Software Impacts 10 (2021) 100172

Contents lists available at ScienceDirect

Software Impacts

journal homepage: www.journals.elsevier.com/software-impacts

Original software publication

sxdm—A python framework for analysis of Scanning X-Ray Diffraction Microscopy data

William Judge [a,*], Michael Plews [a], Brian May [a], Martin V. Holt [b], Jordi Cabana [a]

[a] Department of Chemistry, University of Illinois, Chicago, IL 60607, United States of America
[b] Advanced Photon Source, Argonne National Laboratory, Argonne, IL, 60439, United States of America

ARTICLE INFO

Keywords:
Python
X-ray
SXDM
Framework
GUI

ABSTRACT

Significant interest has been placed on Scanning X-ray Diffraction Microscopy (SXDM) techniques for its ability to spatially resolve material/chemical strain at nanometer length scales. As instrumentation that employs this technique pushes the bleeding edge of research, the ability to process the influx of data produced is becoming difficult on traditional hardware. Traditionally, the analysis protocols were left for the individual to develop, leading to inefficiencies and long computational times for analysis. The python module *sxdm* was developed to reduce efficiencies by standardizing a data structure, displaying experimental analysis metrics in a graphical user interface, and providing resource-efficient analysis protocols.

Code metadata

Current code version	v1.0.0
Permanent link to code/repository used for this code version	https://github.com/SoftwareImpacts/SIMPAC-2021-152
Permanent link to Reproducible Capsule	
Legal Code License	GNU General Public License version 3
Code versioning system used	git
Software code languages, tools, and services used	python
Compilation requirements, operating environments & dependencies	installable through PyPi (pip)
If available Link to developer documentation/manual	https://sxdm.readthedocs.io/en/latest/
Support email for questions	wjudge2@uic.edu

1. Introduction

The ever-growing expansion of scientific technologies has created a world that allows single users to create more raw data than could be processed by that individual alone. This immense flux of data generation required innovative techniques to perform the analysis in a reasonable time frame, which can be carried out in one of three ways.

The first way is to use pre-built computational clusters for General Users. Argonne National Laboratory and Brookhaven National Laboratory have several resources (Theta, Cooley, Aurora, and the BlueGene Series) dedicated to user-submitted proposals. The second technique improvement is through the incremental upgrade of silicon chips used in central processing units (CPUs) or random access memory (RAM) modules by leading manufacturers. With decreasing prices and availability of once enterprise-grade equipment, computationally expensive data analytics can be achieved with comparatively little cost. The third avenue for improvement is to optimize open-source software used in data analysis for low-cost computational systems. This brings a deeper level of democratization of large data throughput to the average, computer-owning, individual.

Often, these three avenues for improving data analysis are intertwined. An advancement in one may push an advancement in another. This is not an intended competition between the three, rather than a cooperative venture to generationally improve the state of data

* Corresponding author.
E-mail addresses: wjudge2@uic.edu (W. Judge), plews2@uic.edu (M. Plews), bmmay@engsys.com (B. May), mvholt@anl.gov (M.V. Holt), jcabana@uic.edu (J. Cabana).

https://doi.org/10.1016/j.simpa.2021.100172
Received 29 October 2021; Received in revised form 8 November 2021; Accepted 10 November 2021

analysis. Therefore, it is ideal when newly developed modules for data processing and analysis are built to allow for the continued use of general computational resources even with this new influx of data, by focusing on the optimization of data handling methods into a more scalable solution.

Materials science has seen a rapid amount of scalable open-source software packages, which bring complex data analysis routines and simulations to the general scientific community. Some examples can be found through PyNx [1], AtomSK [2], and RMCProfile [3]. With their ease of use and low computational cost, these tools have paved the way for accessibility in materials science research. One less established technique used for such domain which could benefit from easy-to-use, scalable data analysis tools is scanning diffraction X-ray microscopy (SXDM). The code presented in this paper aims to contribute toward such outcome.

