# SXDM Documentation

***Release 1.0.0***

**William Judge**

**Nov 23, 2021**

# CONTENTS:

Python library for analyzing **Scanning X-Ray Diffraction Microscopy** data.

> **Warning:** This documentation and the code are under active development. We make every effort to ensure the code is usable, but make no guarantees.

CHAPTER

ONE

# INTRODUCTION

SXDM is a toolkit for interacting with Scanning X-ray microscopy and X-ray Fluorescence data, most likely collected at the Advanced Photon Source - Beamline 26-ID-C. By collecting a set of frames at multiple incident angles/energies, spectral/diffraction maps are reconstructed to provide chemical/strain insight. Although SXDM was not designed for Fluorescence Mapping or XANES Mapping, built in functions allow Users to easily handle datasets to fit their needs.

This project has the following design goals:

- Provide a python toolkit for analysis of X-ray diffraction frames.
- Privide a python toolkit for importing and retrieving X-ray Fluorescence data.
- Store data in an open format for easy distribution.
- Provide framework 26-ID-C data handling.

GUI tools (eg. DUDE) exist for performing this type of analysis. While convenient and more versatile, final setup and testing has not been carried out. This module hopes to alleviate importing and data analysis issues for the 26-ID-C beamline. SXDM does provide an interactive GUI for visualizing the data, but this GUI does not alter the data or export results. This way, the analysis steps are captured either in an IPython notebook or conventional python script. These steps, together with the original data, should then be sufficient to reproduce the results exactly.

## 1.1 Installation

SXDM can be installed from the python package index (PyPI) using pip

```
$ pip install sxdm
```

## 1.2 Development

If you plan to contribute changes to *sxdm*, installing in developer mode may be more your style. This will also allow you to run tests and build documentation.

### 1.2.1 Installation

Assuming you are using conda, here are the steps. Any version of python >=3.5 should be ok.

```
$ conda create -n sxdm python=3.6
$ source activate sxdm
$ git clone clone git@github.com:WilliamJudge94/sxdm.git
$ pip3 install -r sxdm/requirements.txt
$ pip3 install ipykernel
$ ipython kernel install --user --name=sxdm
```

### 1.2.2 Jupyter

Once in Jupyter the User may have to initiate the import through

```
import sys
sys.path.append('/Users/usr/virtual_environment/lib/python3.6/site-packages/')

%matplotlib qt
import sys
sys.path.append('/path/to/sxdm/folder')
from sxdm import *
```

### 1.2.3 Tests

The easiest way to run unit-tests is with python tester from the command line:

```
$ pip install pytest
$ pip install pytest-cov

$ cd /dir/sxdm
$ pytest --cov=sxdm tests/
```

### 1.2.4 Documentation

The documentation is built using sphinx. To make HTML documents, use the following:

```
$ pip install sphinx
$ cd /dir/sxdm/docs
$ make html
```

## 1.3 Scanning X-Ray Diffraction Microscopy

Coming soon. . .

## 1.4 Example Workflow

A typical procedure for interacting with microscope frame-sets involves the following parts:

- Import the raw data (.mda and .tif)
- Apply corrections and align the images
- Calculate some metric and create maps of it
- Visualize the maps, statically or interactively.

Example for a single frameset across difference X-ray incident angles:

```
%load_ext autoreload
%autoreload 2
%matplotlib qt


# Developer Version
import sys
sys.path.append('/path/to/sxdm')
# Developer Version


from sxdm import *

# Set file name
hdf5_save_directory = '/dir/'
hdf5_save_filename = 'test'

# Importing .mda data
import_mda(mda_path, hdf5_save_directory, hdf5_save_filename)

# Importing .tif images - file='/dir/test.h5'
                          - was created by the import_mda() function
import_images(file, images_loc)

# Setting Detector Channels
disp_det_chan(file)
setup_det_chan(file, fluor, roi, detector_scan,
                filenumber, sample_theta, hybrid_x, hybrid_y, mis)

# Set up SXDMFrameset
scan_numbers = [1, 2, 3, 4, 5, ...]
dataset_name = 'Test_Name'
test_fs = SXDMFrameset(file, dataset_name,
                       scan_numbers = scan_numbers, median_blur_algorithm='numpy')

# Alignment
test_fs.alignment()

# Analysis
test_fs.centroid_analysis()

# Viewer
test_fs.centroid_viewer()
```

CHAPTER

TWO

# IMPORTING DATA INTO SXDM

The first step in any SXDM workflow will be to **import the raw data into a common format**. These importer functions are written as needed: if your preferred beamline is not here, submit an issue.

## 2.1 APS Beamline 26-ID-C

### 2.1.1 Experimental Data (.mda) Import

The raw data file `file.mda` given to the User at 26-ID-C saves all source data as a matlab binary file. SXDM preserves the original ("source") file and saves imported and processed data in a second ("destination") HDF file to be used in later analysis. The source `file.mda` file can be easily imported:

```
import_mda(mda_path='path/to/folder/holding/.mda_files',
              hdf5_save_directory='path/to/save/dir',
              hdf5_save_filename='file')
```

This function will iterate through all `scan.mda` files in the *mda_folder* and import all detector channel data into the User defined hdf5 destination/file.

```
# EXAMPLE

import_mda(mda_path='/home/usr/Desktop/mda_folder/',
             hdf5_save_directory='/home/usr/Desktop',
             hdf5_save_filename='test_file')
```

**Note:** Raw reader values are flipped and inverted to match 26-ID-C beamline MatLab Viewer output.

**Note:** This importer is what creates the main .h5 file.

**Note:** There is no way to force overwrite imported data. This will be added soon.

### 2.1.2 Experimental Data (.mda) XRF Data Import

Since XRF data adds another dimensionality to the import once must set maxdims=3. These values will be saved under the xrf/####/D## in the hdf5 file.

```
# EXAMPLE

import_mda(mda_path='/home/usr/Desktop/mda_folder/',
           hdf5_save_directory='/home/usr/Desktop',
           hdf5_save_filename='test_file',
           maxdims=3)
```

### 2.1.3 Experimental Data (.mda) Single Data Import

If one would like to only import a single file they may add the file location to the single_file argument.

```
# EXAMPLE

file_location = '/home/user/Desktop/mda/filename_0001.mda'

import_mda(mda_path='/home/usr/Desktop/mda_folder/',
           hdf5_save_directory='/home/usr/Desktop',
           hdf5_save_filename='test_file',
           maxdims=3,
           single_file=file_location)
```

### 2.1.4 Diffraction Image (.tif) Import

The raw diffraction images `image_#####.tif` given to the User at 26-ID-C will be imported based on the protocol below. SXDM preserves the original ("source") file and saves imported and processed data in a second ("destination") HDF file to be used in later analysis. All source `image_#####.tif` file can be easily imported:

```
import_images(
    file='path/to/save/dir/file.h5',
    images_loc='/path/to/master/images/directory',
    scans=False,
    fill_num=4,
    import_type='uint32',
    delimiter_function=<function delimiter_func at 0x7f0873f3fe18>,
    force_reimport=False,
    )
```

This function will iterate through all folders in the `images_loc` folders and import all `images_####.tif` image data into the User defined hdf5 destination/file.

```
# EXAMPLE
# /home/usr/Desktop/images_folder/scan_folder/image.tif

import_images(
    file='/home/usr/Desktop/test_file.h5',
    images_loc='/home/usr/Desktop/images_folder/',
    scans=[1, 2, 10, 18],
    )
```

**Note:** fill number is the number of digits in the image_####.tif name.

**Note:** scans=False will import all scans in the designated folder. Must be either False or an array.

**Note:** import_type gets passed into 'np.astype()' function

**Note:** This will **Not** reimport the .tif images. If the User would like to do this they can set `force_reimport=True`

## 2.2 Data Structure

The main structure is similar to what is shown below:

```
#Main_HDF5_File#

    #images/
        #0001_scan/
            #000001.tif
            .
            .
            .
            #number.tif

        #0002_scan/
            #000001.tif
            .
            .
            .
            #number.tif

        #0003_scan/
            #000001.tif
            .
            .
            .
            #number.tif

    #mda/
        #0001_scan/
            #D01_channel/
                #detector data
            .
            .
            .
            #D70_channel/
                #detector data

        #0002_scan/
            #D01_channel/
```

(continues on next page)

(continued from previous page)

```
                #detector data
            .
            .
            .
            #D70_channel/
                #detector data

        #0003_scan/
            #D01_channel/
                #detector data
            .
            .
            .
            #D70_channel/
                #detector data

    #detector_channels/
        #detector_scan/
        #filenumber/
        #fluor/
        #hybrid_x/
        #hybrid_y/
        #mis/
        #roi/
        #sample_theta/

    #zone_plate/
        #D_um/
        #d_rN_nm/
        #detector_pixel_size/

    #dataset_name1/
        #dxdy/
        #scan_numbers/
        #scan_theta/

    #dataset_name2/
        #dxdy/
        #scan_numbers/
        #scan_theta/
```

**Note:** Please see *Analyzing the Data/Retrieving Imported Data* for more details

CHAPTER

THREE

# ANALYZING THE DATA

## 3.1 Setting Up Detector Channels

After importing the data, the User will likely want to store detector channel values for the Experimental Run. This avoids having to remember them every time the User would like to go back and analyze a specific dataset. To begin, run the following

```
%load_ext autoreload
%autoreload 2
%matplotlib qt

from sxdm import *

file='/path/to/file.h5'

# Display a preset detector channel input that the User can copy
disp_det_chan(file)


# Example Output

detector_scan = {
    'Main_Scan': 2,
    }

filenumber = 2

fluor = {
    'Cu': 2,
    'Fe': 2,
    'Zn': 2,
    'O': 2,
    }

hybrid_x = 2

hybrid_y = 2

xrf = {
        'Fe':2,
        'Cu':2,
        'Ni':2,
        'Full':2,
        }
```

(continues on next page)

(continued from previous page)

```
mis = {
    '2Theta': 2,
    'Relative_r_To_Detector': 2,
    'Storage_Ring_Current': 2,
    }

roi = {
    'ROI_1': 2,
    'ROI_2': 2,
    'ROI_3': 2,
    }

sample_theta = 2
```

If detector channels have not been previously set default values will present themselves. Change all values according to your experimental setup.

---

**Note:** All dictionary entries are in the form of 'Detector Name': detector number. Example above says the Cu flourescence detector channel was 2. The Fe flourescence detector channel was also 2. Etc. All dictionary entries, reguardless of what they are for, should have different detector channel numbers.

---

`fluor` (dic) The User can place as many Fluorescence dictionary entries as they would like. Except there must be at least 1. Entries can be named however the User would like.

`roi` (dic) The User can place as many Region of Interest dictionary entries as they would like. Except there must be at least 1. Entries can be named however the User would like.

`detector_scan` (dic) Main_Scan must be the first and only dictionary entry. This corresponds to the scan where the User rocked the detector. This will be used to determine the x and y angle values.

`filenumber` (int) This must be a single integer value corresponding to the detector channel associated with the filenumbers of the images.

`sample_theta` (int) This must be a single integer value corresponding to the detector channel associated with the sample theta angle.

`hybrid_x` (int) This must be a single integer value corresponding to the detector channel associated with the hybrid_x location/motor position. this should correspond to the detector number of the motor you are scanning in the x direction

`hybrid_y` (int) This must be a single integer value corresponding to the detector channel associated with the hybrid_y location/motor position. this should correspond to the detector number of the motor you are scanning in the y direction

`xrf` (dic) The User can place as many x-ray fluorescence dictionary entries as they would like. Except there must be at least 1. Entries can be named however the User would like. None of these are in use in SXDM-1.0. They are there for User additions. THESE CORRESPOND TO THE DETECTOR CHANNELS UNDER THE 'xrf' HEADING THE THE HDF FILE!!!

`mis` (dic) The User can place as many miscellaneous (mis) dictionary entries as they would like. Except there must be at least 1. Entries can be named however the User would like. None of these are in use in SXDM-1.0. They are there for User additions.

Once these values are set the User can run

```
# Change Values From Default Output, Run Cell, And Input Values Into Function Below
setup_det_chan(file,
               fluor,
```

(continues on next page)

(continued from previous page)

```
                roi,
                detector_scan,
                filenumber,
                sample_theta,
                hybrid_x,
                hybrid_y,
                mis,
                xrf,)

# You can reset the detector_channels through `del_det_chan(file)` function
```

## 3.2 Setting Up Frameset

After importing the data, and setting the detector channels you will likely need to process and analyze the frame set. This is done through the `sxdm.SXDMFrameset` class. Most **processing and analysis steps are provided as methods on this class**, so the first step is to create a frameset object.

```
%load_ext autoreload
%autoreload 2
%matplotlib qt

from sxdm import *

# Use the same HDF file and group name as when importing
test_fs = SXDMFrameset(file'/path/to/file.h5',
            dataset_name='user_dataset_name',
            scan_numbers=[1, 2, 3, 4, ...],
            fill_num=4,
            restart_zoneplate=False,
            median_blur_algorithm='selective',
            )
```

`file` (str) the path to the hdf5 file you would like to import data from

`dataset_name` (str) the group name of the scans you are importing

`scan_numbers` (nd.array or False) an array of ints of the scan numbers you would like to group together. If False - this will import the stored/previously completed scan numbers data

`fill_num` (int) the amount of digits in the image file number

`restart_zoneplate` (bool) if you would like to restart the zoneplate data set this to True

`median_blur_algorithm` (str) this initializes which type of median blur will be performed on the datasets during analysis. acceptable values consist of 'scipy' and 'selective'. ''scipy performs a median blur on the entire dataset while 'selective' only applies a median blur if the binned 1D data is within a certain User threshold.

### 3.2.1 Median Blur Type Selection

In the creation of the SXDMFrameset there is an option to set a `median_blur_algorithm`. There are two option in the current version of SXDM. `scipy` and `selective`.

