing solutions in an interactive and user-friendly environment. *Jupyter* (https://jupyter.org/) is particularly advantageous and has been chosen by several institutions for such purposes, *e.g.* Google (*Colab*), the EGI federation and the European Synchrotron (*Simple Linux Utility for Resource Management*: *SLURM*).

Here we present a tool which brings together the functionality of the *PyNX* package for phase retrieval and the *bcdi* package for pre- and post-processing in a GUI built for the *Jupyter* framework (Kluyver *et al.*, 2016).

The data input/output follows NeXus definitions (Könnecke *et al.*, 2015) built in a CXI format hdf5 file (Maia, 2012). Both clarity and consistency in data formatting encourage reproducibility of the science (see https://www.panosc.eu/) and guarantee the workflow.

Gwaihir provides an interface to the bleeding edge of data analysis in BCDI; it can be used locally or remotely and offers an interactive and user-friendly interface with complex functionality satisfying both beginners and experts.

2. Software structure

A part of the BCDI community relies on Python, an accessible language that has gradually become one of the most popular, versatile (Perez & Granger, 2007; Newville *et al.*, 2016) and widely taught (Scopatz & Huff, 2015; McKinney, 2017; Boulle & Kieffer, 2019) programming languages in science. *Gwaihir* followed this initiative by regrouping data visualization tools, workflow guidelines and a user-friendly interface around two Python packages (*PyNX* and *bcdi*) that, together, offer a complete data analysis suite (see Figs. 1 and 2).

Gwaihir works with Python 3.9 and is licensed under the GNU General Public License v3.0. The package is dependent on and built around *PyNX* and *bcdi*. The source code as well as the latest developments are available on GitHub (https://github.com/DSimonne/gwaihir), while each stable version will be released on the Python Package Index (PyPi), along with its documentation.

2.1. Presentation of involved Python packages

2.1.1. *bcdi*. Carnis *et al.* (2021*a*) tackled the pre-processing and post-processing of BCDI data (see Fig. 1).

Data pre-processing aims to improve the quality of the phase retrieval by optimizing the size and content of the 3D array used as input. Aside from the specific details of the experimental setup (diffractometer and setup geometry, detector type, file system), the majority of the pre-processing pipeline is actually beamline independent. On the basis of this observation, *bcdi* leverages inheritance and transforms the raw data to a common data format used for phase retrieval. This frees the user from having to learn or remember the technical details for each beamline and sets a common strategy across beamlines. New *Beamline* objects can be created for coherent beamlines which are unsupported as of yet.

The minimal processing consists of loading the raw data and stacking it together as an input for the phase retrieval.

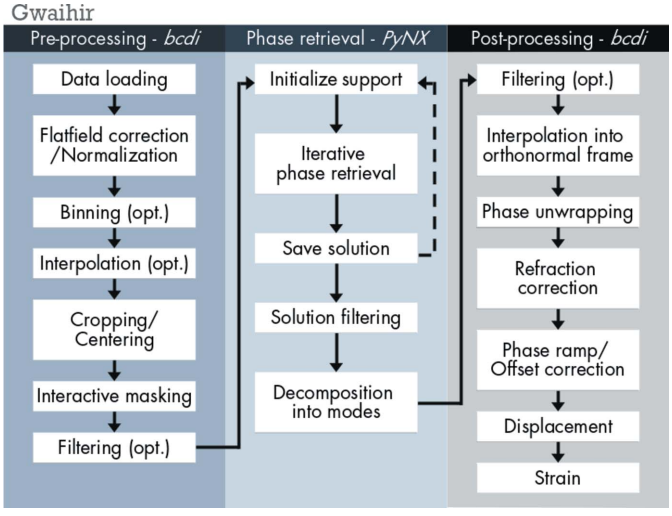

Figure 1
Flow chart of the main steps in the BCDI data analysis workflow. *Gwaihir* links the *bcdi* and *PyNX* packages via its GUI and command line scripts, resulting in a complete and easy to understand data analysis workflow (opt: optional).

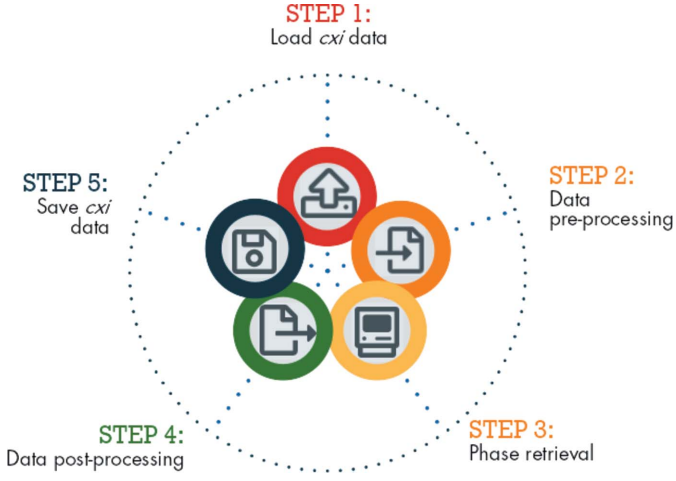

Figure 2
Workflow steps taken in *Gwaihir*; the circular workflow illustrates data reproducibility, a key concept, facilitated by the CXI architecture. The user may start the analysis directly from phase retrieval or post-processing, after reloading the CXI file.

Intermediate optional steps can be added via a YAML (YAML Ain't Markup Language) configuration file. More complete pre-processing involves loading the data (with flat-field correction and masking of damaged detector pixels and detector gaps), normalizing each frame by a monitor (sensor at the beamline measuring the incoming X-ray flux), binning pixels, centering the Bragg peak and cropping the data set to a meaningful size, and interpolating missing intensities in damaged pixels. A mask is automatically created, starting from a 2D array representing the detector. It can be modified interactively to mask the parasitic scattering intensities coming from neighboring crystals via the interactive user interface. Hot pixels (saturated) or pixels with an abnormal statistical behavior are automatically masked during data loading. The result is a 3D array of the same shape as the intensity. Interpolating the data stack from the detector frame to an orthogonal frame (also called a geometric transformation) may be useful, for example, to compare Bragg peaks from different reflections. This geometric transformation can be realized using either the transformation matrix (Pfeifer, 2005) or the existing *xrayutilities* package (Kriegner *et al.*, 2013) depending on which reference basis is needed.

Post-processing regroups methods applied to the complex output of the phase retrieval. The data can be interpolated in an orthogonal frame using the transformation matrix if still in the detector frame (geometric transformation not applied during pre-processing). After phase unwrapping, a refraction correction is optionally applied, and the phase ramp and phase offset are removed. At this point the displacement and the strain component are calculated from the phase. Note that at least three non-coplanar reflections are needed to derive the complete displacement field.

