scimap Documentation

Release 0.2

Mark Wolf

Jul 20, 2021

Contents:

CHAPTER 1

Introduction & Installation

1.1 Motivation

Sorry, all out of motivation for today. Check back tomorrow.

1.2 Installation

The easiest way to run the development version is to use pip's **developer mode** (-e). First download the repository then install using pip. Be aware that downloading the repository may take a while as the test data can be very large.

```
$ git clone https://github.com/m3wolf/scimap.git
$ pip install -r scimap/requirements.txt
$ pip install -e scimap/
```

Now you should be able to import scimap in your python interpreter.

```
>>> import scimap
```

1.2.1 Running Tests

There is a set of unit tests and example data that accompany this project. To run the tests, install the project as described above then execute the test runner:

```
$ python scimap/tests/tests.py
```

and you should see something similar to:

```
..........x......x..x.xxx....xxxx.xxx.xxx......xx....x.........
----------------------------------------------------------------------
Ran 64 tests in 31.432s
```

(continues on next page)

(continued from previous page)

```
OK (expected failures=19)
```

1.2.2 Building Documentaion

The documentation for scimap is in the `docs/` directory (the html output is in `docs/_build/html/`). You will need the sphinx package installed in order to build it:

```
$ pip install sphinx
```

After making changes to the documentation source files (eg `docs/intro.rst`), re-build the documentation with:

```
$ cd scimap/docs/
$ make html
```

and view the result (eg `docs/_build/html/intro.html`). See the reStructuredText documentation for more information on the formatting of these .rst source files.

CHAPTER 2

Instrumentation and Beamline Support

2.1 Bruker GADDS Diffractometers

Bruker has a software package that is shipped with "Series II" diffractometers, called "GADDS". While this software is being replaced by an overhauled suite, it does have a scripting language that allows for control of the instrument via scimap.

Use of Bruker GADDS system with scimap has three main components

2.1.1 Data Acquisition Script

This is where the user will prepare a "slm file" which can be run in GADDS. It will instruct the instrument to trace out a desired path and save both 2D and 1D diffractograms, as well as a .jpg of the camera's view.

```
scimap.write_gadds_script(qrange=(1, 5), sample_name='example',
                          center=(1.32, -44.78), collimator=0.8)
```

This will create an `example-frames/example.slm` file that can be run using GADDS. It will also create some supporting files that will be used in the subsequent analysis.

2.1.2 Importing and Pre-Processing

Once the GADDS script has been run, the data can then be imported into the HDF file for further analysis.

```
scimap.import_gadds_map(sample_name="example",
                        directory="example_frames")
```

The value for `sample_name` should match that passed to the `write_gadds_script` function called above. After calling this function, the file `example.h5` contains the imported data.

2.1.3 Analysis and Visualization

Coming soon.

2.2 Bruker DaVinci Diffractometers

As of this writing, the modern Bruker diffractometer family does not support scripting and so mapping through scimap is not possible. Support for individual Bruker BRML files is available through `scimap.XRDScan` objects.

2.3 APS Beamline 34-ID-E

Scimap can import integrated data obtained from the 34-ID-E microdiffraction beamline at the Advanced Photon Source. The two-dimensional diffraction patterns must first be integrated to one-dimensional patterns using the `Fit2D` application. Once `.chi` files are prepared, they can be imported into scimap.

```
scimap.import_aps_34IDE_map(directory='example_frames/',
                            wavelength=0.516, shape=(10, 10),
                            step_size=0.10)
```

Where the wavelength is given in angstroms. Ideally, the .chi files should use scattering lengths, but any files in 2θ will be automatically converted to scattering length (q).

CHAPTER 3

Accessing the Underlying Data

Scimap tries to keep controller and presentation logic separate from the underlying data; most of the methods on the `XRDMap` class retrieve, display and/or store their data without the user's involvement. Given the complicated nature of scientific analysis, it can sometimes become necessary to retrieve individual diffraction patterns directly, or even to manipulate the data files directly; scimap provides a mechanism for both cases.

Scimap uses HDF5 files to store all the mapping data. This provides two benefits: 1) large datasets can still be analyzed even if they don't fit into main memory, and 2) the results of analysis can easily be shared or published as one file with accompanying metadata. This comes at the cost of increased time needed to write calculated data to disk, rather than manipulating it in main memory.