**sxdm.mis.median_blur_scipy()**

This median blur algorithm calls the `scipy.signal.me_blur`. This will apply a median blur to the entire 1 dimensional datasets produced by the 2 dimensional images.

**sxdm.mis.median_blur_selective()**

This median blur alogrithm bins off line scan data, determines the mean, if there is a value above a User value + mean it will be replaced with the mean value for the chunk. This preserves most of the raw intensity data at the cost of speed.

## 3.3 Zone Plate Values

The program will ask for the following values upon the first run:

Diameter Of The Zone Plate Is _____ microns Outermost Zone Plate d Spacing Is _____ nanometers The Size Of Your Detector Pixels Is _____ microns The Detector Theta Value Is _____ Degrees and the Kev is _____ Kev

These values will be stored into the file as attributes for the dataset_name.

## 3.4 Scan Dimensions Check

Starting the SXDMFrameset will automatically determine the pixel X resolution for all the imported scans as well as all the Y resolutions for all the scans and checks to make sure every scan has identical X resolutions and every scan has identical Y resolutions. Then it checks to see if the median(x) and median(y) resoltuions are equivalent.

If the program throws an error during the resolution check walk through the following:

- Make sure you have set the `hybrid_x` and `hybrid_y` values correctly in the `setup_det_chan()` function.
- Pull up all the scan resolutions with `test_fs.all_res_x`, and `test_fs.all_res_y`. These will be in the same order as *test_fs.scan_numbers*. Remove the scan that is throwing the error when setting up `test_fs = SXDMFrameset()`. Future versions will resample the scans to create identical resolutions in all X, all Y, and in X v. Y.
- If there is still an error the scan dimensions are not the same across all scans. Run `show_hybrid_dimensions(test_fs)` to see all the scan dimensions

## 3.5 Alignment

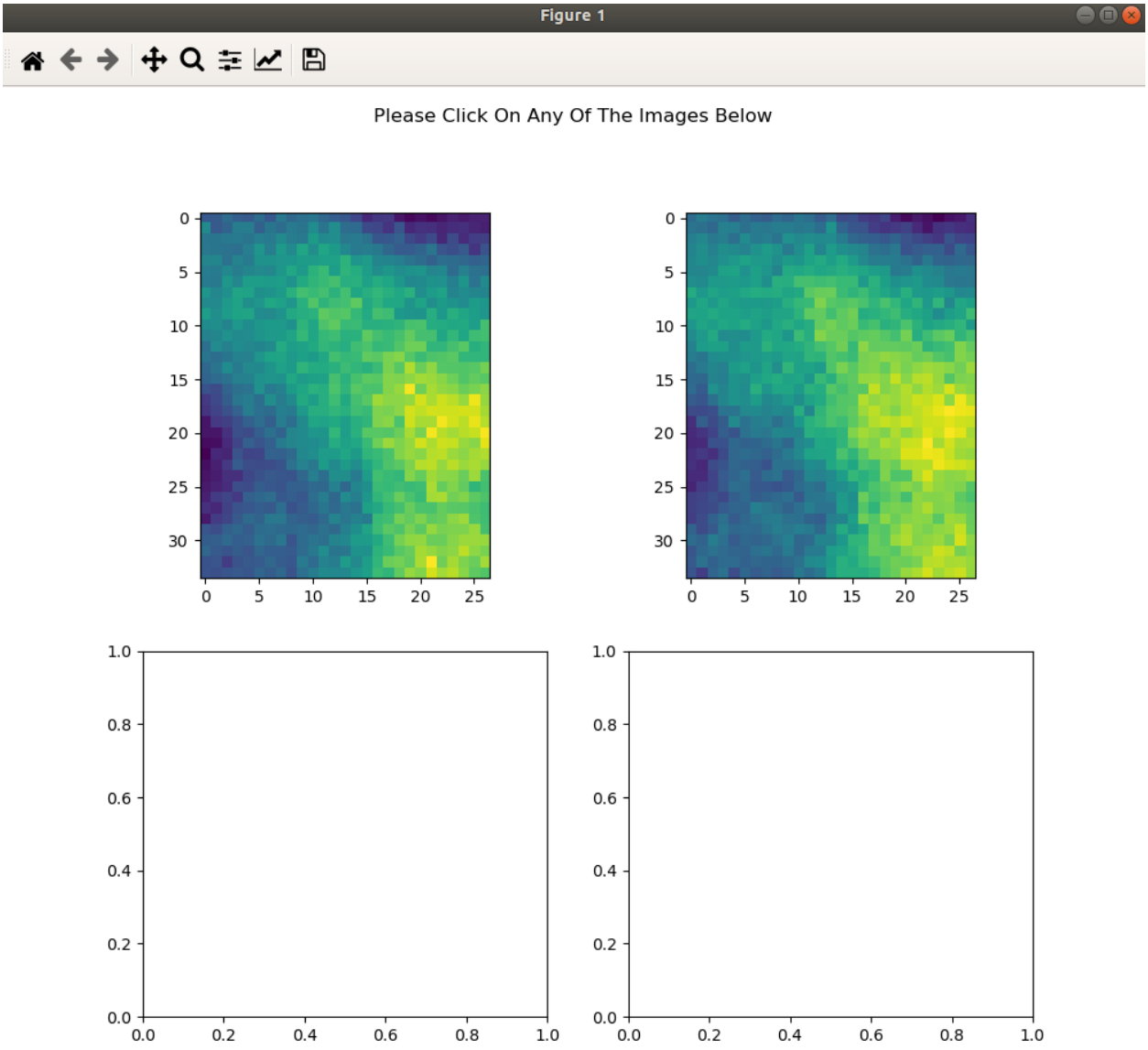
In order to acquire reliable spectra, **it is important that the frames be aligned properly**. Thermal expansion, motor slop, sample damage and imperfect microscope alignment can all cause frames to be misaligned. **It is almost always necessary to align the frames before performing any of the subsequent steps.**

Aligning the scan can be carried out through the following code and following the GUI. Alignment can only be done of the Fluorescence images or the Region of Interest images set in the setup_det_chan() function. User will define which one to use in the GUI. Once all alignment centers have been set, it is ok to just quit out of the windows.

```
from sxdm import *
# Select an imported hdf file to use
test_fs = SXDMFrameset(file="...")

# Run through five passes of the default phase correlation
test_fs.alignment(reset=False)
```

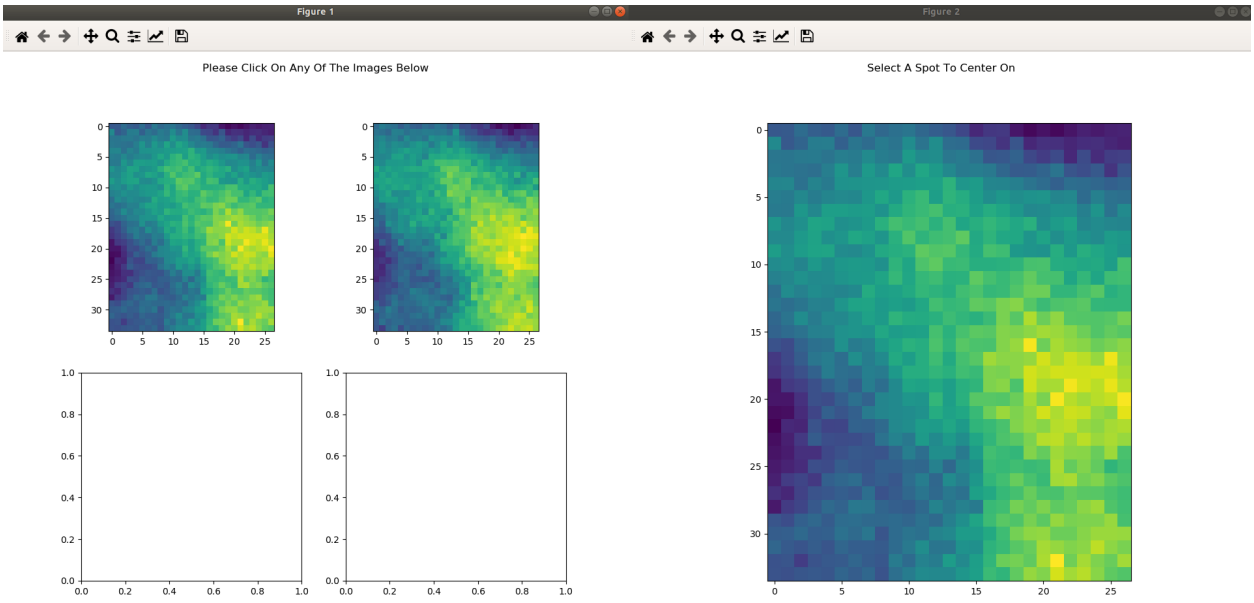
**Brings Up All Fluor Maps**




**User Select Center**

**Showing All Centers**

`reset` (bool) - if you would like to completely reset the alignment make this equal True




**Note: if you import new scan numbers you must make sure reset=True for the first alignment**

## 3.6 Diffraction Axis Values

To determine the chi bounds (angle bounds) for the detector diffraction images as well as determining the numerical aperture, focal length, and instrumental broadening in pixels.

```
test_fs.chi_determination()
```

angle difference (in degrees) from the left/bottom hand side of the detector to the right/top `test_fs.chi` focal length in millimeters can be called with `test_fs.focal_length_mm` numberical aperature in millirads can be called with `test_fs.NA_mrads` instrumental broadening radius in pixels of the diffraction image can be called with `test_fs.broadening_in_pix`

## 3.7 Region Of Interest Analysis

### 3.7.1 Description

This allows the User to section off multiple areas of the diffraction pattern and create heat maps showing which areas of the Field of View light up these diffraction bounding boxes.

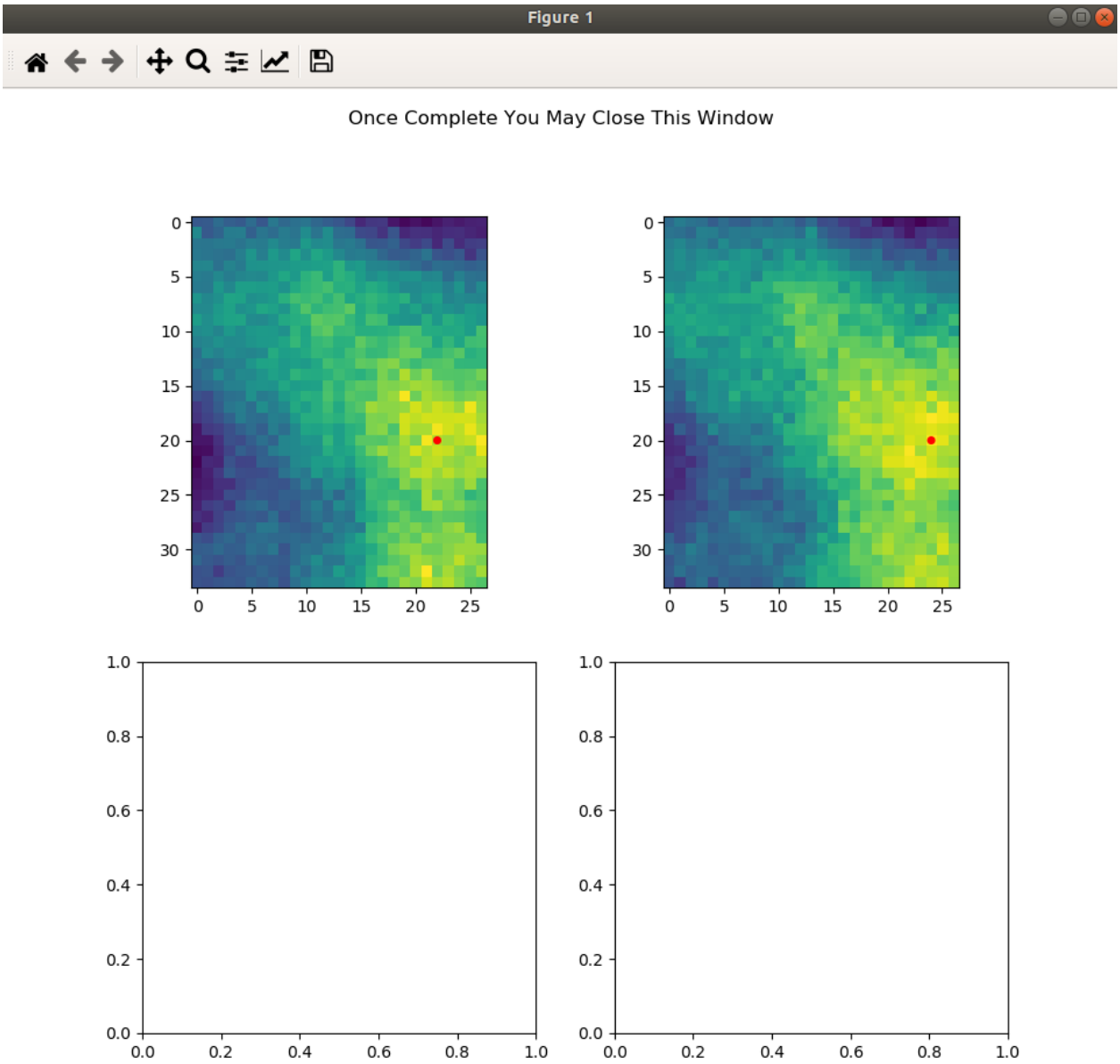


Figure 1
Once Complete You May Close This Window
0
5
10
15
20
25
30
0
5
10
15
20
25
0
5
10
15
20
25
30
0
5
10
15
20
25
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