2.1.2. *PyNX*. *PyNX2011* (Favre-Nicolin *et al.*, 2011) is a toolkit with assorted Python modules and command-line scripts which can be used for the analysis of coherent X-ray imaging data, including phase retrieval (see Fig. 1). All calculations can be executed and distributed on multiple graphical processing units (GPUs) for accelerated computing using MPI.

Some facilities have started to make computational resources available to their users; those machines are an ideal environment for *PyNX* which is now available through *SLURM* at the ESRF and *GRADES* at the Optimized Light Source of Intermediate Energy of LURE (SOLEIL).

2.2. Graphical interface

Gwaihir links, in a single user-friendly and interactive GUI, the aforementioned packages while offering 2D/3D browser-based data visualization tools. The interface is built with the *ipywidgets* library (https://github.com/jupyter-widgets/ipywidgets), compatible with both *Jupyter Notebook* and *JupyterLab* web-based interactive computing platforms.

The GUI is divided into ten tabs (Fig. 3), separated into three groups: instrumental parameters, data processing and data analysis. The aim of this organization is to achieve a comprehensible but fluid workflow, while still separating each step (Section 3.1). A final tab contains information about the different methods and parameters used in the workflow, as well as a tutorial on the GUI.

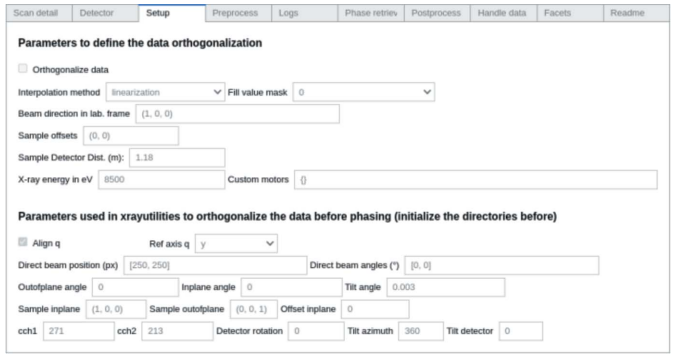

Figure 3
Screenshot of *Gwaihir* GUI displayed in *Jupyter Notebook*. *Jupyter Notebook* can be used on computing clusters via *JupyterHub* or on any machine for local use.

2.2.1. The *Jupyter Notebook* environment. The *Jupyter Notebook* environment (Perez & Granger, 2007; Kluyver *et al.*, 2016) was chosen for its versatile, user-friendly and browser-based interface.

To simplify the data analysis pipelines in fourth-generation synchrotrons, it is of critical importance to offer the possibility for external users to analyze the collected data remotely with access to computational environments

Jupyter Notebook has proven to be an effective tool for the analysis of synchrotron data, both in terms of the GUI (Martini *et al.*, 2020; Simonne *et al.*, 2020) and in terms of supporting scientific communities looking for high-performance frameworks (Yin *et al.*, 2017; Glick & Mache, 2018; Milligan, 2018; Stubbs *et al.*, 2020; Parkinson *et al.*, 2020). Moreover, computational environments and resources can be accessed from any computer via *JupyterHub*, a specific interface for computing clusters. *JupyterHub* offers the possibility to proceed to heavy computations without relying on specific hardware (*e.g.* GPUs) mandatory for accelerated phase retrieval with *PyNX*.

The *Notebook* can be used to take notes on the experiment or to create custom Python functions. They can be shared in `.ipynb` format or as `.pdf` documents. *Gwaihir* transforms each data set into a structured Python object that can be accessed and manipulated in the *Notebook* for *e.g.* personalized figures, parameter tests *etc.*

Moreover, researchers can create their own work spaces from shared resources, with direct access to tailored computational environments, without having to install multiple software products, keeping in mind that the use of GPUs for script optimization is far from accessible. Thus, system administrators can efficiently manage complex environments accessible to all users. Finally, remotely accessing the data avoids data storage issues, which can quickly become problematic with current CDI experiments.

2.2.2. Interactive data analysis/visualization. Interactive data analysis/visualization relies on the *ipywidgets* library.

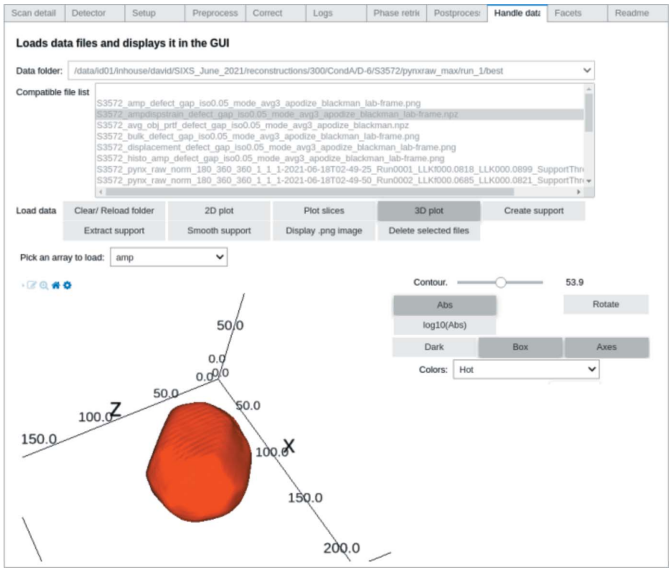

Figure 4
Widgets can be used to select folders and files and to tune the values of input parameters in functions. The selection of the data array, the color map and the contour of the resulting 3D object are performed through different widgets.

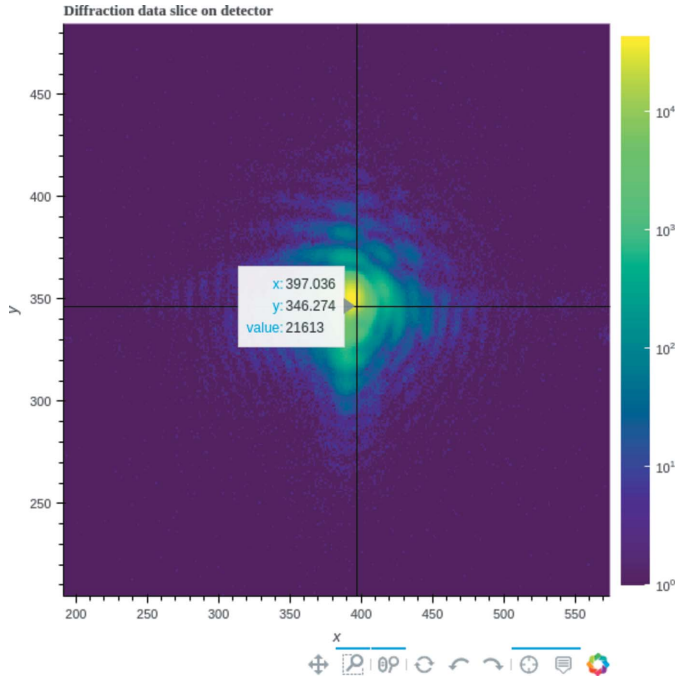

Figure 5
Detector images can be viewed interactively with *Bokeh*, to zoom in on the data, and visualize, for example, the intensity collected on each pixel.