2. Scanning X-ray diffraction microscopy

The core concept of Scanning X-ray Diffraction Microscopy (SXDM) is to raster an X-ray beam of a given wavelength across a material, to spatially resolve Bragg diffraction signals [4,5]. Fundamentally, this technique does not require a high flux X-ray source or a 2-dimensional detector. However, to push SXDM to obtain nanometer resolution along with precise 2-dimensional diffraction patterns, a high-flux X-ray source is needed.

Nano-Scanning X-ray Diffraction Microscopy (nSXDM) can be carried out in several locations, most notably implemented at The Advanced Photon Source (beamline 26-ID-C) and National Synchrotron Light Source II (beamline 3-ID). These beamlines contains a unique instrument capable of implementing nSXDM at a spatial resolution of 30nm [4] or better. Detailed by May et al. [6], general SXDM experiments are initiated by interactions between an incoming X-ray and a staged sample that meets its respective Bragg condition. X-ray focus is achieved, samples are scanned (in X–Y) relative to the beam for a given sample theta, and the process is repeated for subsequent sample rotation angles. Since the detector dimensions are sensitive to either crystal domains (χ) or crystalline phase (2θ), one can create domain misalignment and chemical maps, respectively, for the sample of interest.

A standard workflow to analyze the 2θ vs χ mapping positions is as follows: obtain the raw data and understand data-structure, develop tools for background subtraction, align subsequent sample rotations, remove pixel anomalies, apply an analysis function to the pre-processed mapping positions, visualize the analysis outputs, all in a time efficient manner. Taken individually, most of these processing steps sound trivial. But, due to the lack of code repositories for these types of processes and the large data file sizes from SXDM experiments, it is difficult to achieve a resource-efficient software package. Previously bench-marked code was available to run on a single-threaded process, completed in 12+ hours, and occasionally crashed due to a RAM overflow. Even though this software package was developed for Sector 26-ID-C at the Advanced Photon Source, the goal of the proposed software package is to give researchers a standard platform for SXDM data analysis that is open-sourced, multi-threaded, resource-efficient, and which can be tailored for the specific needs of the scientist. Meeting these goals would allow researchers to focus more on the interpretation of results rather than data preparation for the efficient analysis of these types of datasets on resource-limited hardware.

The proposed sxdm module was scripted in Python, the object-oriented programming language for Sector 26-ID-C at the Advanced Photon Source. This language was chosen over the many alternatives that exist because it combines highly desirable features: it is open-source, has a comparably shallow learning curve and is supported by a large community infrastructure devoted to data science. Publicly available modules such as SciPy [7], Numpy [8], and PyQt5 [9] are critical to creating a fast and lightweight, self-contained module that encompasses a graphical user interface (GUI) that is easy to use . Even though we detail the data analysis in this paper, a more detailed explanation of pre-processing steps, such as data importing into h5, meta-data storing, scan dimension check, alignment, determining angle bounds, and background data creation, can be found on the sxdm documentation page.

3. Analysis

3.1. Hot pixels

During an SXDM experiment, the CCD detector occasionally will record pixels with abnormally large intensity pixel or contain defective pixels that continuously record the same value. The sxdm module allows the user to deal with these “hot pixels” in two different ways. First, the python module named SciPy [7] (Scientific Python) contains a median filter function which is then applied to the 1-D array summed column values or all row values of the 2D CCD detector image. Although fast, SciPy’s median filter comes at the cost of blurring the entire dataset. A selective median blur feature is also offered which is geared toward maintaining data integrity. If the current pixel in the search space is a set value away from the mean, the value is replaced with the mean value. This hot pixel removal method is slower than the SciPy filter, but retains most of the raw CCD detector values.

3.2. Centroid analysis

The primary tool for data analysis in the sxdm module targets the analysis of centroid positions of the diffraction pattern sampled by the CCD detector. The centroid of the signal in the x and y-axis of the detector is directly related to the presence of mechanical strain, microstructure or chemical state, depending on the material.

Fig. 1 shows the basics of the centroid analysis. The first step involves taking the summed diffraction data for a single pixel, subtracting corresponding background data, and removing hot pixels through a selective (Fig. 1B) or SciPy median filter (Fig. 1C). The program allows to selectively crop the data of interest (orange lines in Fig. 1B2/C2) to get a more accurate representation of the value of the signal centroid. This centroid position is then translated to a mapping position on an image array, Fig. 1 D. All data is paralleled through multi-threaded programming, allowing a full 1000 point scan to be completed in ≈20 s, all while using 8 gb of RAM.