### 3.7.2 Segmentation

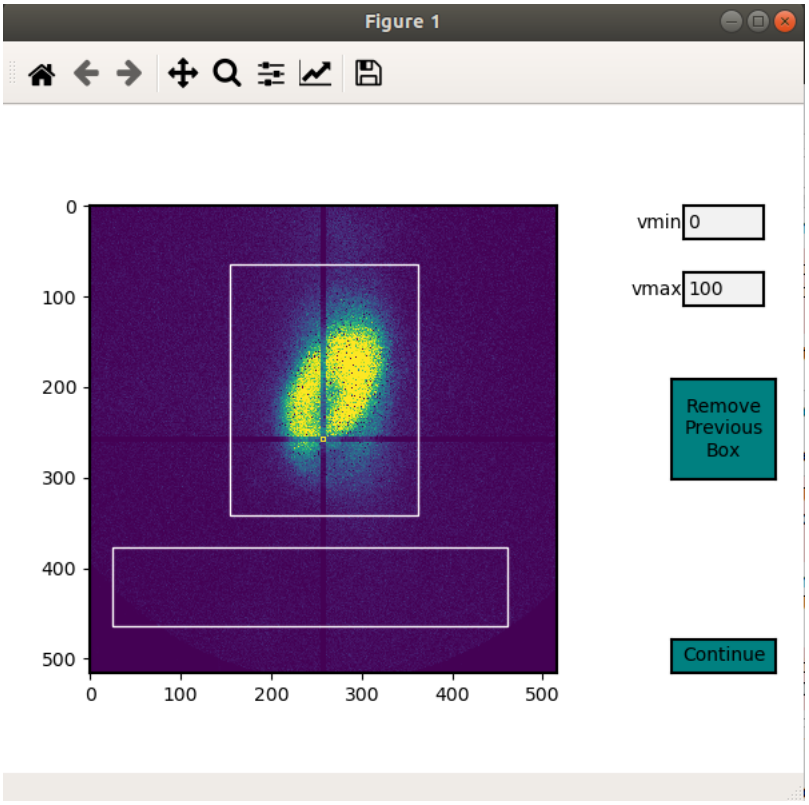
In order for the program to determine a region of interest the User must define areas of interest. This GUI allows the User to define as many Region Of Interests as they please in the diffraction image. Then upon running the Analysis portion, the program will determine the summed value of these regions, plot them, as well as normalize.

Through a GUI the User can select multiple region of interests from the summed diffraction pattern. Set the `diff_segmentation=True` in the `test_fs.region_of_interest()` function for this analysis to be carried out.

```
# Click and drag on the GUI interface to make roi bounding boxes
test_fs.roi_segmentation(bkg_multiplier=1, restart=False)
```

`bkg_multiplier` (int) - an integer value applied to the backgound scans

`restart` (bool) - if set to True this will reset all the segmentation data




**Note:** If the program throws image_array doesnt exist run *create_imagearray(test_fs)*

If the program throws scan_background doesnt exist run *scan_background(test_fs)*

### 3.7.3 Analysis

Allows the User to create new ROI maps for all the imported scans in the frameset. This will handle hot and dead pixels as well as show the user the true gaussian distribution of the fields of view.

```
test_fs.region_of_interest(rows, columns,
                           med_blur_distance=9,
                           med_blur_height=100,
                           bkg_multiplier=1,
                           diff_segmentation=True,
                           slow=False)
```

`rows` (int or tuple) - the total amount of rows the User would like to analyze 25 or (10,17)

`columns` (int or tuple) - the total amount of columns the User would like to analyze 25 or (10,17)

`med_blur_distance` (odd int) - the chunksize for the median_blur() function

`med_blur_height` (int) - the amount above the mean to carry out a median blur - selective median_blur option only

`bkg_multiplier` (int) - the multipler given to the backgound scans

`diff_segmentation` (bool) - if False the program will skip the segmentation analysis

`slow` (bool) - defaults to multiprocess data. If the program uses too much RAM the User can set this value to True to slow down the analysis and save on RAM

To obtain the results from the ROI Analysis use the *create_roi()* function.

```
output = create_rois(test_fs.roi_results)
```

---

**Note: Extremely Large Values??**

If the np.nansum(output, axis=(0,1)) values are too high (1e+285) this is due to poor hot pixel removal. Make sure you are using the *selective* median blur algorithm and lower your median_blur_height value. Also, please see the **Viewer** section for more details.

---

## 3.8 Centroid Analysis

### 3.8.1 Description

This allows the User to determine the diffraction centroid for each pixle in a particular Field of View

### 3.8.2 Analysis

The centroid analysis function can be called through

```
test_fs.centroid_analysis(rows,
                          columns,
                          med_blur_distance=9,
                          med_blur_height=10,
                          stdev_min=25,
                          bkg_multiplier=9)
```

`rows` - total amount of rows in the scans - can also be a tuple of ints

`columns` - total amount of columns in the scans - can also be a tuple of ints

`med_blur_distance` (odd int) - the chunksize for the median_blur() function

`med_blur_height` (int) - the amount above the mean to carry out a median blur - selective median_blur option only

`bkg_multiplier` (int) - the multipler given to the backgound scans

`stdev_min` (int) - the minimum standard deviation of a spectrum which is used to crop signals for centroid determination

`slow` (bool) - defaults to multiprocess data. If the program uses too much RAM the User can set this value to True to slow down the analysis and save on RAM

---

**Note: Unsure About Dimension Size**

If you are unsure of the dimension sizes call `test_fs.frame_shape()`. The first number is the number of scans, the second number is the about of rows + 1, and the third number is the number of columns + 1

---

---

**Note: Difference Between slow=False and slow=True**

The above function calls one of two functions. Either the `centroid_pixel_analysis()` function and vectorizes it for moderate run times with excellent RAM management (1-2GB). Or this will call the `centroid_pixel_analysis_multi()` function which will multiprocess the dataset, but uses considerably more RAM (6-8GB). Analysis route determine by slow bool value.

---

---

**Note: What Is The test_fs.results Variable**

Sets the `test_fs.results` value where the user can return the results of their analysis. Outputs - [pixel position, zero, median blurred x axis, median blurred y axis, truncated x axis for centroid finding, x axis centroid value, truncated y axis for centroid finding, y axis centroid value, summed diffraction intensity]

---

## 3.9 General User Analysis

Sometimes the built in functions do not align with Users diffraction analysis goals. For this there is a general multiprocessing tool for pixel by pixel diffraction pattern analysis.

### 3.9.1 Standard Set Up

This creates the User defined frameset

```
from sxdm import *

test_fs = SXDMFrameset(file'/path/to/file.h5',
            dataset_name='user_dataset_name',
            scan_numbers=[1, 2, 3, 4, ...],
            fill_num=4,
            restart_zoneplate=False,
```

(continues on next page)

(continued from previous page)

```
            median_blur_algorithm='scipy',
            )
```

### 3.9.2 Defining a Function

The User will have to define a function that will be applied to the each background corrected diffraction images. If the User would like to perform operations on the Summed Diffraction Pattern please write in *summed_dif = np.sum(each_scan_diffraction_post_bk_sub, axis=0)* into your first line of your function.

```
def do_something(each_scan_diffraction_post_bk_sub, inputs):
    """
    each_scan_diffraction_post_bk_sub (preset default)
        - This is an automatic input that has to come first. We are passing in
        - the background corrected diffraction patterns for each test_fs.scan_numbers

    inputs (list of ints, ex. [1, 2, 3, 4])
        - the user defined inputs used to split up into function definitions
        - must be static values

    """

    summed_dif = np.sum(each_scan_diffraction_post_bk_sub, axis=0)

    adding, subtracting, dividing, multiplying = inputs

    first = np.add(summed_diff, adding)
    second = np.subtract(first, subtracting)
    third = np.divide(second, dividing)
    fourth = np.multiply(third, multiplying)

    return fourth, third, second, first

analysis_output = do_something(summed_dif, inputs)
```

### 3.9.3 Creating A .tif Image Array

The program needs to have locations for the diffraction.tif images. This creates a centered array for all the locations. The User can choose which scan they would like to center around.

```
create_imagearray(test_fs)
```

### 3.9.4 Implementing General Multiprocessing

This will allow the User to carry out a multiprocesses analysis of the user defined function across all pixels.

```
# Iterate through the first 10 rows and columns
# OR iterate through rows # - # and columns # - #

rows = 10       # to iterate through row 0 - row 10
# OR  set value to (1, 5) - iterates through row 1 - row 5
```

(continues on next page)

(continued from previous page)

```
columns = 10    # to iterate through col 0 - col 10
# OR  set value to (7, 12) - iterates through col 7 - col 12

inputs = [1, 3, 5, 7]

output = general_analysis_multi(test_fs,
                                rows,
                                columns,
                                do_something,
                                inputs,
                                bkg_multiplier=0)


# The output has a general formula [(row, column), analysis_output]
```

### 3.9.5 Conveniently Return General Analysis Values

```
# Define the analysis outputs: must be in the same order as your original function␣
↪output
user_acceptable_values = ['fourth', 'third', 'second', 'first']

# Return values
all_values = general_pooled_return(output, 'fourth', user_acceptable_values)

# You can also call 'row_column' or 'help' to show the row and column locations or a␣
↪list of all acceptable values
# Both 'row_column' and 'help' are created automatically. DO NOT add them to the user_
↪acceptable_values
row_column_values = general_pooled_return(output, 'row_column', user_acceptable_
↪values)
```

**Note:** A built in utility checks the computer RAM usage for the User. If the User's function requires a substantial amount of RAM, the program will default to *analysis_output = False*. This avoids computer crashes. A warning will also be thrown to the User. To change this value one must go to *~/sxdm/sxdm/generalize.py/general_pixel_analysis_multi* and change the *90* in *if ram_check() > 90:* to the **max percent** of the computers RAM the User would like to abort analysis at.

## 3.10 Retrieving Imported Data

### 3.10.1 Return Detector Data - Before Users Set Up SXDMFrameset

```
scans = [1, 2, 3, 4, 5]
string_scans = scan_num_convert(scans)
return_det(file, string_scans, group='xrf', default=False, dim_correction=False)
```

Returns all information for a given detector channel for the array of scan numbers.

`file` - test_fs.file

`scan_numbers` - test_fs.scan_numbers

`group` - Examples: filenumber, sample_theta, hybrid_x, hybrid_y, fluor, roi, mis, xrf

`default` - if True this will default to the first fluorescence image

`dim_correction` - if True this will add empty rows and columns to smaller datasets to make them the same shape

### 3.10.2 Return Detector Data - After Users Set Up SXDMFrameset

```
test_fs = SXDMFrameset(*args)

file = test_fs.file
scan_numbers = test_fs.scan_numbers

return_det(file, scan_numbers, group='fluor', default=False)
```

Returns all information for a given detector channel for the array of scan numbers.

`file` - test_fs.file

`scan_numbers` - test_fs.scan_numbers

`group` - Examples: filenumber, sample_theta, hybrid_x, hybrid_y, fluor, roi, mis, xrf

`default` - if True this will default to the first fluorescence image

`dim_correction` - if True this will add empty rows and columns to smaller datasets to make them the same shape

### 3.10.3 Centering Detector Data

```
centering_det(test_fs, group='fluor', center_around=False, summed=False,␣
↪default=False)
```

This returns the User defined detector for all scans set in the test_fs.scan_numbers and centers them around a User defined centering scan index

`self` - the SXDMFrameset

`group` - a string defining the group value to be returned filenumber, sample_theta, hybrid_x, hybrid_y, fluor, roi

`center_around` - if this is set to -1, arrays will not be shifted

`summed` - if True this will return the summed returned detector value (summed accross all scans)

`default` - if True this will choose the first fluor or first ROI

---

**Note:** The centered file numbers are usually stored as test_fs.im_array

---

### 3.10.4 Show HDF5 File Groups

```
h5group_list(file, group_name='base')
```

This allows the User to view the group names inside the hdf5 file. 'base' shows the topmost group. If it errors this means you have hit a dataset and need to call the h5grab_data() function.

`file` - test_fs.file

`group_name` - /path/to/group/

### 3.10.5 Return HDF5 File Data

```
h5grab_data(file, data_loc)
```

This will grab the data stored in a group. If it errors this means you are not in a dataset directory inside the hdf5 file.