Widgets are represented in the back-end by a single object linked to a single parameter. The front-end relies on JavaScript code; each time a widget is displayed, a new representation of that same object is created. The widget style, orientation and layout attributes can be edited to customize the final window; *e.g.* the layout attribute exposes a number of properties that impact how widgets are laid out, such as height and width.

During manipulation of the GUI, it is possible to quickly produce a final result owing to these intuitive and interactive widgets, whose values are passed as arguments to the data analysis functions (Fig. 4). For example, to designate the type of detector used, a value must be selected in a dropdown list that comprises the following options: Eiger2M, Maxipix, Eiger4M, Merlin, Timepix. These correspond to the different detectors currently supported in the *bcdi* package.

The final interface is created by setting the different widgets on a grid, with the output of different functions printed below.

Jupyter natively offers multiple options for interactive data plotting. Most of the figures displayed in the GUI are based on the *matplotlib* package (Hunter, 2007). For example, it is possible to select a list of 3D complex data arrays and visualize 2D slices in each dimension of their amplitude or phase, with different color maps (diverging, sequential, cyclic) and scale (linear, logarithmic).

Moreover, an interactive 3D visualization tool is provided which relies on the *ipyvolume* library (Breddeld, 2021), itself built on top of *ipywidgets*, and is specifically designed to quickly render large 3D data arrays. A concrete application of volume rendering is shown in Fig. 4, in which a specific contour of a 3D array is represented.

In the specific case of single volume rendering for the reconstructed object, the object surface is defined by a threshold of its maximum density (Fig. 4). Finally, the object surface can be color-mapped with the values of the displacement and strain retrieved during the data analysis (see Fig. 6).

To further interact with the figures (*e.g.* zoom, set the color bar range *etc.*), tools were implemented that rely on *Bokeh* (Bokeh Development Team, 2018), a Python library that transforms figures into interactive web pages (Fig. 5) that can also be displayed in *Jupyter Notebook*.

In summary, the GUI regroups data reduction, analysis and visualization tools in the *Jupyter Notebook* interface via interactive methods.

2.3. Command line scripts

The complete workflow for data processing can also be launched from the command line using Python scripts. In *Gwaihir*, the link between each package is based on BASH scripts. This approach is both fast and versatile, designed to quickly iterate on several data sets to test parameters values, but less intuitive.

A set of default parameters stored in a configuration file is used which can also be overwritten by providing keywords directly in the command line (*e.g.* the scan number). The configuration files are written in YAML (Fig. 6). Replication of the data analysis can be easily performed by sharing this configuration file.

3. Data reproducibility

Discussions about defining a set of rules that regulate research practice (Kretser *et al.*, 2019) and reduce the gray zone which includes scientific misconduct at all levels of academia

```yaml
scan: 1968
reconstruction_file: "result_1.cxi"
phasing_binning: [3, 1, 1]
preprocessing_binning: [1, 1, 1]
data_frame: "detector"
ref_axis_q: "y"
save_frame: "lab_flat_sample"
simulation: False
invert_phase: True
flip_reconstruction: False
setup:
    beamline: "P10"
    rocking_angle: "inplane"
    sdd: 1.818
    energy: 11294
    beam_direction: [1, 0, 0]
    sample_offsets: [0, 0, 0, 0]
    tilt_angle: 0.003
    direct_beam: [1340, 876]
    dirbeam_detector_angles: [0, 0]
    outofplane_angle: 40.0
    inplane_angle: 5.0
saving:
    save_rawdata: False
    save_support: False
detector:
    detector: "Eiger4M"
    roi_detector: [626, 1126, 1125, 1625]
phase_processing:
    phase_ramp_removal: "gradient"
    threshold_gradient: 1.0
    phase_offset: 0.0
    offset_method: "mean"
    isosurface_strain: 0.3
    strain_method: "default"
data_visualization:
    strain_range: 0.002
    phase_range: 3.142
    grey_background: True
    tick_spacing: 100
    tick_direction: "inout"
    tick_length: 3
    tick_width: 1
apodization:
    half_width_avg_phase: 1
    apodize: True
    apodization_window: "blackman"
    apodization_mu: [0.0, 0.0, 0.0]
    apodization_sigma: [0.3, 0.3, 0.3]
    apodization_alpha: [1.0, 1.0, 1.0]
```

Figure 6
Configuration file in YAML, a human-readable data-serialization language with a minimalist syntax. Each parameter used for the analysis methods is stored. A configuration file is generated for the pre-processing and post-processing scripts, as well as for the phase retrieval.

(Kornfeld & Titus, 2016) are growing, raising awareness on data reproducibility in the scientific community.

Staggering numbers (Baker, 2016) show that about 65% of scientists in the field of physics and engineering struggle to reproduce others' results, and more than 50% fail to reproduce their own. These numbers can sometimes be linked to very precise environments and techniques, with experimental conditions and processes difficult to reproduce in different laboratories, and also to knowledge transfer from academia to industry (Sarwitz, 2015). However, according to the study by Baker (2016), code availability, insufficient peer reviewing and limited access to raw data contribute to non-reproducible research.

Therefore, code availability, access to raw data combined with metadata, and reproducible workflows are goals of the utmost importance for experimental science (Munafò *et al.*, 2017). Reproducible data will result in a global improvement of confidence in new techniques, such as BCDI, which could subsequently result in growth of interest and community.

Raw data can be accessed through the CXI database (https://cxidb.org/; Maia, 2012), which aims to create a single data-storing architecture/format for coherent X-ray imaging experiments.

Data reproducibility is improved by encapsulating all the parameters and results in a final file. Therefore, any user should be able to reproduce the results from scratch. The version of each package (*Gwaihir*, *bcdi* and *PyNX*) is saved in the final file. It will be possible to use virtual environments to reproduce the same environment as that used during data analysis.

From this perspective, to improve the reproducibility of results, *Gwaihir* proposes a data analysis workflow for BCDI experiments.