3.3. Region of interest analysis

The second data analysis tool presented by the sxdm module was initially built to help users analyze the scan angle progress of a concurrent experiment. Similar data analysis protocols to those shown in Fig. 1 were implemented for the region of interest (ROI) calculations for each mapping position. The only difference is instead of calculating the centroid of the signal, the ROI mapping protocols take the sum of the signal. Once a user re-defines multiple ROIs inside the diffraction images (outside those stored during experimental run times), these ROI protocols are used to visually inspect distributions of diffraction signals coming from the current field of view (FOV). An example use case of this can be seen in Fig. 2. Without much effort, a user can identify enough points in their diffraction sample rocking curve to obtain reasonable analysis results. In this case, the user is able to identify conditions for the maximum intensity (i.e., at the Bragg condition) and does not need to take any more points to provide meaningful analytics to the system sampled.

Fig. 2 shows an example sample rocking curve taken for different scans of a FOV. One can see that it is critical to rock through various angles around an individual Bragg peak for a given crystal. Most experiments rely on guesswork to estimate when the experiment has

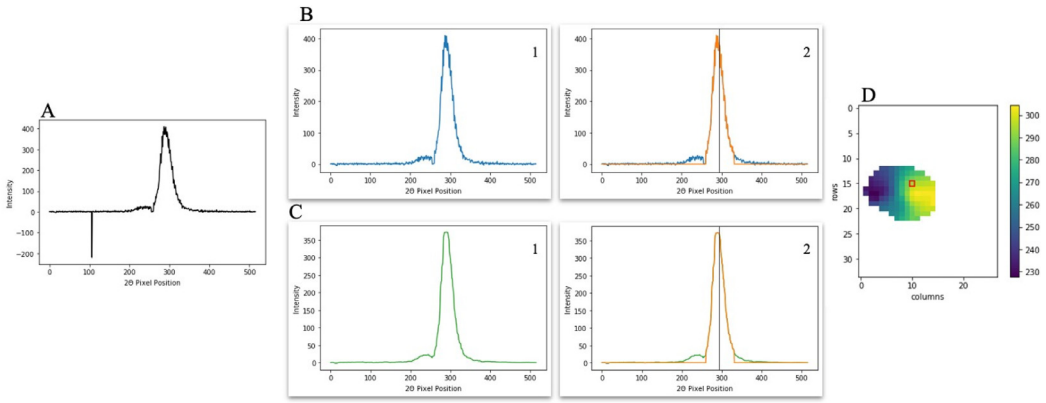

Fig. 1. Steps followed to process the data in the protocol for centroid analysis. First a raw spectral image (A) is subject to either a Selective Median Blur (B1) or SciPy Median Blur (C1). Then, a signal identifier is used to segment out the signal from the background (B2, C2 — Orange Lines). The centroid is established by the black Line, then placed into a 2-dimensional array for visualization (D). Plots in A, B, and C correspond to the red box shown in section D..

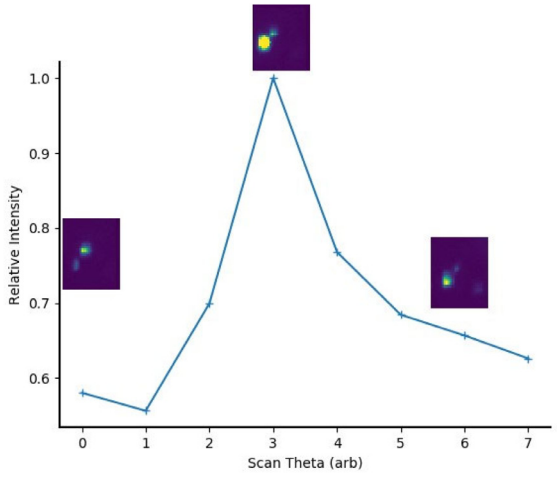

Fig. 2. A plot of the total diffraction intensity vs the scan number during a typical SXDM experiment. Scan theta incremental number is d. This plot shows the user has reached the diffraction intensity peak and can stop the experiment early without the need for more sample angle iterations.