`file` - test_fs.file

`data_loc` - /path/to/data

### 3.10.6 Show Alignment Data

```
grab_dxdy(self)
```

This returns the dx and dy centering values that are stored from the alignment function

`self` - the SXDMFrameset

### 3.10.7 Read HDF5 Group Attributes

```
h5read_attr(file, loc, attribute_name)
```

This returns the attribute value stored

`file` - test_fs.file

`loc` - '/path/to/group/with/attribute'

`attribute_name` - 'the_attribute_name'

### 3.10.8 Find Frameset Dimensions

```
test_fs.frame_shape()
```

This returns the image dimensions for the SXDMFrameset class object

### 3.10.9 Calculate Background and FileNumber Locations

```
test_fs.ims_array()
```

This will auto load/calculate the background images and the image location array

### 3.10.10 Show Raw .tif Image Dimensions

```
test_fs.image_data_dimensions()
```

This will return the diffraction image dimensions

### 3.10.11 Pixel Analysis

If the user would like to return a certain pixel analysis value they can use the `pixel_analysis_return()` function to achieve this. Returns a dictionary of entries

```
#'row_column',
#'summed_dif', - auto set to 0 for saving RAM usage
#'ttheta',
#'chi',
#'ttheta_corr',
#'chi_corr',
#'ttheta_cent',
#'chi_cent',
#'roi'
```

### 3.10.12 Saving and Reloading Data

Saves `self.results` to the `test_fs.saved_file` - this value/file is automatically created in the initial SXDMFrameset setup

```
test_fs.save()
```

To reload saved data in the test_fs.saved_file run

```
test_fs.reload_save()
```

This will load the results to test_fs.results

CHAPTER

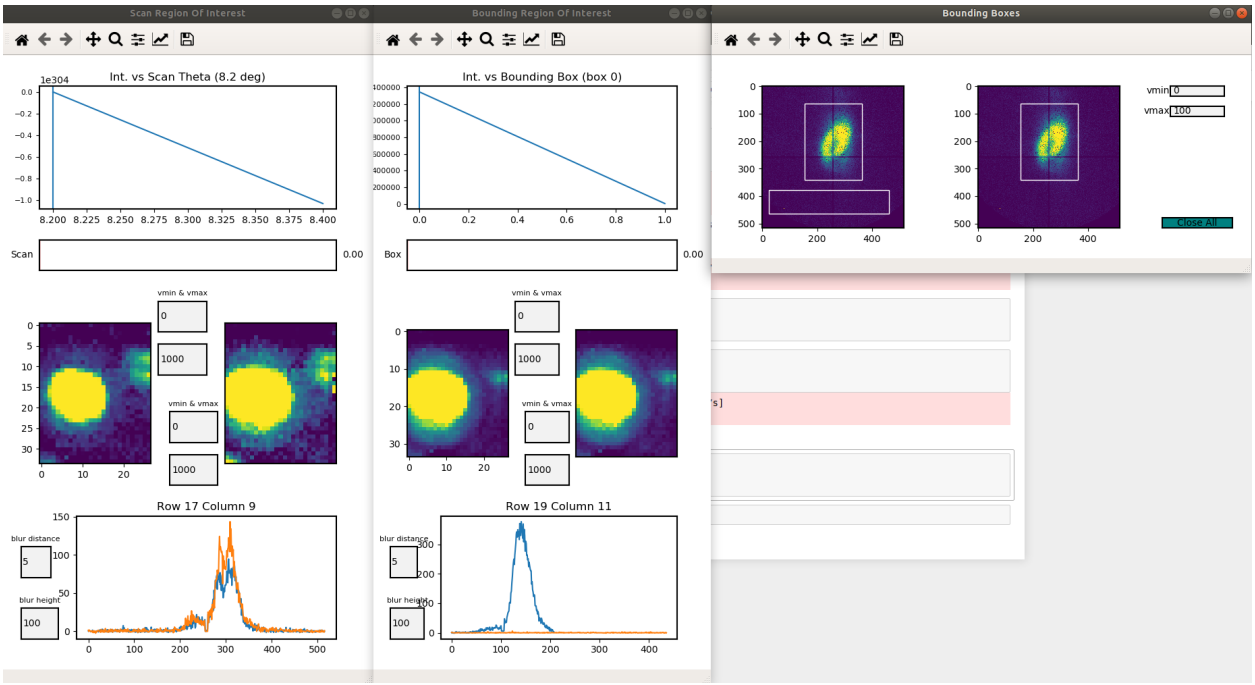
FOUR

# VISUALIZATION OF RESULTS

## 4.1 Region of Interest Viewer

This will bring up the default Region Of Interest viewer, once the `test_fs.roi_results` have been calculated.

```
test_fs.roi_viewer()
```




**Warning:** Please use the *close all* box to close all windows. If not the program may keep cached data

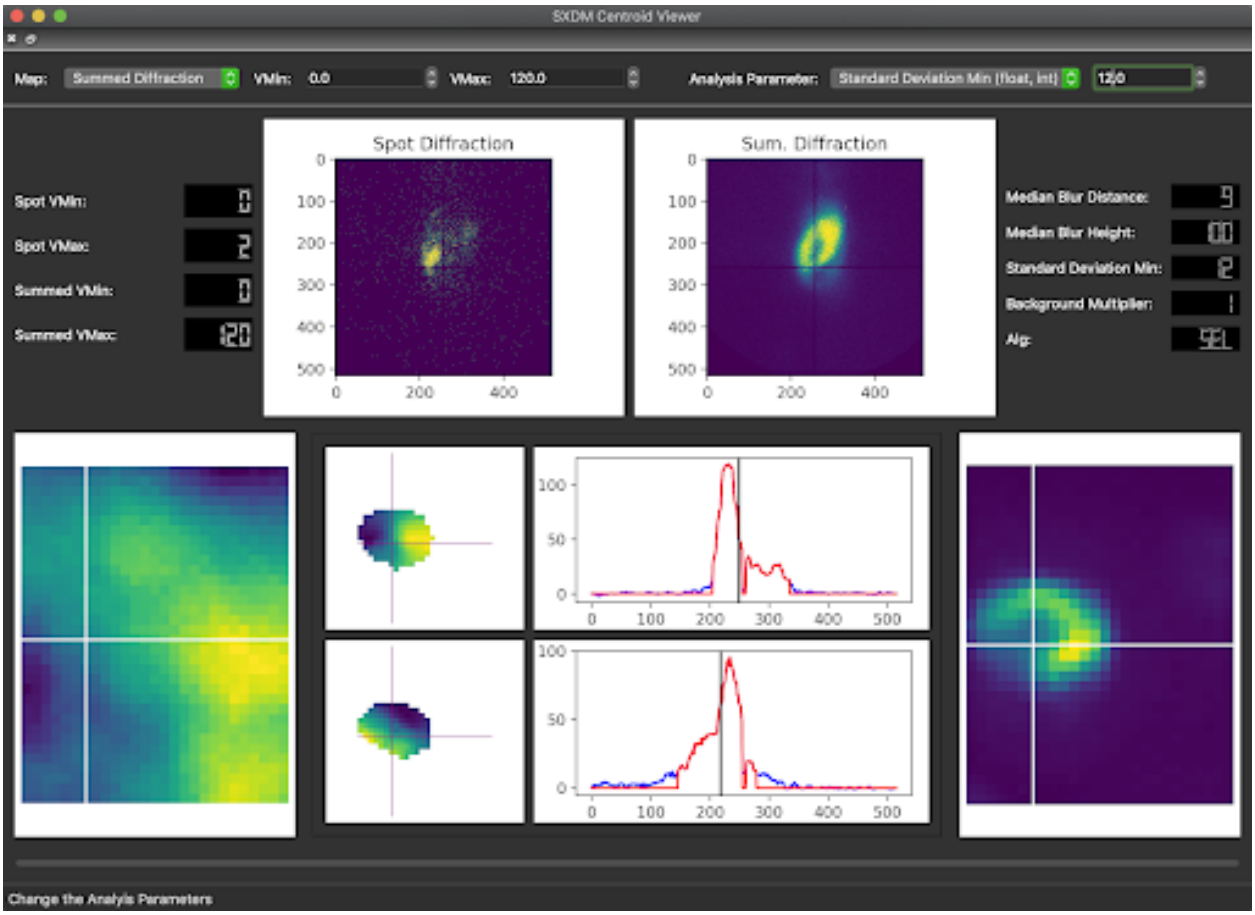
**Note:** Median Blur Height will be only be enabled for a *median_blur_algorithm='selective'*

**Note:** User can move sliders as well as click on maps to show intensity values/scan values

## 4.2 Centroid Analysis Viewer

This will bring up the default Region Of Interest viewer, once the `test_fs.centroid_results` have been calculated.

```
test_fs.centroid_viewer()
```




**Note:** Median Blur Height will be only be enabled for a *median_blur_algorithm='selective'*

**Note:** User can click on maps to display how the program is interpreting pixel data.

## 4.3 Return Raw Centroid Map Values

Takes the test_fs.results and a User defined map_type and returns either the centroid data or the ROI data for the `test_fs.resutls` variable.

```
centroid_roi_map(results, map_type)
```

`results` (nd.array) - the test_fs.results value

`map_type` - acceptable values - full_roi, chi_centroid, ttheta_centroid

## 4.4 Make New Bound For Centroid Maps

The centroid maps are in values associated with their centroid position of the .tif image dimensions (usually 516, 516). To change what the bound/values are for the centroid values the user can set values for user_map and new_bounds to rebound the centroid maps.

```
maps_correct(user_map, new_bounds)
```

`user_map` - the output of the centroid_roi_map() function

`new_bounds` - np.linspace(lowerbound, higherbound, dim of image)

## 4.5 Errors

The main visual error will be the Psyduck error. This occurs when the program cannot properly load the corresponding data for a given plot. Psyduck is confused and so is everyone else.

CHAPTER

FIVE

# SXDM

## 5.1 sxdm package

### 5.1.1 Submodules

### 5.1.2 sxdm.alignment module

sxdm.alignment.**alignment_function**(*self*)

Allows the user to align the scans based on Fluorescence or Region Of Interest.

Sets alignment variables and stores them in designated .h5 file. Easier to reload alignment data or redo alignment data

**Parameters** **self** ((SXDMFrameset)) – the sxdmframeset object

**Returns**

**Return type** Nothing

sxdm.alignment.**reset_dxdy**(*self*)

Resets the dxdy movements used for alignment of the frames

**Parameters** **(SXDMFrameset)** (*self*) – the sxdmframeset object

**Returns**

**Return type** Nothing

### 5.1.3 sxdm.background module

sxdm.background.**scan_background**(*self*, *amount2ave=3*, *multiplier=1*)

Create background images for each scan.

Takes the first "amount2ave" and last "amount2ave" images of each scan and average them together to each scan. Create a dictionary of these background images.

**Parameters**

- **self** ((SXDMFrameset)) –
- **amount2ave** ((int)) – amount of images from the beginning and enf of each scan to average together
- **multiplier** ((int/float)) – multiplier for each background scan

**Returns**

**Return type** A dictionary of background images with entries for each scan number

sxdm.background.**scan_background_finder**(*destination*, *background_dic*)
The destination is the scan numbers associated with a given pixel location. This will take all scans in a pixel location and return a numpy array of their appropriate background images.

**Parameters**

- **destination** ((numpy array)) – the output of h5get_image_destination(self, pixel) list of scan numbers which the user wants to get the background images for
- **background_dic** ((dic)) – the dictionary otuput from the scan_background() function

**Returns**

**Return type** A numpy array of background images corresponding to the scans in the destination input

### 5.1.4 sxdm.chi_determination module

**class** sxdm.chi_determination.**Chi_FiguresClass**
Bases: object

A class function that allows the code to store figure data. Makes it easier to transfer data between figures

sxdm.chi_determination.**broadening_in_pixles**(*self*)
Determine the instrumental broadening in pixels as well as the focal length and numberical aperature

**Returns**

- *Nothing*
- *Sets*
- ====
- *self.focal_length_mm*
- *self.NA_mrads*
- *self.broadening_in_pix*

sxdm.chi_determination.**chi_function**(*self*)
Runs all code necessary to determine the chi bounds

**Parameters** **(SXDMFrameset)** (*self*) – the sxdmframeset

**Returns**

**Return type** Nothing

sxdm.chi_determination.**chis**(*self*)
Calculates the bounds for the detector

**Parameters** **(SXDMFrameset)** (*self*) – the sxdmframeset

**Returns**

- *Nothing*
- *Prints and Sets*
- ======
- *self.chi value,*
- *self.focal_length,*

- *self.NA_mrads (numberical aperature),*
- *self.broadening_in_pix of the detector*

sxdm.chi_determination.**closebtn_press**(*event*, *self*, *chi_figures*)
Set the chi angle difference, position difference, and image dimension upon closing the figure

**Parameters**

- **(matplotlib event)** (*event*) – matplotlib event
- **(SXDMFrameset)** (*self*) – the SXDMFrameset)
- **(Chi_FiguresClass)** (*chi_figures*) – the chi_figuresclass

**Returns**

**Return type** Nothing - runs chis() function. see documentation for values set

sxdm.chi_determination.**closebtn_start**(*chi_figures*, *btn_ax*)
Set up the close button

**Parameters**

- **(Chi_FiguresClass)** (*chi_figures*) – the chi_figuresclass
- **(matplotlib_axis)** (*btn_ax*) – the figure axis corresponding to the close button

**Returns**

**Return type** Nothing

sxdm.chi_determination.**display_first_images**(*chi_figures*)
From the chi figure class set up the first set of figures

**Returns**

**Return type** Nothing

sxdm.chi_determination.**display_second_images**(*chi_figures*)
Display the second figure

**Parameters** **(Chi_FiguresClass)** (*chi_figures*) – the chi_figuresclass

**Returns**

**Return type** Nothing

sxdm.chi_determination.**first_chi_figure_setup**(*self*)
Setting up the chi figure GUI

**Parameters** **(SXDMFrameset)** (*self*) – the sxdmframeset object

**Returns**

**Return type** The Chi_FigureClass object created by the function

sxdm.chi_determination.**idx_tb_setup**(*vmin_spot_ax*, *vmax_spot_ax*)
Set up the index textboxes DEPRECIATED

**Returns**

**Return type** position 1 and position 2 textboxes for index values

sxdm.chi_determination.**minmax_tb_setup**(*vmin_spot_ax*, *vmax_spot_ax*)
Set up the minmax textboxes

**Parameters**

- **(matplotlib axis)** (*vmax_spot_ax*) – the figure axis that corresponds to the vmin value of the spot diffraction
- **(matplotlib axis)** – the figure axis that corresponds to the vmax value of the spot diffraction

**Returns**

**Return type** vmin and vmax textboxes

sxdm.chi_determination.**pos_tb_setup**(*pos1_ax*, *pos2_ax*)
Set up the position text boxes

**Parameters**

- **(matplotlib axis)** (*pos2_ax*) – the figure axis corresponding to the vertical line position of the first (left) image
- **(matplotlib axis)** – the figure axis corresponding to the vertical line position of the second (right) image

**Returns**

**Return type** position 1 and position 2 textboxes for index values

sxdm.chi_determination.**return_chi_images**(*file*, *chi_figures*)
Return the detector movement data

**Parameters**

- **(str)** (*file*) – the file the user would like to find the detector images from
- **(Chi_FigureClass)** (*chi_figures*) – the figure class. used to make it easier to move data around