3.1. Workflow for Bragg coherent diffraction imaging

Gwaihir offers an interactive workflow separated into three main groups, data pre-processing, phase retrieval and data post-processing (Fig. 1). It is intended to be reproducible and

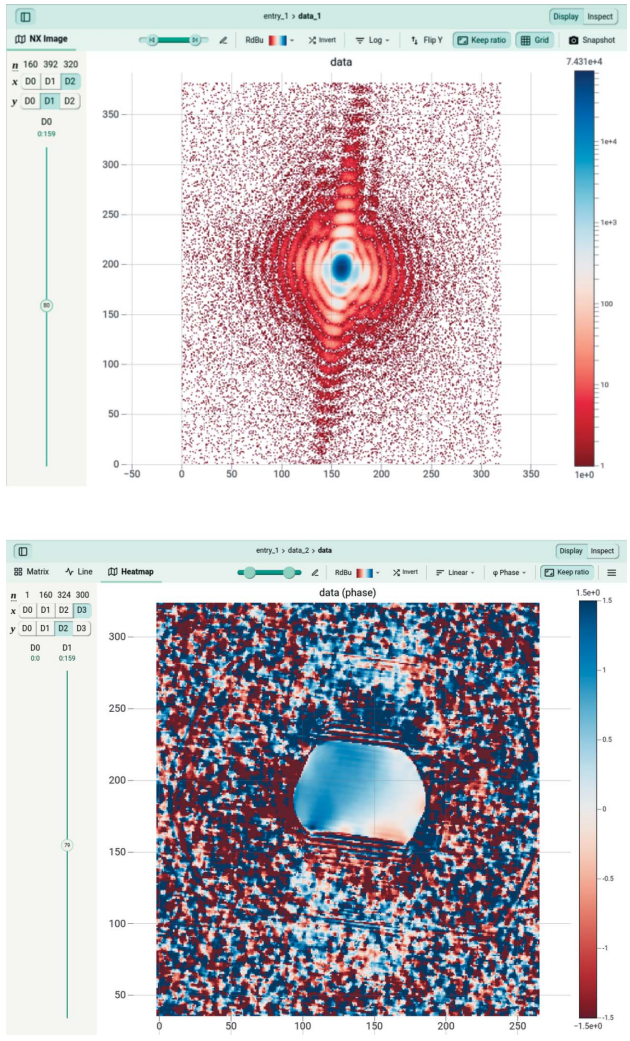

Figure 7
Here a 2D slice of a 3D coherent diffraction pattern is shown (top), with the phase of the electron density (radiants) of the reconstructed object in which the facets are clearly visible (bottom). The data array is displayed via *JupyterHub*, a browser-based data analysis platform available on the *SLURM* computer cluster for the ESRF and the *GRADES* computing cluster at SOLEIL.

simple to share, resulting in the phase and amplitude of the probed object. To illustrate this, the results of the following procedure on a data set collected at the P10 beamline at PETRA III are shown in Fig. 7 (CXI data set ID 195).

The parameters used during the analysis are displayed in the GUI in an order following a typical workflow to underline the evolution of the data processing. These parameters are ultimately saved as attributes of the Python data object, to keep track of the actions applied to the raw data to obtain the final result.

3.1.1. Pre-processing. First, the correct beamline and instrumental parameters must be selected (*e.g.* sample–detector distance, probing energy, detector pixel size *etc.*), usually constant throughout the experiment. For example, the orthogonalization tab regroups the parameters needed to properly set up the transformation matrix between the sample frame and, for example, the laboratory frame. By choosing to

transform the final object in a common, orthogonal frame, it becomes easier to compare the evolution of the object when probing different Bragg reflections (Lauraux *et al.*, 2021).

Once the raw data have been collected, different pre-processing parameters can be modified to optimize the collected 3D diffraction intensity, *e.g.* the size and center of the array to analyze (either fixed manually as the center of the Bragg peak, if known, or determined as the center of mass of the 3D array during the analysis). The array can also be cropped to reduce its size, removing the points furthest from the Bragg peak where the signal-to-noise ratio is too low (Fig. 7).

It is important to create a detector mask prior to the experiment to correct the raw data for hypothetical hot pixels and background. Moreover, it is also possible to correct the raw data frame by frame for spurious data which taint the diffraction pattern. For example, it is possible, while recording the 3D diffracted intensity of a given reflection, to have signal coming either from the substrate or from neighboring objects that will be summed to the probed object intensity.

Finally, it is possible to normalize the raw data by an intensity monitor, or compute the scattering vector $\mathbf{q}$ of the measurement, from the instrumental geometry and parameters.

3.1.2. Phase retrieval. The diffracted intensity $I(q)$ collected by the detector is proportional to the squared modulus of the structure factor $F(\mathbf{q})$ [equations (1) and (2)]:

$$F(\mathbf{q}_{hkl}) = \sum_{j}^{N} f_j(\mathbf{q}_{hkl}) \exp[i\mathbf{q}_{hkl} \cdot \mathbf{u}(\mathbf{r})] \quad (1)$$

and

$$I(\mathbf{q}_{hkl}) \propto |F(\mathbf{q}_{hkl})|^2, \quad (2)$$

where $f_j(\mathbf{q}_{hkl})$ is the atomic form factor of atom j, $\mathbf{q}_{hkl}$ is the scattering vector and $\Phi_{hkl} = \mathbf{q}_{hkl} \cdot \mathbf{u}(\mathbf{r})$ is the phase lost during the measurement; hkl correspond to the Miller indices of the crystallographic planes probed by the incoming beam.

The phase of the Bragg electronic density (Fig. 7) Φ_{hkl} is proportional to the scalar component of the displacement $\mathbf{u}(\mathbf{r})$ that is parallel to $\mathbf{q}_{hkl}$. A phase value of π is equivalent to a displacement from the equilibrium position of half the lattice spacing in the direction of $\mathbf{q}_{hkl}$.

To retrieve the phase from the diffracted intensity, *PyNX* uses iterative algorithms (see Section 2.1.2). New methods such as convolution neural networks are under study and have started to show some encouraging results, but are not yet sufficiently robust, particularly in the case of strained particles (Cherukara *et al.*, 2018*b*; Chan *et al.*, 2021; Wu *et al.*, 2021).

The parameters necessary for phase retrieval can be modified in the GUI (Fig. 8), such as the support threshold, the number of iterations for each algorithm, the object initialization procedure (square, sphere, auto-correlation) and so on. Each parameter is detailed in the GUI and also discussed in the literature (Fienup, 1982, 1978; Marchesini, 2007; Favre-Nicolin *et al.*, 2020*a*). Initial guesses are given for each parameter but must be refined by the user to optimize the results.

State-of-the-art GPUs available at large-scale facilities where data collection occurs enable almost real-time data reduction. This results in quick visualization of the amplitude and phase of the probed object, offering the possibility to synchrotron users to optimize their data acquisition during the experiment. Such live feedback is critical to the success of an experiment and provides broader opportunities in the type of experiments that can be carried out.

Recent updates of *PyNX*, including mathematical operators (Favre-Nicolin *et al.*, 2020*a*) which represent most of the reconstruction operations traditionally used in phase retrieval (Gerchberg & Saxton, 1972; Fienup, 1978; Marchesini, 2007), laid the foundation for quick and interactive phase retrieval in *Gwaihir*.

It is possible to refine the input parameters directly in the GUI by visualizing their impact on the reconstructed object. This allows the refinement of the phase retrieval input parameters before submitting a batch job, which will spawn a subprocess on the computing cluster for phase retrieval. With a single click, several dozens of solutions can be computed in a matter of minutes on computing clusters. On a personal computer, one can use operators in the *Notebook* or spawn a job locally, yielding the same results with different execution times.

With well-tuned parameters and high-quality data sets, phase retrieval converges towards the same solution, but with minor differences between each reconstructed object, related to the phase retrieval process.