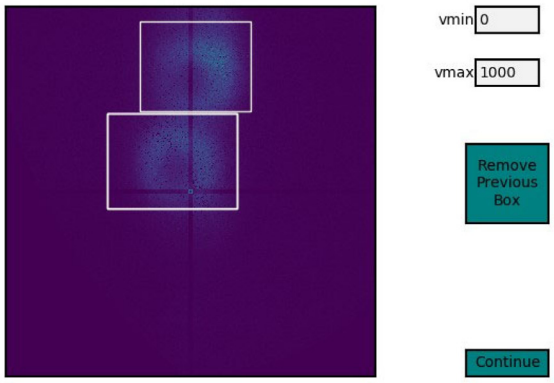

Fig. 3. A region of interest graphical user interface (GUI). This GUI allows the user to define their own diffraction bounding boxes, which will be used for generating separate region of interest maps for the selected diffraction regions.

reached the apex. This program considers the entire FOV or a section of the FOV and creates these rocking curves of intensity vs ordered angles to identify if the experiment can be concluded or which angles are missing data.

Once the experiment is complete, the ROI functionality can not only be used to segment multiple particles in the current FOV, but also help define strain/chemical phases by plotting differences in chemical states along a single particle. Users can define multiple ROI's for the given summed diffraction pattern FOV. These ROI's may correspond to different chemical states, particle strain regions, or multiple particles in the FOV. An example of the bounding box GUI can be seen in Fig. 3, which makes it intuitive to select these ROIs for the summed diffraction pattern. For each mapping position, all ROIs defined by the user are calculated in parallel, only taking 30 s to complete through multi-threading the sub-processes for the user. Built-in functions such as **here** can then be called to return a 4-dimensional map of the total intensity distributions for each selected ROI imported into the DataFrame: (user created ROI(s), real-space X, real-space Y, ROI intensity).

3.4. General analysis

Even though these analysis packages are helpful, users are able and encouraged to write their own functions to perform analysis tailored to their diffraction patterns. The General User Analysis suite provided allows users to conduct background subtractions, multi-threading, or data processing of their custom function as applied to individual pixels in the DataFrame.

The code found in Fig. 4 briefly details how 12 lines of code can achieve quick addition and subtraction analysis over all the data provided in the SXDMFrameset. This removes the need for users to develop resource efficient code to implement a background subtraction, scan alignment, and multiprocessing functions used in data analysis (2000 lines of code). Loading entire 100gb datasets into memory is unfeasible on ordinary laptops. The general user analysis provided by sxdm allows users to perform complex operations on 100's of GB of data, all from their laptops with as little as 8–16 GB of RAM. This brings data analysis to normal users without the need for costly high-performance compute units.

4. Viewing

This module comes with Viewer packages that allow users to visualize the centroid data and the single ROI data — lowering the knowledge barrier of high-level data analysis for scanning X-ray diffraction microscopy and making this technique available to more scientists. The graphical user interface (GUI) for the user is shown in Fig. 5. Built using the PyQT5 python package, the GUI is fast, responsive, and updates its appearance according to the user's native operating system. From this GUI, one can view the X-ray fluorescence map for a single element, individual diffraction patterns, summed diffraction patterns,

```
from sxdm import *
test_fs = SXDMFrameset(*args, **kwargs)
def user_func(each_scan_diffraction_post_bk_sub,
                    inputs):
    """Adds and Subtracts Values From Summed Diff. Pattern"""

    summed_dif = np.sum(each_scan_diffraction_post_bk_sub,
                        axis=0)
    adding, subtracting = inputs
    first = np.add(summed_diff, adding)
    second = np.subtract(first, subtracting)
    return second, first
# Set Up Background and File Locations
create_imagearray(test_fs)
# Perform Analysis
output = general_analysis_multi(test_fs,
                        rows=10,
                        columns=10,
                        analysis_function=user_func,
                        inputs=[1,3],
                        bkg_multiplier=0)
# Define the Analysis Outputs
user_acceptable_values = ['second', 'first']
# Return values
all_values = general_pooled_return(output, 'second',
                                   user_acceptable_values)
```

Fig. 4. A basic General User Function tutorial to apply a user generated mathematical operation on the collected diffraction pattern for a given mapping position.