**Returns**

**Return type** Nothing - sets chi_fiures.images value

sxdm.chi_determination.**return_chi_images_loc**(*file*, *chi_figures*, *detector_channel_loc='detector_channels/detector_scan/Main_Scan'*, *zfill_num=4*)
Grabs the image location for the detector sweep scan

**Parameters**

- **file** ((str)) – the .h5 file associated with the dataset
- **chi_figures** ((Chi_FigureClass)) – the Chi_FigureClass allowing transfer of data from figure to figure
- **detector_channel_loc** ((str)) – the h5 group location to the detector sweep scan used for chi bounds determination
- **zfill_num** ((int)) – the integer value for how many digits the scan number must have

**Returns**

**Return type** Nothing - sets the chi_figures.scan_theta and chi_figures.images_location valuess

sxdm.chi_determination.**second_change**(*event*, *chi_figures*)
When the second figure values change, update the second figure

**Parameters**

- **(matplotlib event)** (*event*) – matplotlib event
- **(Chi_FiguresClass)** (*chi_figures*) – the chi_figuresclass

**Returns**

**Return type** Nothing

sxdm.chi_determination.**second_chi_figure_setup**(*chi_figures*)
Setting up the second figure

**Parameters** **(Chi_FigureClass)** (*chi_figures*) – the chi_figuresclass

**Returns**

**Return type** Nothing

sxdm.chi_determination.**vs_change**(*event*, *chi_figures*)
When the user changes the vmin or vmax update the figures

**Parameters**

- **(matplotlib event)** (*event*) – matplotlib event
- **(Chi_FiguresClass)** (*chi_figures*) – the chi_figuresclass

**Returns**

**Return type** Nothing

### 5.1.5 sxdm.clicks module

sxdm.clicks.**check_mouse_ax**(*event*, *self*)
Grabs the mouse axes position

**Parameters**

- **(matplotlib event)** (*event*) – the matpotlib event
- **(SXDMFrameset)** (*self*) – the sxdmframeset

**Returns**

**Return type** Nothing - just checks which axis the mouse is in

sxdm.clicks.**fig1_click**(*event*, *self*, *fig*, *images*)
Deal with event clicks in the first figure and redraw plots

**Parameters**

- **(matplotlib event)** (*event*) – matplotlib event
- **(SXDMFrameset)** (*self*) – the sxdmframeset
- **(matplotlib figure)** (*fig*) – the figure to be clicked on
- **(nd.array)** (*images*) – the images to display

**Returns**

**Return type** Nothing - deals with the clicks on the first image

sxdm.clicks.**fig_leave**(*event*, *self*)
If the user leaves an axis set the self.viewer_currentax to None :param event (matplotlib event): the matplotlib event :param self (SXDMFrameset): the sxdmframeset

**Returns**

**Return type** Nothing

sxdm.clicks.**onclick_1**(*event*, *self*)
Deal with click events for first figure. Only allows for 30 figures

Once you click on figure one open up a second figure with the figure you just clicked on, but larger

**Parameters**

- **event** ((matplotlib event)) – not 1100% sure. Needed to be the first param in a matplotlib event
- **self** ((SXDMFrameset)) – An SXDMFrameset which allows the function to store click values

**Returns**

**Return type** Nothing

sxdm.clicks.**onclick_2**(*event*, *self*)
Determine the pixel location the user has clicked on.

From the second figure that popped up, once you click on it, determine the location of that click

**Parameters**

- **(matplotlib event)** (*event*) – Unsure. Needed to be the first variable in a matplotlib event click
- **(SXDMFrameset)** (*self*) – An SXDMFrameset that allows the clicks to be saved

sxdm.clicks.**save_alignment**(*event*, *self*)
Allow the alignment click information to be saved to self.file

**Parameters** **(SXDMFrameset)** (*self*) – the sxdmframeset object

**Returns**

- *Nothing. It just stores /dxdy to the self.file as well*
- *as adding alingment_group and alignment_subgroup attributes*

### 5.1.6 sxdm.det_chan module

sxdm.det_chan.**add_column**(*im*, *max_column*)
Add a numpy.nan column to an image array based on how many columns in the max amount of columns

**Parameters**

- **(a 2D array)** (*im*) – the image array to add columns to
- **(int)** (*max_column*) – the final amount of columns the user want to make the im array

**Returns**

**Return type** a new im array with the User selected max_column

sxdm.det_chan.**add_row**(*im*, *max_row*)
Add a numpy.nan rows to an image array based on how many rows in the max amount of rows

**Parameters**

- **(a 2D array)** (*im*) – the image array to add columns to
- **(int)** (*max_row*) – the final amount of rows the user want to make the im array

**Returns**

**Return type** a new im array with the User selected max_row

sxdm.det_chan.**del_det_chan**(*file*)
Deletes the /detector_channels group :param file (str): the .h5 file you would like to delete the detector_channel group from

**Returns**

**Return type** Nothing

sxdm.det_chan.**det_dim_fix**(*array*, *max_row*, *max_column*)
Add columns and rows to image matrix to make everything the same dimensions

**Parameters**

- **(np.ndarray)** (*array*) – a 3D array of images
- **(int)** (*max_column*) – the max amount of rows the User would like to make all the images
- **(int)** – the max amount of columns the User would like to make all the images

**Returns**

**Return type** the 3D image array with dimensions (#images, max_row, max_column)

sxdm.det_chan.**disp_det_chan**(*file*)
Allows the user to quicky see what all values are set to in the detector_channel group

**Parameters** **(str)** (*file*) – the .h5 file you would like to delete the detector_channel group from

**Returns**

- *prints the current values of the detector_channel group. If no values are currently set it displays something the*
- *User can copy and paste into a cell to set the appropriate dectector values*

sxdm.det_chan.**max_dims**(*array*)
Get the max dimensions for an array of 2D images

**Parameters** **(np.ndarray)** (*array*) – a 3 dimensional array

**Returns**

**Return type** the max rows and the max columns round for the 3 dimensionsal array

sxdm.det_chan.**return_det**(*file*, *scan_numbers*, *group='fluor'*, *default=False*, *dim_correction=True*)
Returns all information for a given detector channel for an array of scan numbers

**Parameters**

- **file** ((str)) – the .h5 file location
- **scan_numbers** ((np.array)) – an array of scan numbers the user wants to get a specific detector channel information for
- **group** ((str)) – a string corresponding to a group value that the user wants to return
- **default** ((bool)) – If True this will default to the first acceptable detector_channel in the hdf5 file
- **dim_correction** ((bool)) – If False this will not add rows and columns to the returned array to make them all the same shape

**Returns**

**Return type** The specified detector channel for all specified scan numbers, the .mda detector value

sxdm.det_chan.**setup_det_chan**(*file*, *fluor*, *roi*, *detector_scan*, *filenumber*, *sample_theta*, *hybrid_x*, *hybrid_y*, *mis*, *xrf*)

Sets detector channel information and stores it in the .h5 file

**Parameters**

- **file** ((str)) – the hdf file path
- **fluor** ((dic)) – a dictionary entry with all the Fluorescence images names as well as their corresponding .mda detector channel vale
- **roi** ((dic)) – a dictionary entry with all the Region of Interest images names as well as their corresponding .mda detector channel vale
- **detector_scan** ((int)) – the dectector channel that corresponds to the scanning of the detector - used to determine detector dimensions
- **filenumber** ((int)) – the dectector channel that corresponds to the image file numbers
- **sample_theta** ((int)) – the detector channel that corresponds to the sample theta
- **hybrid_x** ((int)) – the detector channel that corresponds to the hybrid_x
- **hybrid_y** ((int)) – the detector channel that corresponds to the hybrid_y
- **mis** ((dic)) – a miscellaneous dictionary with entries of detector channels that might be usefull. ex. 2Theta, Ring_Current
- **xrf** ((dic)) – an x-ray fluorescence dictionary with entries of the detector channels CORRESPOND TO THE DETECTOR CHANNELS UNDER THE 'xrf' HEADING THE THE HDF FILE!!!

**Returns**

**Return type** Nothing

sxdm.det_chan.**space_check**(*fluor*, *roi*, *detector_scan*, *filenumber*, *sample_theta*, *hybrid_x*, *hybrid_y*, *mis*, *xrf*)

Based on the input variables this checks to see if the user has put any spaces into their group names.

Warns the user that they are using spaces

**Parameters**

- **fluor** ((dic)) – a dictionary entry with all the Fluorescence images names as well as their corresponding .mda detector channel value
- **roi** ((dic)) – a dictionary entry with all the Region of Interest images names as well as their corresponding .mda detector channel vale
- **detector_scan** ((int)) – the dectector channel that corresponds to the scanning of the detector - used to determine detector dimensions
- **filenumber** ((int)) – the dectector channel that corresponds to the image file numbers
- **sample_theta** ((int)) – the detector channel that corresponds to the sample theta
- **hybrid_x** ((int)) – the detector channel that corresponds to the hybrid_x
- **hybrid_y** ((int)) – the detector channel that corresponds to the hybrid_y
- **mis** ((dic)) – a miscellaneous dictionary with entries of detector channels that might be usefull. ex. 2Theta, Ring_Current
- **xrf** ((dic)) – an x-ray fluorescence dictionary with entries of the detector channels CORRESPOND TO THE DETECTOR CHANNELS UNDER THE 'xrf' HEADING THE THE HDF FILE!!!

**Returns**

**Return type** (bool) on whether or not there is a space in any of the dictionary entries given by the user

sxdm.det_chan.**true_filenumbers**(*file*, *scan_numbers*, *shapes*)

Since the .mda files do not store the correct file_names/image_names inside of it. This creates an appropriate file number for each pixel in a scan

**Parameters**

- **(str)** (*file*) – the location of the .h5 file
- **(nd.array)** (*shapes*) – array of all the scan numbers the User wishes to create image numbers for
- **(nd.array)** – the shapes of all the scans
- **Returns** –
- **=======** –
- **nd.array with the .tif image numbers/locations for each voxel of the scan** (*an*) –

### 5.1.7 sxdm.generalize module

sxdm.generalize.**general_analysis_multi**(*self*, *rows*, *columns*, *analysis_function*, *analysis_input*, *bkg_multiplier=0*)

Runs the centroid analysis in a pool.map function

**Parameters**

- **self** ((SXDMFrameset)) – the sxdmframeset object
- **rows** ((int or tup)) – the total amount of rows the User wants to analyze or a tuple of the rows
- **columns** ((int or tup)) – the total amount of columns the User wants to analyze or a tuple of the columns
- **med_blur_distance** ((int)) – the median blur distance - must be odd
- **med_blur_height** ((int)) – the median blur height
- **stdev** ((int)) – the standard deviation used to segment data
- **bkg_multiplier** ((int)) – the multiplier to the background images
- **analysis_function** ((func)) – a function to be passed into the analysis. the program will find the summed diffraction pattern for each pixel, for each scan number of the dataset, background subtract, and pass the summed diffraction into the first analysis_function argument.
- **analysis_input** ((user defined)) – an static input array, value, etc passed into the second argument in the analysis_function.

**Returns**

**Return type** a pooled results from the centroid_pixel_analysis_multi() function

sxdm.generalize.**general_pixel_analysis_multi**(*row*, *column*, *image_array*, *scan_numbers*, *background_dic*, *file*, *analysis_function*, *analysis_input*)

The analysis done on a single pixel

**Parameters**

- **row** ((int)) – the row the User wants to do analysis on
- **column** ((int)) – the column the User wants to do analysis on
- **image_array** ((nd.array)) – the image location array - can be created with create_imagearray(self)
- **scan_numbers** ((nd.array)) – the list of scan numbers used
- **background_dic** ((dic)) – the background scan dictionary entry - can be made with scan_backgrounnd(self)
- **file** ((str)) – the hdf5 file location
- **analysis_function** ((func)) – a function to be passed into the analysis. the program will find the summed diffraction pattern for each pixel, for each scan number of the dataset, background subtract, and pass the summed diffraction into the first analysis_function argument.
- **analysis_input** ((user defined)) – an static input array, value, etc passed into the second argument in the analysis_function.

**Returns**

**Return type** the analysis results as an nd.array

sxdm.generalize.**general_pooled_return**(*results*, *user_val*, *user_acceptable_values*)

Makes it easy to return values from the pooled results from the multi.analysis function

**Parameters**

- **(n dimensional array)** (*results*) – the output from the analysis function
- **(str)** (*user_val*) – a string that defines what the user wants to be returned. Type 'help' for all acceptable values
- **(list)** (*user_acceptable_values*) – an array of all the analysis_function outputs in the order the user has set them as.

**Returns**

**Return type** An n dimensional array consisting of the user selected data output from the multi.analysis function

sxdm.generalize.**general_pre_multi**(*inputs*, *image_array*, *scan_numbers*, *background_dic*, *file*, *analysis_function*, *analysis_input*)

Allows Python to run the roi analysis in a pool.map function

**Parameters**

- **inputs** ((nd.array)) – the iterable inputs of rows and columns
- **image_array** ((nd.array)) – the total locations of all the images - created by create_imagearray(self)
- **scan_numbers** ((nd.array)) – the list of all the scan numbers
- **background_dic** ((dic)) – the dictionary of all the background images - created by scan_background(self)

- **file** ((str)) – the full hdf5 file location
- **analysis_function** ((func)) – a function to be passed into the analysis. the program will find the summed diffraction pattern for each pixel, for each scan number of the dataset, background subtract, and pass the summed diffraction into the first analysis_function argument.
- **analysis_input** ((user defined)) – an static input array, value, etc passed into the second argument in the analysis_function.