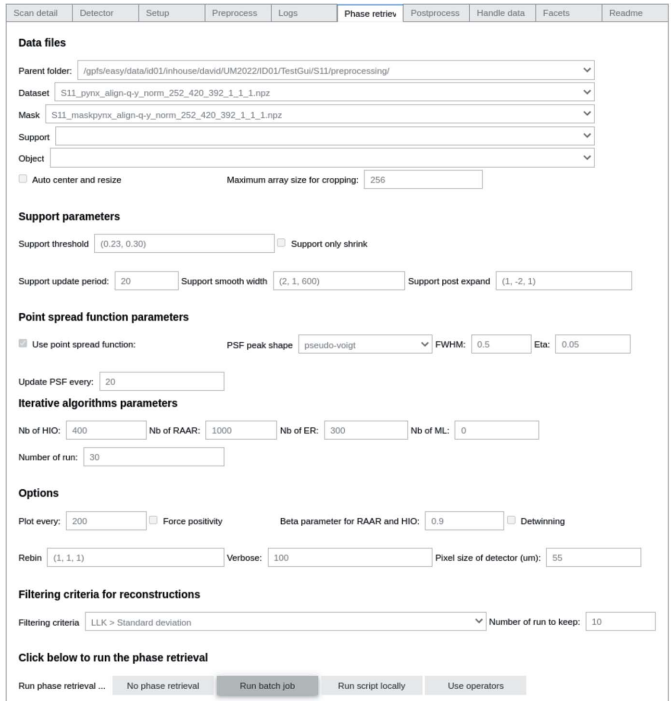

Figure 8
Phase retrieval tab in *Gwaihir*. Parameters are separated into groups (files, support, point-spread function, algorithms *etc.*) and detailed in the README tab. The object can be reconstructed through a batch job, submitted to the computing cluster in the back-end, or with operators that will plot the evolution of the reconstruction object in the *Notebook*.

Gwaihir provides a wide range of selection criteria to find the best solution. Each reconstructed object has a list of final attributes that can be used as criteria for selection, such as the free log-likelihood (Favre-Nicolin *et al.*, 2020*b*) or the standard deviation of the modulus of the reconstructed object. In the case of crystallographic defects, specific metrics (Chi, Sharp, Max Volume *etc.*) were derived that perform best depending on the type of object defect (Ulvestad *et al.*, 2017).

Following the selection criteria, it is possible to quickly identify the solutions of poor quality that must be excluded to create a set of best solutions. *PyNX* then offers a method that merges this set into a single solution by computing eigenvectors for the selected solutions (Favre-Nicolin *et al.*, 2020*b*). An alternative approach also available in *Gwaihir* is to take the average of the best solutions (see *e.g.* Ulvestad *et al.*, 2014).

3.1.3. Post-processing. Once the solution with the best figure of merit is selected, it is possible to use *bcdi* scripts to process the phase of the object. The phase origin can first have an impact when comparing the lattice displacement between different reconstructions. In the case of weak strain, it can be sufficient to consider the center of mass of the object as the origin of the phase. However, this can become quite complex in the case of defects or defaults in the object. Special methods are defined to target this issue in *bcdi*. For example, Guizar-Sicairos *et al.* (2011) and Hofmann *et al.* (2020) proposed a convenient method for the numerical calculation of phase gradients in the presence of phase jumps. More details and guidelines are given by Carnis *et al.* (2019, 2021*a*) as well as in the README tab of the GUI. Different methods can be tested and their results directly compared via the visualization tab.

3.1.4. Facet analysis. The surface in BCDI corresponds to the surface voxel layer defined by a threshold of its maximum density. Guidelines on how to select this threshold are given by Carnis *et al.* (2019), the objective being to correctly select the surface voxels of the object.

Once the threshold for the iso-surface is selected, it is possible to visualize a contour of the object directly in the GUI (Fig. 4) with specialized solutions such as *Paraview* (Ahrens *et al.*, 2005) (Fig. 9). Crystallographic facets can be identified on the surface of the reconstructed object when studying faceted objects with a highly coherent beam (Richard *et al.*, 2018), allowing in-depth studies of facet-dependent strain and displacement. Lattice strain and displacement are key factors in fields such as heterogeneous catalysis (Ulvestad *et al.*, 2016; Kim *et al.*, 2018; Fernández *et al.*, 2019; Passos *et al.*, 2020; Carnis *et al.*, 2021*b*) or electrochemistry (Vicente *et al.*, 2021).

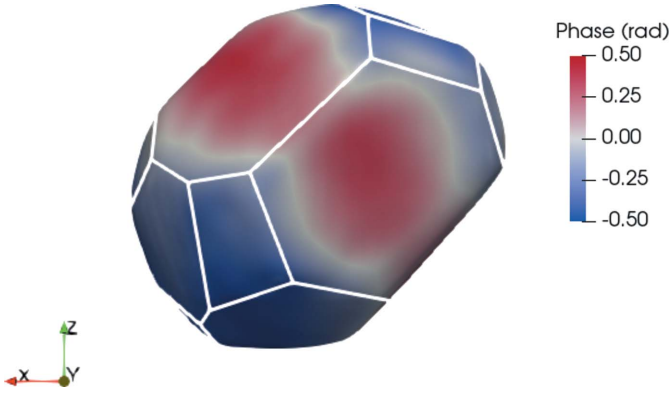

Figure 9
In this screenshot taken from *Paraview*, the particle facets are identified by the white contours. The coloring represents the phase of the reconstructed object, proportional to the lattice displacement. This particle has a diameter of about 800 nm.

Table 1
The output of facet analysis is a list of values for each facet.

The accessible features are facet size, average strain, average displacement, the facet center and the facet normal. The uncertainty on the average displacement and strain corresponds to the standard deviation of the displacement and strain distribution, respectively.

Facet ID	Facet normal	Relative facet size	Average displacement (Å)	Average strain (10^{-4})
1	$(1, 1, 1)$	0.106	0.080 ± 0.173	-0.28 ± 0.51
2	$(\bar{1}, \bar{1}, \bar{1})$	0.223	0.052 ± 0.26	-1.17 ± 0.64
3	$(1, \bar{1}, 0)$	0.106	0.080 ± 0.173	-0.28 ± 0.51
4	$(1, 0, 0)$	0.096	0.137 ± 0.192	0.27 ± 0.49
0	Edges and corner	NaN	-0.224 ± 0.259	0.20 ± 0.86

Retrieving the facets can be achieved by analyzing the probability distributions of the orientations of triangle normals on a mesh representation of the object (Grothausmann *et al.*, 2012). This method is used in the *Paraview* plugin *FacetAnalyser* (Grothausmann & Beare, 2015), and yields a list of features detailed in Table 1. A similar method is used in *bcdi*.