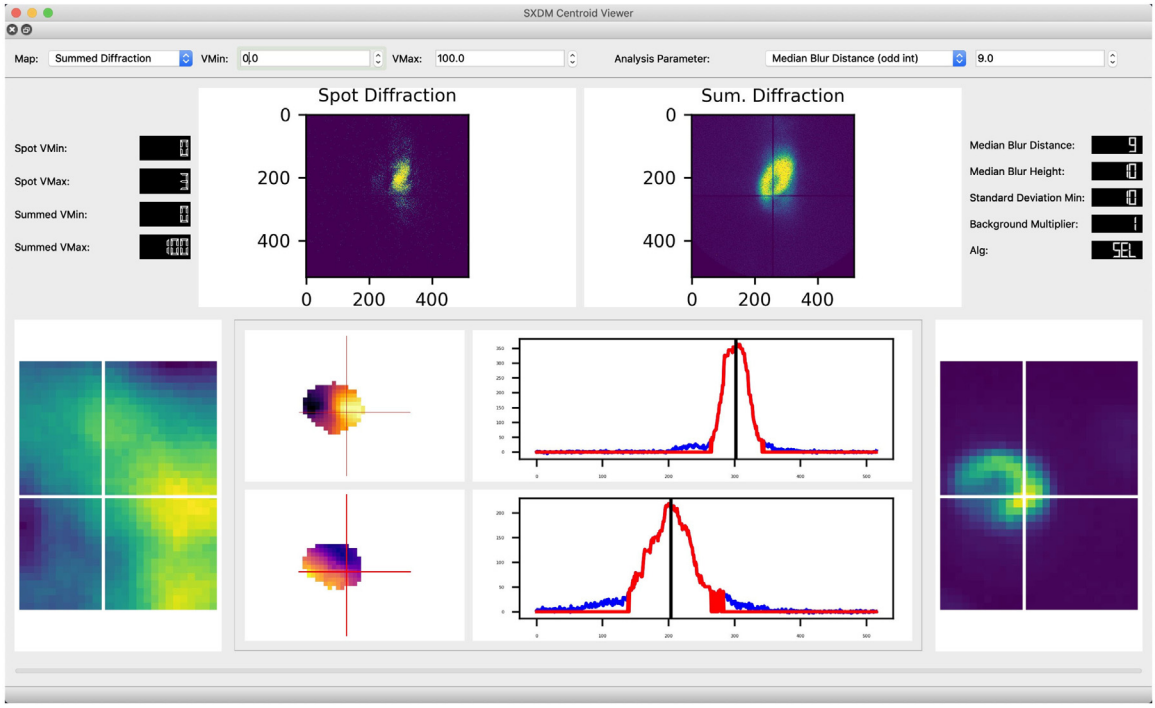

Fig. 5. The centroid viewer graphical user interface (GUI). This GUI allows the user to easily brows through all summed diffraction patterns (Spot Diffraction) for a given mapping position (Fluorescence signal). One is also given the ability to view the centroid calculations for both the 2θ and χ axes shown in the line plots in the figure.

region of interest map, and the centroid analysis for both diffraction axis domains. Not only are users able to interactively move the cursor around the field of view, but there are also able to change centroid analysis parameters and reprocess the data with these new parameters all from one location.