**Returns**

**Return type** the results from the general_pixel_analysis_multi() function

### 5.1.8 sxdm.h5 module

sxdm.h5.**close_h5**(*hdf*)
Closing and opened hdf5 file

**Parameters** **(str)** (*file*) – the path to the hdf5 file

**Returns**

**Return type** Closes an opened hdf5 file

sxdm.h5.**h5create_dataset**(*file*, *ds_path*, *ds_data*)
Creates a dataset in the user defined group with data equal to the user defined data

**Parameters**

- **file** ((str)) – the user defined hdf5 file
- **ds_path** ((str)) – the group path to the dataset inside the hdf5 file
- **ds_data** ((nd.array)) – a numpy array the user would like to store

**Returns**

**Return type** Nothing

sxdm.h5.**h5create_file**(*loc*, *name*)
Creates hdf5 file

**Parameters**

- **loc** ((str)) – the location of the hdf5 file
- **name** ((str)) – the name of the hdf5 file WITHOUT .h5 at the end

**Returns**

**Return type** Nothing

sxdm.h5.**h5create_group**(*file*, *group*)
Creates a group based on the Users group input

**Parameters**

- **(str)** (*group*) – the user defined hdf5 file
- **(str)** – the group the user would like to create inside the file

**Returns**

**Return type** Nothing

sxdm.h5.**h5del_data**(*file*, *group*)
Sets all data equal to zero for a given dataset.

**Parameters**

- **file** ((str)) – the hdf5 file path
- **group** ((str)) – the group location in the hdf5 the user wants to delete the data for

**Returns**

**Return type** Nothing

sxdm.h5.**h5del_group**(*file*, *group*)
Deletes the user defined group. DOES NOT REDUCE FILE SIZE

**Parameters**

- **file** ((str)) – the user defined hdf5 file
- **group** ((str)) – the group inside the hdf5 file the user would like to delete

**Returns**

**Return type** Nothing

sxdm.h5.**h5delete_file**(*file*)
Deletes the file set by the user

**Parameters** **file** ((str)) – the full location of the hdf5 file the user would like to delete

**Returns**

**Return type** Nothing

sxdm.h5.**h5get_image_destination**(*self*, *pixel*)
Determine where all the .tif images are for all the scan numbers disregarding the nan values

**Parameters**

- **(SXDMFramset)** (*self*) – the sxdmframset
- **(str array)** (*pixel*) – the image number from each scan that corresponds to a certain pixel

**Returns**

**Return type** All of the diffraction FULL image locations for each scan in a 3D array - excluding the np.nan's

sxdm.h5.**h5get_image_destination_v2**(*self*, *pixel*)
Determine where all the .tif images are for all the scan numbers disregarding the nan values

**Parameters**

- **(SXDMFramset)** (*self*) – the sxdmframset
- **(str array)** (*pixel*) – the image number from each scan that corresponds to a certain pixel

**Returns**

**Return type** All of the diffraction FULL image locations for each scan in a 3D array - excluding the np.nan's

sxdm.h5.**h5grab_data**(*file*, *data_loc*)
Returns the data stored in the user defined group

**Parameters**

- **(str)** (*data_loc*) – the user defined hdf5 file
- **(str)** – the group the user would like to pull data from

**Returns**

**Return type** the data stored int the user defined location

sxdm.h5.**h5group_list**(*file*, *group_name='base'*)
Displays all group members for a user defined group

**Parameters**

- **(str)** (*group_name*) – the path to the hdf5 file
- **(str)** – the path to the group the user wants the Keys for. Set to 'base' if you want the top most group

**Returns**

**Return type** a list of all the subgroups inside the user defined group

sxdm.h5.**h5images_wra**(*file*, *scan*, *im_loc*, *im_name*, *delete=False*, *import_type='uint32'*)
Used to import/delete .tif images into the .h5 file

**Parameters**

- **(str)** (*import_type*) – the user defined hdf5 file
- **(nd.array)** (*scan*) – the scan numbers the user wants to import
- **(str)** – the location of the image file
- **(bool)** (*delete*) – set to True if the user would like to delete the data in the hdf5 file
- **(str)** – string passed into imageio.imread(image).astype(import_type)

**Returns**

**Return type** Nothing

sxdm.h5.**h5path_exists**(*file*, *loc*)
See if the group the user selected exists

**Parameters**

- **(str)** (*loc*) – the path to the hdf5 file
- **(str)** – the location to the group the user wishes to check the status of

**Returns**

**Return type** a bool (True or False) on whether or not the group exists

sxdm.h5.**h5read_attr**(*file*, *loc*, *attribute_name*)
Read an attribute from a user selected group and attribute name

**Parameters**

- **(str)** (*attribute_name*) – the path to the hdf5 file
- **(str)** – the location to the group inside the hdf5 file
- **(str)** – the name of the attribute

**Returns**

**Return type** Attribute value

sxdm.h5.**h5replace_data**(*file*, *group*, *data*)
Replaces all data in a dataset.

Helps with space savings since deleting groups does not decrease the file size. Easier to replace data

**Parameters**

- **(str)** (*group*) – the user defined hdf5 file
- **(str)** – the user defined group in the hdf5 file
- **(nd.array)** (*data*) – the data the user would like to sub in

**Returns**

**Return type** Nothing

sxdm.h5.**h5set_attr**(*file*, *loc*, *attribute_name*, *attribute_val*)
Set and attribute for a User selected group

**Parameters**

- **(str)** (*attribute_name*) – the path to the hdf5 file
- **(str)** – the group location in the hdf5 file
- **(str)** – the name the user wants to set the attribute value of
- **(str or int)** (*attribute_val*) – the value of the attribute

**Returns**

**Return type** Nothing

sxdm.h5.**open_h5**(*file*)
Opening an hdf5 file

**Parameters** **(str)** (*file*) – the path to the hdf5 file

**Returns**

**Return type** The opened hdf5 file

### 5.1.9 sxdm.importer module

sxdm.importer.**images_group_exsist**(*file*, *scan*)
See if the images group exists.

sxdm.importer.**import_images**(*file*, *images_loc*, *scans=False*, *fill_num=4*, *import_type='uint32'*, *delimiter_function=<function delimiter_func>*, *force_reimport=False*)
Allows the user to import all .tif images into the .h5 file

**Parameters**

- **(str)** (*import_type*) – the user defined hdf5 file
- **(str)** – the location of the images folder from the beamline
- **(nd.array)** (*scans*) – the scans the user would like to import
- **(int)** (*fill_num*) – the amount of numbers in the images folders names
- **(str)** – a string value passed into imageio.imread().astype(import_type)
- **(function)** (*delimiter_function*) – a function which determines the image number. redefine this if 26 - ID -C naming scheme changes

- **(bool)** (*force_reimport*) – set to True if you would like to force reimport images

**Returns**

**Return type** Nothing

sxdm.importer.**import_mda**(*mda_path*, *hdf5_save_directory*, *hdf5_save_filename*, *single_file=False*, *maxdims=2*)

Allows the User to import all .mda image and line scan data into an hdf5 format

**Parameters**

- **(str)** (*single_file*) – the string path to the folder holding the .mda files
- **(str)** – the path location where you would like to save your data to EXAMPLE: '/home/Desktop'
- **(str)** – the file inside that path in which you would like to save data to (DO NOT INCLUDE ".h5") EXAMPLE: 'test'
- **(str)** – the location of the file to import
- **(int)** (*maxdims*) – the maximum dimensions for the readMDA function to import from the .mda file

**Returns**

**Return type** Nothing - Will Not Reimport Files

### 5.1.10 sxdm.logger module

sxdm.logger.**initialize_logging**(*self*)

Initiate logging

Not set up

### 5.1.11 sxdm.mis module

sxdm.mis.**array2dic**(*array*)

Turn a numpy array into a dictionary

**Parameters** **(numpy array)** (*array*) – a numpy array of form [(num1, num2), (num3, num4)] that will be turned into a dictionary

**Returns** **a dictionary of form {0**

**Return type** (num1, num2), 1: (num3, num4)``}``

sxdm.mis.**array_shift**(*self*, *arrays2shift*, *centering_idx*)

Translate a 3D stack of numpy arrays to align based on set centering values

**Parameters**

- **(SXDMFrameset)** (*self*) – the sxdmframeset
- **(numpy array)** (*centering_idx*) – the 3 dimensional array the user wants to center around one of the indexes
- **(numpy array)** – the dxdy values (translation values) for each matrix in the array

**Returns**

**Return type** the shifted array based on the dxdy values

sxdm.mis.**centering_det**(*self*, *group='fluor'*, *center_around=False*, *summed=False*, *default=False*)
Return a detector that has been centered around a set value

**Parameters**

- **(SXDMFrameset)** (*self*) – the sxdmframeset
- **(str)** (*group*) – the group name the user would like to center to - acc_vals = fluor and roi
- **(bool)** (*default*) – if this equals -1 then there will be no centering adjustments
- **(bool)** – if True it will sum together all scans after centering
- **(bool)** – if True it will set the fluor image to center on or roi image to view to the first index

**Returns**

- *an array of the fluor images or the roi images for all the SXDMFrameset scans centered around*
- *the users value*

sxdm.mis.**create_file**(*file*)
Create an hdf5 file

**Parameters** **(str)** (*file*) – the path of the hdf5 file the user would like to create

**Returns**

**Return type** Nothing

sxdm.mis.**create_rois**(*self*)
Take the self.roi_results and make them into something more useful

**Parameters** **(SXDMFrameset)** (*self*) – the sxdmframset

**Returns**

- *the region of interst maps for each scan, the region of interest maps for the user defined*
- *sub regions of interest*

sxdm.mis.**delimiter_func**(*string*)
The delimiter function used in the image importer Used to just get the number of the image from the name of the image/mda file

**Parameters** **(str)** (*string*) – the string the user wants to place the delimiter function on

**Returns**

**Return type** a cropped string based on the delimiter function

sxdm.mis.**dic2array**(*dic*)
Turn a dictionary into a numpy array

**Parameters** **(dictionary)** (*dic*) – a dictionary of form {0: (num1, num2), 1: (num3, num4)}

**Returns**

**Return type** a numpy array of form [(num1, num2), (num3, num4)]

sxdm.mis.**figure_size_finder**(*images*)
Determines the amount of boxes to place in the alignment viewer

**Parameters** **(nd.array)** (*images*) – the numpy array

**Returns**

**Return type** the amount of images to make axes for in plt.subplot((return, return))

sxdm.mis.**get_idx4roi**(*pix*, *destination*, *scan_numbers*)
Based on the pixel being loaded, this returns the index of each scan used in the self.scan_numbers

**Parameters**

- **(array of strings)** (*scan_numbers*) – the image numbers for each scan for a given pixel
- **(array of strings)** – full diffraction image locations for each scan for a given pixel
- **(array of strings)** – the self.scan_numbers value

**Returns**

**Return type** an array of index values for a master roi used to store values in the correct position

sxdm.mis.**grab_dxdy**(*self*)
Return the dxdy movements for each scan

**Parameters** **(SXDMFrameset)** (*self*) – the sxdmframeset

**Returns**

**Return type** the dxdy values stored in the self.file

sxdm.mis.**grab_fov_dimensions**(*self*)
Returns the image dimensions for the User

**Parameters** **(SXDMFrameset)** (*self*) – the sxdmframeset

**Returns**

**Return type** the np.shape() of the fluorescence images - which are identical for the the entire field of view

sxdm.mis.**image_numbers**(*self*)
Grab the image numbers

**Parameters** **(SXDMFrameset)** (*self*) – the sxdmframset

**Returns**

**Return type** all of the diffraction image numbers for each scan in a 3D matrix

sxdm.mis.**load_variable_pickle**(*file*)

sxdm.mis.**median_blur_numpy**(*input_array*, *median_blur_distance*, *cut_off_value_above_mean*, *with_low=False*)
Median Blur a 1D array. Used for eliminating hot or dead pixels

**Parameters**

- **(1D array)** (*input_array*) – a one dimensional numpy array
- **(int)** (*median_blur_distance*) – the chunk size of numbers to to check for a median blur corrections

**Returns**

**Return type** a 1 dimensional numpy array that has been median blurred

sxdm.mis.**median_blur_selective**(*input_array*, *median_blur_distance*, *cut_off_value_above_mean*, *with_low=False*)

Allows for the user to have a selective median blur for individual spots. Meaning it will not assign a median blur to the entire spectra.

**Parameters**

- **(nd.array)** (*input_array*) – the 1D spectral array
- **(int)** (*cut_off_value_above_mean*) – a value that assigns how large of a median blur the User wants to walk through.
- **(int)** – a value above the mean value for the med_blur_distance in which the value gets attributed as the median of that data chunk
- **(bool)** (*with_low*) – True median blurs on values way above or way below the mean. Rather than just way above.

**Returns**

**Return type** An 1D spectral array with appropriate values median blurred

sxdm.mis.**order_dir**(*path*)

For a selected path return the ordered filenames. Meant for ordering images inside of a folder

**Parameters** **(str)** (*path*) – the path to the folder you would like to order the contents of

**Returns**

**Return type** the full image location, image name

sxdm.mis.**ram_check**()

Check how much RAM is being used. If it's over 90% then the analysis function stop loading information

**Returns**

**Return type** the percent of RAM usage

sxdm.mis.**resolution_check**(*self*, *user_resolution_um=0.0005*)

Check if all X dimensions for all scans are equal to each other. Also checks in all y dimensions for all scans are equal to each other. Then checks if the x and y are equal to one another.

**Parameters**

- **(SXDMFramset)** (*self*) – the sxdmframset
- **(float)** (*user_resolution_um*) – the resolution in um the user would like the scan dimensions for

**Returns**

**Return type** Nothing - sets resolution values

sxdm.mis.**results_2dsum**(*self*)

Returns the 2d summed diffraction pattern from the results self.saved_file without loading all the diffraction data. This works on the self.results variable.