3.1 Retrieving Common Representations

The following methods of the `XRDMap` class can be used to retrieve a variety of packaged data.

- XRDMap.diffractogram: Get bulk-averaged diffraction data from all positions.
- XRDMap.get_diffractogram: Get the diffraction data for a single position.

3.2 Accessing the XRD Data Store and HDF File

It is possible to interact with the data as numerical arrays. The `XRDStore` class provides an interface for accessing the defined datasets. It can be retrieved throught the `XRDMap().store()` method, or instantiated directly.

Warning: The XRDStore class should be used as a **context manager** whenever possible. Failure to close the underlying HDF5 file, especially if using a writeable mode, is likely to lead to file corruption.

```
xmap = XRDMap(...)
with xmap.store() as store:
    Is = store.intensities
```

In the above example, `store.intensities` gives the intensity (photon counts) for each mapping position. The result will be an *m x n* array where *m* is the number of mapping positions and *n* is the number of angles/scattering vectors.

CHAPTER 4

Refinement

Analyzing XRD mapping data requires some sort of refinement on each mapping position. To accomplish this, the scimap provides the `scimap.xrd_map.refine_mapping_data()` method. Once refined, the results are stored in the HDF5 file and can be plotting using `scimap.xrd_map.plot_map()`. Different approaches are activated using the `backend` parameter. Many of the options, however, are unfinished or imperfect:

4.1 Fullprof

Warning: This backend is functional but fragile.

This backend requires that FullProf refinement be installed and available. The environmental variable `$FULLPROF` should point to the installation directory.

```
mymap = scimap.XRDMap(...)
mymap.refine_mapping_data(backend="fullprof")
```

FullProf refinement creates temporary files that are cleaned up upon successful refinement; if refinement fails, these files will be left behind for troubleshooting.

4.2 GSAS-II

Error: This backend is not functional. GSAS-II is under active development and if an API for Pawley refinement becomes available, this backend may be updated.

4.3 Pawley

Warning: This backend is incomplete. Us at your own risk.

This backend is an implementation of simple Pawley refinement in python.

4.4 Custom-Backends

If none of the available backends suit your needs, a custom backend may be provided. The backend should be a subclass of `scimap.base_refinement.BaseRefinement`. The `predict()` method should return the predicted intensities based on 2θ values, and a number of methods should be overridden that accept 2θ values and return refined parameters:

- `goodness_of_fit()`
- `background()`
- `cell_params()`
- `scale_factor()`
- `broadenings()`
- `phase_fractions()`

```
# Sub-class the base refinement
class CustomRefinement(scimap.BaseRefinement):
    def phase_fractions(self, two_theta, intensities):
        # Do some calculations here
        ...
    # Override the other methods here
    ...

# Now do the refinement
mymap = scimap.XRDMap(...)
mymap.refine_mapping_data(backend=CustomRefinement)
```

CHAPTER 5

Materials

Having a proper understanding of the material structure being studied is key to extracting usable information to map. The `scimap.XRDMap()` class accepts a `Phases=[]` argument, which is a list of `scimap.Phase` subclasses. The modules `scimap.lmo` and `scimap.nca` include pre-defined phases for $LiMn_2O_4$ and $LiNi_{0.8}Co_{0.15}Al_{0.05}O_2$ respectively. Unless you happen to be working with these materials, you will likely need to define some classes in order to perform a thorough analysis.

```
from scimap import Phase, TetragonalUnitCell, XRDMap

# Create a new class for our material
class Unobtainium(Phase):
    unit_cell = TetragonalUnitCell()
    # Define a list of hkl planes that define this strcture
    reflection_list = [
        Reflection('000', qrange=(2.75, 2.82)),
    ]

 # Now use our new class to analyze mapping data
 mymap = XRDMap(Phases=[Unobtainium])
```

Notice that the *Unobtainium* class is not instantiated before being passed to *XRDMap*. This is because each mapping position gets a new phase object that can be refined.

The *reflection_list* attribute describes the crystallographic reflection planes for this crystal system. Each entry in this list is a *Reflection* object. The first argument is a string with the hkl indices. Additionally, a *qrange* argument (2-tuple) should be given that gives the scattering vector (q) limits. Q can be calculated from 2θ values at a given wavelength λ:

$$q = \frac{4\pi}{\lambda} sin\left(\frac{2\theta}{2}\right)$$

As a shortcut, the function `scimap.twotheta_to_q` can also be used.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search