In this example, we use the *FacetAnalyser* plugin (Fig. 9). Note that the edges and corners, sites of particular interest for heterogeneous catalysis (Taylor & Armstrong, 1925), are also retrieved together as voxels not belonging to any facets. The result depends on the algorithm input parameters, such as the minimum relative facet size, the angular acceptance for the facet normals *etc.*

The `.vtk` output file of *FacetAnalyser* can be opened and manipulated in the GUI. Each facet is assigned a crystallographic orientation that allows *in fine* facet-dependent structural analysis. The facets can be visualized individually in three dimensions in the GUI, together with the edges and corners, to help understand the particle shape.

3.2. Resolution

Finally, the spatial resolution of the reconstructed object, defined according to multiple parameters in the literature, is critical for imaging techniques.

The voxel size in real space depends on the collected volume of reciprocal space: $2\pi/\delta q_x$, $2\pi/\delta q_y$, $2\pi/\delta q_z$, where q_x, q_y, q_z are the coordinates of the scattering vector. Three methods are commonly used to estimate the spatial resolution.

First, the phase retrieval transfer function (PRTF) (Chapman *et al.*, 2006) is the ratio of the calculated amplitude to the measured amplitude as a function of the resolution ring, which is a fraction of the sampling frequency of the diffraction pattern. The relative frequency at which the PRTF is equal to 0.5, or to $1/e$, can be used as an estimated spatial resolution of

the reconstruction. Note that Cherukara *et al.* (2018*a*) have recently demonstrated that the resolution of 3D real-space images obtained from Bragg X-ray coherent diffraction measurements is direction dependent.

Secondly, the Fourier shell correlation (van Heel & Schatz, 2005) measures the normalized cross-correlation coefficient between two 3D volumes and hence depends on two reconstructions that have to be the result of independent data sets.

Thirdly, spatial resolution can be quantified by differentiating line profiles of electron density amplitude across the object–air interface and fitting these with a Gaussian profile. The average 3D spatial resolution is taken as 2σ of the fitted Gaussian (Hofmann *et al.*, 2020).

Concerning the resolution of the retrieved atomic displacement, Labat *et al.* (2015) have demonstrated a displacement field accuracy of a few picometres with BCDI.

The resolution can be quantified in the *Gwaihir* GUI following different criteria and shared along with the processed data. Testing the influence of different parameters on the resolution and final results will help to derive optimum parameter values for data analysis.

3.3. Data storage

Since not all beamlines provide self-explaining NeXus data sets, it is the *bcdi* package that allows the support of the coherent imaging beamlines ID01 (ESRF), P10 (PETRA III), SixS and CRISTAL (SOLEIL), NanoMAX (MAX IV), and 34-ID-C (APS). Data pre-processing will generate two files stored as *NumPy* arrays (Van der Walt *et al.*, 2011), corresponding to the diffraction intensity and mask. These two files are then used for phase retrieval, for which the final object is saved in a CXI file, later used for data post-processing. In the case of simulation, the simulated diffraction intensity can be stored as a *NumPy* array to start the workflow from phase retrieval.

In *Gwaihir*, data sharing across teams and team members is facilitated by the creation of a single output file, respecting the CXI (Maia, 2012) and thus the NeXus (Könnecke *et al.*, 2015) architectures. As written by Könnecke *et al.* (2015), authors of data reduction and data analysis software can use NeXus to store processed data along with metadata and a processing log. This is illustrated in Fig. 10, in which the CXI data set tree is displayed. It regroups parameters from phase retrieval together with parameters from pre-processing and post-processing.

The raw data, along with all the parameters associated with the analysis, and the final results are stored in a single CXI file, allowing complete data analysis reproducibility (Fig. 10). This is possible at every step of the workflow. The aim is first to have a comprehensible architecture and secondly to be able to reproduce others' results from this file. On a small scale, results are easier to share between collaborators and are more understandable, whereas on a larger scale it will facilitate peer review.

Key parameters for data reproducibility are the transformation matrix used for the final interpolation, the voxel size of the resulting 3D array, the probed reciprocal space range after

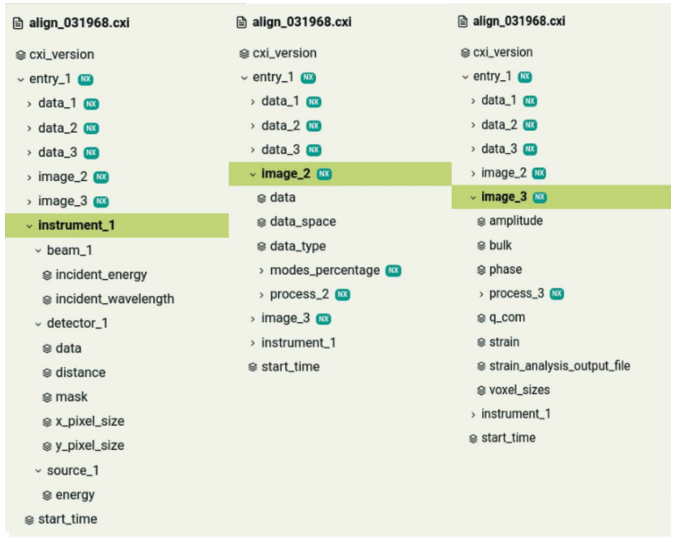

Figure 10
Each parameter used during the data analysis is stored in the same `.cxi` file, along with the results, in a nested architecture: 'instrument_1' regroups parameters associated with the instrumental setup; 'data_1' is the diffraction pattern collected; 'image_2' regroups the parameters associated with phase retrieval; 'data_2' is the reconstructed Bragg electronic density chosen for post-processing; 'image_3' regroups parameters linked to data processing, as well as the processed amplitude, phase and resulting strain; 'data_3' is a link to the processed phase of the object. The file structure is displayed via *JupyterHub*.

data pre-processing (δq_x, δq_y, δq_z), the iso-surface threshold *etc*. By sharing these parameters along with the diffraction intensity and complex Bragg electronic density, we can gain more transparency in the methods used to obtain the final results. Comments or metadata, such as the horizontal and vertical coherence lengths, or beam size, if determined prior to the experiment, can also be saved.

3.4. Examples

Example data can be downloaded from the CXI data bank (Maia, 2012). We have shown in Fig. 7 an example of an output file, using data collected at P10 (PETRA III; CXI entry 195). The GUI was tested on data collected at the ID01 beamline (ESRF; CXI entry 182) and also at the SixS beamline (SOLEIL; CXI entry 194), for which it is used as an 'online' analysis tool.

Demonstration videos detailing the GUI can be found in the GitHub repository at https://github.com/DSimonne/gwaihir. The `.cxi` file that resumes the data analysis workflow can be downloaded at https://www.dsimonne.eu/PhDAttachments/align_031968.cxi.

4. Conclusions

Gwaihir is a versatile tool for BCDI data analysis. It puts state-of-the-art BCDI analysis tools at researchers' fingertips in an intuitive interface and exploits the *Jupyter* framework to provide either local or remote operation seamlessly.