Problems - Uses a lot of RAM

**Parameters** **(SXDMFrameset)** (*self*) – the sxdmframeset object

**Returns**

**Return type** 2d image array of the summed diffraction pattern

sxdm.mis.**save_variable_pickle**(*array_of_vars*, *file*)

sxdm.mis.**scan_num_convert**(*scan_numbers*, *fill_num=4*)
Convert the users int scan numbers into strings

**Parameters**

- **(str or int)** (*scan_numbers*) – takes an array of str values or int values and zfills them accordingly
- **(int)** (*fill_num*) – the string fill number the user would like to use

**Returns**

**Return type** the scan numbers in an array of strings with the user selected fill number

sxdm.mis.**set_centering**(*self*)
Set the centering values for each scan

**Parameters** **(SXDMFrameset)** (*self*) – the sxdmframeset

**Returns**

**Return type** The centering index - sets all centering data

sxdm.mis.**shape_check**(*self*)
Check to see if the imported hybrids have the same shape

**Parameters** **(SXDMFrameset)** (*self*) – the sxdmframeset used

**Returns**

**Return type** Nothing. Prints Important Data

sxdm.mis.**show_hybrid_dimensions**(*self*)
Shows the hybrid x and hybrid y image dimensions

**Parameters** **(SXDMFrameset)** (*self*) – the sxdmframeset object

**Returns**

**Return type** prints the hybrid x an y dimensions

sxdm.mis.**tif_separation**(*string*, *func=<function delimiter_func>*)
Separating the image number from the full name stored in the images folder through the user of the delimiter_func

**Parameters**

- **(str)** (*string*) – the string the user wants to place the delimiter function on
- **(function)** (*func*) – the delimiter function for the string. Makes it easier to change in later versions

**Returns**

**Return type** the image string name of just the number

sxdm.mis.**total_rows_int_tup**(*input*, *difference=False*)
Allows the user to return the true starting and ending values for a scan. Useful for when scans do not start at zero.

**Parameters**

- **(int or tuple)** (*input*) – the int or tuple input of the analysis functions
- **(bool)** (*difference*) – if the User would like to take the difference between the starting and ending points

**Returns**

**Return type** Either the starting and ending values for a scan or the difference between them

sxdm.mis.**val_check**(*array*, *resolution*)
Check the resolution of the x and y dimensions of the scans

**Parameters**

- **(nd.array)** (*array*) – array of all the scan x or y dimensions
- **(float)** (*resolution*) – a numbers in microns that sets the resolution check of the x and y dimensions

**Returns**

**Return type** bool (True or False) on whether or not the scan dimensions are within the set resolution

sxdm.mis.**zfill_scan**(*scan_list*, *fill_num*)
Change int to str and fill them with the user set number of zeros

**Parameters**

- **(str)** (*scan_list*) – a list string of the scans the user would like to import
- **(int)** (*fill_num*) – the amount of numbers each number should have. fill_num=4 turns '40' into '0040'

**Returns**

**Return type** A list of scan numbers with corrected zfill values

### 5.1.12 sxdm.multi module

sxdm.multi.**centroid_map_analysis**(*self*, *rows*, *columns*, *med_blur_distance=4*, *med_blur_height=10*, *stdev_min=35*, *multiplier=1*, *center_around=False*)
Calculates spot diffraction and data needed to make 2theta/chi/roi maps

**Parameters**

- **self** ((SXDMFramset)) – the sxdmframset
- **rows** ((int)) – the total number of rows you want to iterate through
- **columns** ((int)) – the total number of columns you want to iterate through
- **med_blur_distance** ((int)) – the amount of values to scan for median blur
- **med_blur_height** – (int) the height cut off for the median blur
- **stdev_min** ((int)) – standard deviation above the mean of signal to ignore
- **multiplier** ((int)) – multiplier for the background signal to be subtracted
- **center_around** ((int)) – the index of the scan you would like to center around

**Returns**

- *A numpy matrix of every pixel asked. Each pixel contains*
- *row_column,(row, column), summed_dif, ttheta, chi,*
- *ttheta_centroid_finder, ttheta_centroid,*
- *chi_centroid_finder, chi_centroid, full_roi*

sxdm.multi.**create_imagearray**(*self*, *center_around=False*)
Creates the self.image_array variable needed for pixel_analysis

**Parameters**

- **(SXDMFrameset)** (*self*) – the sxdmframset
- **(bool)** (*center_around*) – setting this to True allows the user to default to the first image index to center around

**Returns**

**Return type** Nothing - sets the self.image_array value

sxdm.multi.**initialize_vectorize**(*num_rows*, *num_columns*)

**Places the row and column number next to each other in an iterable array.** Used for starmap multiprocessing pixels.

**Parameters**

- **num_rows** ((int)) – Total number of rows the user wants to calculate
- **num_columns** ((int)) – Total number of columns the user wants to calculate

**Returns**

**Return type** An iterable array of pixel locations to be passed into the multiprocessing pixel analysis

sxdm.multi.**iterations**(*self*, *num_rows*, *num_columns*)

**Places the row and column number next to each other in an iterable array.** Used for starmap multiprocessing pixels.

**Parameters**

- **self** ((SXDMFrameset)) –
- **num_rows** ((int)) – Total number of rows the user wants to calculate
- **num_columns** ((int)) – Total number of columns the user wants to calculate

**Returns**

**Return type** An iterable array of pixel locations to be passed into the multiprocessing pixel analysis

sxdm.multi.**pooled_return**(*results*, *user_val*)

Makes it easy to return values from the pooled results from the multi.analysis function

**Parameters**

- **(n dimensional array)** (*results*) – the output from the analysis function
- **(str)** (*user_val*) – a string that defines what the user wants to be returned

**Returns**

**Return type** An n dimensional array consisting of the user selected data output from the multi.analysis function

sxdm.multi.**roi_analysis**(*self*, *rows*, *columns*, *med_blur_distance=4*, *med_blur_height=10*, *stdev_min=35*, *multiplier=1*, *center_around=False*, *diff_segmentation=True*)

Calculates region of interest for each scan as well as the region of interest maps for user defined sub region of interests

**Parameters**

- **self** ((SXDMFramset)) – the sxdmframset

- **rows** ((int)) – the total number of rows you want to iterate through
- **columns** ((int)) – the total number of columns you want to iterate through
- **med_blur_distance** ((int)) – the amount of values to scan for median blur
- **med_blur_height** – (int) the height cut off for the median blur
- **stdev_min** ((int)) – standard deviation above the mean of signal to ignore
- **multiplier** ((int)) – multiplier for the background signal to be subtracted
- **center_around** ((int)) – the index of the scan you would like to center around
- **diff_segmentation** ((bool)) – if set to True this will initiate the user sub region of interest analysis

**Returns**

- *A numpy matrix of every pixel asked. Each pixel contains*
- *[(row, column), idxs,* – raw_scan_data, corr_scan_data, scan_data_roi_vals, summed_data, corr_summed_data, summed_data_roi_vals]

### 5.1.13 sxdm.multi_update module

sxdm.multi_update.**centroid_analysis_multi**(*self*, *rows*, *columns*, *med_blur_distance=9*, *med_blur_height=100*, *stdev=35*, *bkg_multiplier=0*)

Runs the centroid analysis in a pool.map function

**Parameters**

- **self** ((SXDMFrameset)) – the sxdmframeset object
- **rows** ((int or tup)) – the total amount of rows the User wants to analyze or a tuple of the rows
- **columns** ((int or tup)) – the total amount of columns the User wants to analyze or a tuple of the columns
- **med_blur_distance** ((int)) – the median blur distance - must be odd
- **med_blur_height** ((int)) – the median blur height
- **stdev** ((int)) – the standard deviation used to segment data
- **bkg_multiplier** ((int)) – the multiplier to the background images

**Returns**

**Return type** a pooled results from the centroid_pixel_analysis_multi() function

sxdm.multi_update.**centroid_pixel_analysis_multi**(*row*, *column*, *median_blur_distance*, *median_blur_height*, *stdev_min*, *image_array*, *scan_numbers*, *background_dic*, *file*)

The analysis done on a single pixel :param row: the row the User wants to do analysis on :type row: (int) :param column: the column the User wants to do analysis on :type column: (int) :param median_blur_distance: the amount of values to scan for median blur :type median_blur_distance: (int) :param median_blur_height: the height cut off for the median blur :type median_blur_height: (int) :param stdev_min: standard deviation above the mean of signal to ignore :type stdev_min: (int) :param image_array: the image location array - can be created with create_imagearray(self) :type image_array: (nd.array) :param scan_numbers: the list of scan numbers used :type scan_numbers: (nd.array) :param background_dic: the

background scan dictionary entry - can be made with scan_backgrounnd(self) :type background_dic: (dic) :param file: the hdf5 file location :type file: (str)

**Returns**

**Return type** the analysis results as an nd.array

sxdm.multi_update.**centroid_pre_analysis**(*inputs*, *meds_d*, *meds_h*, *st*, *image_array*, *scan_numbers*, *background_dic*, *file*)

Allows Python to run the centroid analysis in a pool.map function

**Parameters**

- **inputs** ((nd.array)) – the iterable inputs of rows and columns
- **meds_d** ((int)) – the median blur distance value
- **meds_h** ((int)) – the median_blur height value
- **image_array** ((nd.array)) – the total locations of all the images - created by create_imagearray(self)
- **scan_numbers** ((nd.array)) – the list of all the scan numbers
- **background_dic** ((dic)) – the dictionary of all the background images - created by scan_background(self)
- **file** ((str)) – the full hdf5 file location
- **diff_segments** ((bool)) – if True this will run the segmentation analysis as well

**Returns**

**Return type** the results from the centroid_pixel_analysis_multi() function

sxdm.multi_update.**h5get_image_destination_multi**(*scan_numbers*, *pixel*)

Determine where all the .tif images are for all the scan numbers disregarding the nan values

**Parameters**

- **(nd.array)** (*scan_numbers*) – the scan numbers the user wants to get
- **(str array)** (*pixel*) – the image number from each scan that corresponds to a certain pixel

**Returns**

**Return type** All of the diffraction FULL image locations for each scan in a 3D array - excluding the np.nan's

sxdm.multi_update.**roi_analysis_multi**(*self*, *rows*, *columns*, *med_blur_distance=9*, *med_blur_height=100*, *bkg_multiplier=0*, *diff_segments=True*)

Runs the region of interest analysis in a pool.map function

**Parameters**

- **self** ((SXDMFrameset)) – the sxdmframeset object
- **rows** ((int or tup)) – the total amount of rows the User wants to analyze or a tuple of the rows
- **columns** ((int or tup)) – the total amount of columns the User wants to analyze or a tuple of the columns
- **med_blur_distance** ((int)) – the median blur distance - must be odd
- **med_blur_height** ((int)) – the median blur height

- **bkg_multiplier** ((int)) – the multiplier to the background images
- **diff_segments** ((bool)) – if True this will analyze the roi segmentations

**Returns**

**Return type** a pooled results from the roi_pixel_analysis_multi() function

sxdm.multi_update.**roi_pixel_analysis_multi**(*row*, *column*, *median_blur_distance*, *median_blur_height*, *image_array*, *scan_numbers*, *background_dic*, *file*, *diff_segments=False*)

The analysis done on a single pixel - this will get the new roi for each theta and be able to segment out diffraction areas and create the roi for them :param row: the total number of rows you want to iterate through :type row: (int) :param column: the total number of columns you want to iterate through :type column: (int) :param median_blur_distance: the amount of values to scan for median blur :type median_blur_distance: (int) :param median_blur_height: the height cut off for the median blur :type median_blur_height: (int) :param image_array: the location for all the pixels - can be created through create_imagearray(self) :type image_array: (nd.array) :param scan_numbers: the list of scan numbers :type scan_numbers: (array) :param background_dic: the dictionary for the background images - created through scan_background(self) :type background_dic: (dic) :param file: the location to the hdf5 file :type file: (str) :param diff_segments: array used for segmenting the diffraction patterns :type diff_segments: (array)

**Returns**

- *the analysis results as an nd.array*
- *[(row, column), idxs,* – raw_scan_data, corr_scan_data, scan_data_roi_vals, summed_data, corr_summed_data, summed_data_roi_vals]

sxdm.multi_update.**roi_pre_analysis**(*inputs*, *meds_d*, *meds_h*, *image_array*, *scan_numbers*, *background_dic*, *file*, *diff_segments*)

Allows Python to run the roi analysis in a pool.map function

**Parameters**

- **inputs** ((nd.array)) – the iterable inputs of rows and columns
- **meds_d** ((int)) – the median blur distance value
- **meds_h** ((int)) – the median_blur height value
- **image_array** ((nd.array)) – the total locations of all the images - created by create_imagearray(self)
- **scan_numbers** ((nd.array)) – the list of all the scan numbers
- **background_dic** ((dic)) – the dictionary of all the background images - created by scan_background(self)
- **file** ((str)) – the full hdf5 file location
- **diff_segments** ((bool)) – if True this will run the segmentation analysis as well

**Returns**

**Return type** the results from the roi_pixel_analysis_multi() function

sxdm.multi_update.**sum_pixel_multi**(*file*, *images_loc*)

Sum a pixel :param file (str): the full file location of the hdf5 file :param image_loc (list of str): the full image location in the hdf5 file

**Returns**

**Return type** all images set in the image_loc variable

### 5.1.14 sxdm.pixel module

sxdm.pixel.**centroid_finder**(*oneDarray_start*, *stdev_min=35*)

Determine the centroid function :param oneDarray_start (numpy array): a one dimensions numpy array :param stdev_min (int): the standard deviation minimum the user would like to section the data off with

**Returns**

**Return type** the corrected one dimensional array and the centroid of the array

sxdm.pixel.**centroid_pixel_analysis**(*self*, *row*, *column*, *median_blur_distance*, *median_blur_height*, *stdev_min*)