5. Conclusions

The sxdm python module provides valuable impact in the analysis of scanning X-ray diffraction microscopy data. It simplifies the experimental run times by providing fast analysis of experimental rocking-curves and allows users to focus on developing data analysis functions without worrying about data pre-processing or resource management of multi-threading. These advancements have decreased data analysis runtimes from hours to seconds, while resource requirements have dropped from 64GB+ to 8 GB. The sxdm module also provides a user-intuitive way to interact/analyze the centroids of the mapped diffraction data; therefore, decreasing the amount of time/knowledge/inconsistencies in data analysis through standardized analysis protocols. Current applications of this software have been used to determine crystal strain of Li-Ion battery materials at the nanoscale. Although the work is in progress, sxdm's first utilization has shown promising utility in the material science field. However, some limitations do exist. First, data importers may be challenging to write if users are unfamiliar with the incoming file structures. Second, duplication of incoming data is required. Lastly, this software package has only been developed for Ubuntu/MacOS in a Jupyter Notebook environment. Future developments could include

allowing for user-friendly creation of data importer functions, streaming data from current experimental files without data duplication, and testing the software on more computing environments such as Windows or headless servers.

CRediT authorship contribution statement

William Judge: Conceptualization, Methodology, Software, Validation, Writing – original draft, Data curation, Writing editing. **Michael Plews:** Methodology. **Brian May:** Data curation, Methodology, Validation. **Martin V. Holt:** Supervision. **Jordi Cabana:** Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This material is based upon work supported by the National Science Foundation, United States under Grant No. CBET2022723.

References

[1] V. Favre-Nicolin, J. Coraux, M.-I. Richard, H. Renevier, Fast computation of scattering maps of nanostructures using graphical processing units, J. Appl. Crystallogr. 44 (3) (2011) 635–640, http://dx.doi.org/10.1107/S0021889811009009.

[2] P. Hirel, Atomsk: a tool for manipulating and converting atomic data files, Comput. Phys. Commun. 197 (2015) 212–219, http://dx.doi.org/10.1016/j.cpc.2015.07.012.

[3] M.G. Tucker, D.A. Keen, M.T. Dove, A.L. Goodwin, Q. Hui, RMCProfile: reverse Monte Carlo for polycrystalline materials, J. Phys.: Condens. Matter 19 (33) (2007) 335218, http://dx.doi.org/10.1088/0953-8984/19/33/335218.

[4] R.P. Winarski, M.V. Holt, V. Rose, P. Fuesz, D. Carbaugh, C. Benson, D. Shu, D. Kline, G. Brian Stephenson, I. McNulty, J. Maser, A hard X-ray nanoprobe beamline for nanoscale microscopy, J. Synchrotron Radiat. 19 (6) (2012) 1056–1060, http://dx.doi.org/10.1107/S0909049512036783.

[5] P. Thibault, M. Dierolf, A. Menzel, O. Bunk, C. David, F. Pfeiffer, High-resolution scanning x-ray diffractionmicroscopy, Science 321 (5887) (2008) 379–382, http://dx.doi.org/10.1126/science.1158573, URL arXiv:http://science.sciencemag.org/content/321/5887/379.full.pdf, http://science.sciencemag.org/content/321/5887/379.

[6] B.M. May, Y.S. Yu, M.V. Holt, F.C. Strobridge, U. Boesenberg, C.P. Grey, J. Cabana, Nanoscale detection of intermediate solid solutions in equilibrated LixFePO4Microcrystals, Nano Lett. 17 (12) (2017) 7364–7371, http://dx.doi.org/10.1021/acs.nanolett.7b03086.

[7] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson, K.J. Millman, N. Mayorov, A.R.J. Nelson, E. Jones, R. Kern, E. Larson, C.J. Carey, I. Polat, Y. Feng, E.W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E.A. Quintero, C.R. Harris, A.M. Archibald, A. H.Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0 contributors, SciPy 1.0: Fundamental algorithms for scientific computing in python, Nature Methods 17 (2020) 261–272, http://dx.doi.org/10.1038/s41592-019-0686-2.

[8] C.R. Harris, K.J. Millman, S.J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N.J. Smith, R. Kern, M. Picus, S. Hoyer, M.H. van Kerkwijk, M. Brett, A. Haldane, J. Fernàndez del R` ıo, M. Wiebe, P. Peterson, P. G` erard Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, T.E. Oliphant, Array programming with NumPy, Nature 585 (2020) 357–362, http://dx.doi.org/10.1038/s41586-020-2649-2.

[9] PyQT, Pyqt reference guide, URL http://www.riverbankcomputing.com/static/Docs/PyQt4/html/index.html.