Gwaihir can interrogate the data in two or three dimensions to exploit the computing power of cluster resources. It offers

the possibility to share data in CXI format across users, a common architecture that includes all parameters and results of the methods used during pre-processing, post-processing and phase retrieval, and enables consistent and reproducible data analysis.

In a world where data are steadily made more available, *Gwaihir* is a tool that overcomes multiple issues by bridging remote access, cluster computing and a user-friendly interface, consequentially improving the link between synchrotrons and their users.

Acknowledgements

We thank Dr Frederic-Emmanuel Picca for his help regarding the creation of a *Debian 11* package for the installation of the *FacetAnalyser* plugin.

Funding information

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 Research and Innovation Programme (grant No. 818823 awarded to MIR).

References

Ahrens, J., Geveci, B. & Law, C. (2005). *ParaView: An End-User Tool for Large Data Visualization*, The Visualization Handbook, Vol. 717. Munich: Elsevier.

Baker, M. (2016). *Nature*, **533**, 452–454.

Bokeh Development Team (2018). *Bokeh*, https://bokeh.pydata.org/en/latest/.

Boulle, A. & Kieffer, J. (2019). *J. Appl. Cryst.* **52**, 882–897.

Breddeld, M. (2021). *ipyvolume*, version 0.6.0, https://github.com/maartenbreddels/ipyvolume.

Carnis, J., clatlan, Simonne, D., Leake, S., Dzhigaev, D., Kishore, K., Dupraz, M., Bot, D., Singaravelan, K. & Richard, M.-I. (2021*a*). *carnisj/bcdi*, version 0.2.1, https://doi.org/10.5281/zenodo.5741935.

Carnis, J., Gao, L., Labat, S., Kim, Y. Y., Hofmann, J. P., Leake, S. J., Schülli, T. U., Hensen, E. J., Thomas, O. & Richard, M. I. (2019). *Sci. Rep.* **9**, 1–13.

Carnis, J., Kirner, F., Lapkin, D., Sturm, S., Kim, Y. Y., Baburin, I. A., Khubbutdinov, R., Ignatenko, A., Iashina, E., Mistonov, A., Steegemans, T., Wieck, T., Gemming, T., Lubk, A., Lazarev, S., Sprung, M., Vartanyants, I. A. & Sturm, E. V. (2021*b*). *Nanoscale*, **13**, 10425–10435.

Chan, H., Nashed, Y. S. G., Kandel, S., Hruszkewycz, S. O., Sankaranarayanan, S. K. R. S., Harder, R. J. & Cherukara, M. J. (2021). *Appl. Phys. Rev.* **8**, 021407.

Chapman, H. N., Barty, A., Marchesini, S., Noy, A., Hau-Riege, S. P., Cui, C., Howells, M. R., Rosen, R., He, H., Spence, J. C. H., Weierstall, U., Beetz, T., Jacobsen, C. & Shapiro, D. (2006). *J. Opt. Soc. Am. A*, **23**, 1179–1200.

Cherukara, M. J., Cha, W. & Harder, R. J. (2018*a*). *Appl. Phys. Lett.* **113**, 203101.

Cherukara, M. J., Nashed, Y. S. G. & Harder, R. J. (2018*b*). *Sci. Rep.* **8**, 3577.

Favre-Nicolin, V., Coraux, J., Richard, M.-I. & Renevier, H. (2011). *J. Appl. Cryst.* **44**, 635–640.

Favre-Nicolin, V., Girard, G., Leake, S., Carnis, J., Chushkin, Y., Kieffer, J., Paleo, P. & Richard, M.-I. (2020*a*). *J. Appl. Cryst.* **53**, 1404–1413.

Favre-Nicolin, V., Leake, S. & Chushkin, Y. (2020*b*). *Sci. Rep.* **10**, 2264.

Fernández, S., Gao, L., Hofmann, J. P., Carnis, J., Labat, S., Chahine, G. A., van Hoof, A. J. F., Verhoeven, M. W. G. M. T., Schülli, T. U., Hensen, E. J. M., Thomas, O. & Richard, M.-I. (2019). *Nanoscale*, **11**, 331–338.

Fienup, J. (1982). *Appl. Opt.* **21**, 2758–2769.

Fienup, J. R. (1978). *Opt. Lett.* **3**, 27–29.

Gerchberg, R. & Saxton, O. (1972). *Optik*, **35**, 237–246.

Glick, B. & Mache, J. (2018). *J. Comput. Sci. Coll.* **34**, 180–188.

Grothausmann, R. & Beare, R. (2015). *MIDAS J.* https://doi.org/10.54294/bssu14.

Grothausmann, R., Fiechter, S., Beare, R., Lehmann, G., Kropf, H., Vinod Kumar, G. S., Manke, I. & Banhart, J. (2012). *Ultramicroscopy*, **122**, 65–75.

Guizar-Sicairos, M., Diaz, A., Holler, M., Lucas, M. S., Menzel, A., Wepf, R. A. & Bunk, O. (2011). *Opt. Express*, **19**, 21345–21357.

Heel, M. van & Schatz, M. (2005). *J. Struct. Biol.* **151**, 250–262.

Hofmann, F., Phillips, N. W., Das, S., Karamched, P., Hughes, G. M., Douglas, J. O., Cha, W. & Liu, W. (2020). *Phys. Rev. Mater.* **4**, 013801.

Hunter, J. D. (2007). *Comput. Sci. Eng.* **9**, 90–95.

Kim, D., Chung, M., Carnis, J., Kim, S., Yun, K., Kang, J., Cha, W., Cherukara, M. J., Maxey, E., Harder, R., Sasikumar, K., Sankaranarayanan, S. K. R. S., Zozulya, A., Sprung, M., Riu, D. & Kim, H. (2018). *Nat. Commun.* **9**, 3422.

Kim, D., Chung, M., Kim, S., Yun, K., Cha, W., Harder, R. & Kim, H. (2019). *Nano Lett.* **19**, 5044–5052.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S. & Willing, C. (2016). *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, edited by F. Loizides & B. Schmidt, pp. 87–90. Amsterdam: IOS Press.

Könnecke, M., Akeroyd, F. A., Bernstein, H. J., Brewster, A. S., Campbell, S. I., Clausen, B., Cottrell, S., Hoffmann, J. U., Jemian, P. R., Männicke, D., Osborn, R., Peterson, P. F., Richter, T., Suzuki, J., Watts, B., Wintersberger, E. & Wuttke, J. (2015). *J. Appl. Cryst.* **48**, 301–305.

Kornfeld, D. S. & Titus, S. L. (2016). *Nature*, **537**, 29–30.

Kretser, A., Murphy, D., Bertuzzi, S., Abraham, T., Allison, D. B., Boor, K. J., Dwyer, J., Grantham, A., Harris, L. J., Hollander, R., Jacobs-Young, C., Rovito, S., Vafiadis, D., Woteki, C., Wyndham, J. & Yada, R. (2019). *Sci. Eng. Ethics*, **25**, 327–355.

Kriegner, D., Wintersberger, E. & Stangl, J. (2013). *J. Appl. Cryst.* **46**, 1162–1170.

Labat, S., Richard, M.-I., Dupraz, M., Gailhanou, M., Beutier, G., Verdier, M., Mastropietro, F., Cornelius, T. W., Schülli, T. U., Eymery, J. & Thomas, O. (2015). *ACS Nano*, **9**, 9210–9216.

Lauraux, F., Cornelius, T. W., Labat, S., Richard, M.-I., Leake, S. J., Zhou, T., Kovalenko, O., Rabkin, E., Schülli, T. U. & Thomas, O. (2020). *J. Appl. Cryst.* **53**, 170–177.

Lauraux, F., Labat, S., Yehya, S., Richard, M.-I., Leake, S. J., Zhou, T., Micha, J.-S., Robach, O., Kovalenko, O., Rabkin, E., Schülli, T. U., Thomas, O. & Cornelius, T. W. (2021). *Crystals*, **11**, 312.

Maia, F. R. (2012). *Nat. Methods*, **9**, 854–855.

Marchesini, S. (2007). *Rev. Sci. Instrum.* **78**, 049901.

Martini, A., Guda, S. A., Guda, A. A., Smolentsev, G., Algasov, A., Usoltsev, O., Soldatov, M. A., Bugaev, A., Rusalev, Y., Lamberti, C. & Soldatov, A. V. (2020). *Comput. Phys. Commun.* **250**, 107064.

McKinney, W. (2017). *Python for Data Analysis: Data Wrangling with Pandas, NumPy and IPython*. Sebastopol: O Reilly.

Miao, J., Charalambous, P., Kirz, J. & Sayre, D. (1999). *Nature*, **400**, 342–344.

Miao, J., Kirz, J. & Sayre, D. (2000). *Acta Cryst.* D**56**, 1312–1315.

Miao, J. & Sayre, D. (2000). *Acta Cryst.* A**56**, 596–605.

Milligan, M. B. (2018). *Proceedings of the Practice and Experience on Advanced Research Computing (PEARC18)*, 22–26 July 2018, Pittsburg, PA, USA. NY: Association for Computing Machinery.

Munafò, M. R., Nosek, B. A., Bishop, D. V., Button, K. S., Chambers, C. D., Percie Du Sert, N., Simonsohn, U., Wagenmakers, E. J., Ware, J. J. & Ioannidis, J. P. (2017). *Nat. Hum. Behav.* **1**, 1–9.

Newton, M. C., Leake, S. J., Harder, R. & Robinson, I. K. (2010). *Nat. Mater.* **9**, 120–124.

Newton, M. C., Nishino, Y. & Robinson, I. K. (2012). *J. Appl. Cryst.* **45**, 840–843.

Newville, M., Stensitzki, T., Allen, D. B., Rawlik, M., Ingargiola, A. & Nelson, A. (2016). *ascl*:1606.014.

Öztürk, H., Huang, X., Yan, H., Robinson, I. K., Noyan, I. C. & Chu, Y. S. (2017). *New J. Phys.* **19**, 103001.

Parkinson, D. Y., Krishnan, H., Ushizima, D., Henderson, M. & Cholia, S. (2020). *Proceedings of the 2nd IEEE/ACM Annual Workshop on Extreme-Scale Experiment-in-the-Loop Computing (XLOOP)*, 12 November 2020, GA, USA, pp. 29–34. IEEE.

Passos, A. R., Rochet, A., Manente, L. M., Suzana, A. F., Harder, R., Cha, W. & Meneau, F. (2020). *Nat. Commun.* **11**, 1–8.

Perez, F. & Granger, B. E. (2007). *Comput. Sci. Eng.* **9**, 21–29.

Pfeifer, M. A. (2005). PhD thesis, University of Illinois at Urbana-Champaign, USA.

Richard, M.-I., Fernández, S., Eymery, J., Hofmann, J. P., Gao, L., Carnis, J., Labat, S., Favre-Nicolin, V., Hensen, E. J. M., Thomas, O., Schülli, T. U. & Leake, S. J. (2018). *Nanoscale*, **10**, 4833–4840.

Robinson, I. & Harder, R. (2009). *Nat. Mater.* **8**, 291–298.

Robinson, I. K., Da, Y., Spila, T. & Greene, J. E. (2005). *J. Phys. D Appl. Phys.* **38**, A7–A10.

Robinson, I. K., Vartanyants, I. A., Williams, G. J., Pfeifer, M. A. & Pitney, J. A. (2001). *Phys. Rev. Lett.* **87**, 195505.

Sarewitz, D. (2015). *Nature*, **525**, 159.

Scopatz, A. & Huff, K. D. (2015). *Effective Computation in Physics*. Newton: O'Reilly.

Simonne, D. H, Martini, A., Signorile, M., Piovano, A., Braglia, L., Torelli, P., Borfecchia, E. & Ricchiardi, G. (2020). *J. Synchrotron Rad.* **27**, 1741–1752.

Stubbs, J., Looney, J., Poindexter, M., Chalhoub, E., Zynda, G. J., Ferlanti, E. S., Vaughn, M., Fonner, J. M. & Dahan, M. (2020). *Practice and Experience in Advanced Research Computing (PEARC20)*, 26–30 July 2020, New York, NY, USA, pp. 91–98. Association for Computing Machinery.

Taylor, H. S. & Armstrong, E. F. (1925). *Proc. R. Soc. London Ser. A*, **108**, 105–111.

Ulvestad, A., Cho, H. M., Harder, R., Kim, J. W., Dietze, S. H., Fohtung, E., Meng, Y. S. & Shpyrko, O. G. (2014). *Appl. Phys. Lett.* **104**, 073108.

Ulvestad, A., Nashed, Y., Beutier, G., Verdier, M., Hruszkewycz, S. O. & Dupraz, M. (2017). *Sci. Rep.* **7**, 9920.

Ulvestad, A., Sasikumar, K., Kim, J. W., Harder, R., Maxey, E., Clark, J. N., Narayanan, B., Deshmukh, S. A., Ferrier, N., Mulvaney, P., Sankaranarayanan, S. K. R. S. & Shpyrko, O. G. (2016). *J. Phys. Chem. Lett.* **7**, 3008–3013.

Vicente, R. A., Neckel, I. T., Sankaranarayanan, S. K. R. S., Solla-Gullon, J. & Fernández, P. S. (2021). *ACS Nano*, **15**, 6129–6146.

Walt, S. van der, Colbert, S. C. & Varoquaux, G. (2011). *Comput. Sci. Eng.* **13**, 22–30.

Wu, L., Juhas, P., Yoo, S. & Robinson, I. (2021). *IUCrJ*, **8**, 12–21.

Yin, D., Liu, Y., Padmanabhan, A., Terstriep, J., Rush, J. & Wang, S. (2017). *Practice and Experience in Advanced Research Computing (PEARC17)*, 9–13 March, New York, NY, USA. Association for Computing Machinery.