The analysis done on a single pixel :param self (SXDMFrameset): the sxdmframset :param rows: the total number of rows you want to iterate through :type rows: `(int)` :param columns: the total number of columns you want to iterate through :type columns: `(int)` :param med_blur_distance: the amount of values to scan for median blur :type med_blur_distance: `(int)` :param med_blur_height: the height cut off for the median blur :type med_blur_height: `(int)` :param stdev_min: standard deviation above the mean of signal to ignore :type stdev_min: `(int)` :param bkg_multiplier: multiplier for the background signal to be subtracted :type bkg_multiplier: `(int)`

**Returns**

- *the analysis results as an nd.array*
- *# For a given row and column*
- *[*
- *(row_index, column_index),*
- *summed diffraction pattern (set to zero to save RAM. User can change in source code in /pixel.py),*
- *two theta centroid value (float)*
- *chi centroid value (float)*
- *two theta cropped signal (nd.array)*
- *full two theta signal (nd.array)*
- *chi cropped signal (nd.array)*
- *full chi signal (nd.array)*
- *summed region of interest value (float)*
- *]*

sxdm.pixel.**chi_maths**(*summed_dif*, *median_blur_distance*, *median_blur_height*, *stdev_min*, *q=False*)

Determine the centroid of the chi axis :param summed_dif (nd.array image): a 2D summed diffraction image :param median_blur_distance (int): the amount of values to take the median of :param median_blur_height (int): the value above the median to replace with the median value :param stdev_min (int): the standard deviation above the noise to consider the signal valid :param q (bool): used for an old multi processing function - unused now - might be put in at a later date

**Returns**

- *the edited (median blured) sum down the x axis*
- *the centroid of the data*
- *the cropped data to find the centroid*
- *the edited (median blured) sum down the x axis as a numpy array*

sxdm.pixel.**grab_pix**(*array*, *row*, *column*, *int_convert=False*)
Return a pixel at a given row and column value :param array (nd.array): a 3 dimensional numpy array :param row (int): the row the user would like to grab :param column (int): the column the user would like to grab :param int_convert (bool): if the user would like to change np.nans to integers set this to True

**Returns**

**Return type** All data associated with the set row and column for the 3 dimensional array

sxdm.pixel.**pixel_diffraction_grab**(*self*, *image_array*, *row*, *column*)
Returns the diffraction image for a specific scan location (row, column)

**Parameters**

- **self** ((SXDMFrameset object)) – the sxdmframset object
- **image_array** ((nd.array)) – the array that holds all the diffraction image locations self.im_array()
- **row** ((int)) – the row the User would like to return data for
- **column** ((int)) – the column the User would like to return data for

**Returns**

**Return type** Summed diffraction image for a single pixel

sxdm.pixel.**roi_pixel_analysis**(*self*, *row*, *column*, *median_blur_distance*, *median_blur_height*, *diff_segments=False*)
The analysis done on a single pixel - this will get the new roi for each theta and be able to segment out diffraction areas and create the roi for them :param self (SXDMFrameset): the sxdmframset :param rows: the total number of rows you want to iterate through :type rows: (int) :param columns: the total number of columns you want to iterate through :type columns: (int) :param med_blur_distance: the amount of values to scan for median blur :type med_blur_distance: (int) :param med_blur_height: the height cut off for the median blur :type med_blur_height: (int) :param diff_segments: array used for segmenting the diffraction patterns :type diff_segments: (array)

**Returns**

- *the analysis results as an nd.array*
- *[(row, column), idxs,* – raw_scan_data, corr_scan_data, scan_data_roi_vals, summed_data, corr_summed_data, summed_data_roi_vals]

sxdm.pixel.**segment_diffraction_roi**(*diffraction*)
Creates a roi value and the raw data analysis for the ROI maps

**Parameters** **(image)** (*diffraction*) – the summed diffraction image

**Returns**

**Return type** ttheta, ttheta_copy, np.sum(ttheta_copy), scan_roi_val

sxdm.pixel.**sum_pixel**(*self*, *images_loc*)
Sum a pixel :param self (SXDMFrameset): the sxdmframset :param image_loc (list of str): the full image location in the hdf5 file

**Returns**

**Return type** all images set in the image_loc variable

sxdm.pixel.**sum_pixel_v2**(*images_loc*, *file*)
Sum a pixel :param image_loc (list of str): the full image location in the hdf5 file :param file: the opened hdf file :type file: (hdf file)

**Returns**

**Return type** all images set in the image_loc variable

sxdm.pixel.**theta_maths**(*summed_dif*, *median_blur_distance*, *median_blur_height*, *stdev_min*, *q=False*)

Determine the centroid of the theta axis :param summed_dif (nd.array image): a 2D summed diffraction image :param median_blur_distance (int): the amount of values to take the median of :param median_blur_height (int): the value above the median to replace with the median value :param stdev_min (int): the standard deviation above the noise to consider the signal valid :param q (bool): used for an old multi processing function - unused now - might be put in at a later date

**Returns**

- *the edited (median blured) sum down the y axis*
- *the centroid of the data*
- *the cropped data to find the centroid*
- *the edited (median blured) sum down the y axis as a numpy array*

### 5.1.15 sxdm.postprocess module

sxdm.postprocess.**centroid_roi_map**(*results*, *map_type='chi_centroid'*)

Returns the centroid or roi based on the Centroid Analysis :param results (nd.array): self.results or the output of self.analysis :param map_type (str): the value the user would like to return acceptable values are chi_centroid, ttheta_centroid, or full_roi

**Returns**

**Return type** the user selected map in an nd.array

sxdm.postprocess.**maps_correct**(*user_map*, *new_bounds*)

Takes the centroid_rou_map() function output and gives it new bounds :param user_map (nd.array): the output of the centroid_roi_map function :param new_bounds (np.linspace): np.linspace(lowerbound, higherbound, dim of image)

**Returns**

- *nd.array of the user_map, but with new bounds rather than with the dimensions*
- *of the diffraction image*

sxdm.postprocess.**pixel_analysis_return**(*results*, *row*, *column*, *show_accep_vals=False*)

Easy return of the results based on the input row and column value :param results (nd.array): self.results or output of self.analysis :param row (int): the row the user would like to look at :param column (int): the column the user would like to look at

**Returns**

- *A dictionary with entries*
- *'row_column',*
- *'summed_dif',*
- *'ttheta',*
- *'chi',*
- *'ttheta_corr',*
- *'chi_corr',*
- *'ttheta_cent',*

- *'chi_cent',*
- *'roi'*

sxdm.postprocess.**saved_return**(*file*, *group*, *summed_dif_return=False*)
Load saved data :param file (str): a user defined hdf5 file :param group (str): the group the user would like to import :param summed_dif_return (bool): if True this will import all data. it is set to False because this import take up a lot of RAM

**Returns**

**Return type** a nd.array thay can be set to the self.results value

sxdm.postprocess.**twodsummed**(*results*)
Returns the summed diffraction pattern for the analysis output :param results (nd.array): the self.results or output of self.analysis

**Returns**

**Return type** the summed diffraction patter from the analysis output

### 5.1.16 sxdm.preprocess module

sxdm.preprocess.**gaus_check**(*self*, *center_around=False*, *default=False*)
Determines the intensity change over all group scans

**Parameters** **(SXDMFrameset)** (*self*) – the sxdmframset

**Returns**

- *the x and y values for a plot that shows intensity vs scan angle.*
- *also auto displays the figure*

sxdm.preprocess.**init_dxdy**(*self*)
Initializes a group that scan store dxdy data

**Parameters** **(SXDMFrameset)** (*self*) – the sxdmframeset

**Returns**

**Return type** Nothing - creates a blank self.dxdy_store variable

sxdm.preprocess.**initialize_experimental_attrs**(*self*)
Initialized the Kev and detector's theta position.

Sets attributes of Kev and detector theta position to the current user defined group

**Parameters** **(SXDMFrameset)** (*self*) – the sxdmframeset

**Returns**

**Return type** Nothing - sets Kev and detector_theta attributes

sxdm.preprocess.**initialize_group**(*self*)
Initialized the group assigned by the user to the user defined file.

From the scan numbers the program determines the scan_theta as well checking if identical settings have already been imported. If so the program reloads saved settings

**Parameters** **(SXDMFrameset)** (*self*) – the sxdmframeset

**Returns**

**Return type** Nothing - sets /scan_numbers and /scan_theta

sxdm.preprocess.**initialize_saving**(*self*)
Initialize the ability to save data

**Parameters** **(SXDMFrameset)** (*self*) – the sxdmframset

**Returns**

**Return type** Nothing - creates the filename_savedata.h5 file to save data to

sxdm.preprocess.**initialize_scans**(*self*, *scan_numbers=False*, *fill_num=4*)
Initialize all necessities for each scan

**Parameters**

- **(SXDMFrameset)** (*self*) – the sxdmframeset
- **(nd.array of int)** (*scan_numbers*) – the list of scan numbers the user wants to create [178, 178, . . . ]
- **(int)** (*fill_num*) – the zfill number for the scan number integers

**Returns**

**Return type** Nothing - sets self.scan_numbers, self.det_smpl_theta, and self.scan_theta

sxdm.preprocess.**initialize_zoneplate_data**(*self*, *reset=False*)
Initialize values for the zoneplate used as well as detector pixel size.

Asks the user what the parameters of the zone plate are and detector pixel size. If they have been already set then the program displays the current values when setting up SXDMFrameset object

**Parameters**

- **(SXDMFrameset)** (*self*) – the sxdmframset
- **(bool)** (*reset*) – if True this allows the user to reset all values

**Returns**

**Return type** Nothing - sets values inside /zone_plate/

sxdm.preprocess.**max_det_val**(*self*, *detector='fluor'*)
Used to determine the max values for a given detector channel

**Parameters**

- **(SXDMFrameset)** (*self*) – the sxdmframeset
- **(str)** (*detector*) – value to be passed into the return_det() function

**Returns**

**Return type** the max values for all scans for a given detector channel input

### 5.1.17 sxdm.roi module

sxdm.roi.**add_rois**(*rois*, *types='scan'*)
Add the create_rois output and return the correct plots

**Parameters**

- **rois** ((nd.array)) – the rois the user would like to add together
- **types** ((str)) – either ‘scan’ or ‘bounding’ - depends on what roi’s the User sets for the rois variable

**Returns**

**Return type** 1d.array of the roi's summed

sxdm.roi.**bounding_box_setup**(*figure_class*)
Set up the viewer figure for what bounding boxes have been selected

**Parameters** **(ROI_FigureClass)** (*figure_class*) – the roi_figureclass object

**Returns**

**Return type** Nothing - sets figure_class.axes

sxdm.roi.**bounding_box_total_setup**(*user_class*)
Complete setup of the summed diffraction pattern with the bounding boxes on top of them

**Parameters** **user_class** ((SXDMFrameset)) – the sxdmframeset object

**Returns**

**Return type** Nothing - sets up all the bounding box figures and loads all items

sxdm.roi.**bounding_roi_figure_setup**(*figure_class*)
Initiates the bounding box figure with axes

**Parameters** **(ROI_FiguresClass)** (*figure_class*) – the roi_figuresclass object

**Returns**

**Return type** Nothing

sxdm.roi.**bounding_slider_setup**(*figure_class*, *user_class*)
Sets up the bounding figure slider inside the roi figures

**Parameters**

- **(ROI_FiguresClass)** (*figure_class*) – the roi_figuresclass object
- **(SXDMFrameset)** (*user_class*) – the sxdmframeset object

**Returns**

**Return type** Nothing

sxdm.roi.**bounding_tb_setup**(*figure_class*)
Sets up all the textboxes and buttons

**Parameters** **(ROI_FigureClass)** (*figure_class*) – the roi_figureclass object

**Returns**

**Return type** Nothing

sxdm.roi.**close_all**(*event*)

sxdm.roi.**create_bounding_box_rois**(*fs*)

sxdm.roi.**create_total_linescan**(*fs*)

sxdm.roi.**display_left_roi**(*figure_class*, *user_class*, *types='scan'*)
Sets up the display for the left roi plot

**Parameters**

- **figure_class** ((ROI_FigureClass)) – the roi_figureclass object
- **user_class** ((SXDMFrameset)) – the sxdmframeset object
- **types** ((str)) – either 'scan' or 'bounding'

**Returns**

**Return type** Nothing - displays the left roi figure

sxdm.roi.**display_left_roi_reload**(*val*, *figure_class*, *user_class*, *types='scan'*)
Reloads the left roi

**Parameters**

- **val** ((matplotlib event)) – the val matplotlib event
- **figure_class** ((ROI_FigureClass)) – the roi_figureclass object
- **user_class** ((SXDMFrameset)) – the sxdmframeset object
- **types** ((str)) – either 'scan' or 'bounding'

**Returns**

**Return type** Nothing - reloads the left image

sxdm.roi.**display_right_roi**(*figure_class*, *im*)
Displays the right roi plot

**Parameters**

- **figure_class** ((ROI_FigureClass)) – the roi_figureclass object
- **im** ((nd.array)) – the image array to be displayed

**Returns**

**Return type** Nothing - displays the right roi image

sxdm.roi.**display_right_roi_reload**(*val*, *figure_class*, *im*)
Load the Right ROI data

**Parameters**

- **val** ((matplotlib event)) – the val matplotlib event
- **figure_class** ((ROI_FigureClass)) – the roi_figureclass object
- **im** ((nd.array)) – the image to be displayed

**Returns**

**Return type** Nothing - reloads the right figure data

sxdm.roi.**display_summed_ims**(*figure_class*, *user_class*)
Show the summed diffraction pattern with all assigned bounding boxes

**Parameters**

- **figure_class** ((ROI_FigureClass)) – the roi_figureclass object
- **user_class** ((SXDMFrameset)) – the sxdmframeset object

**Returns**

**Return type** Nothing - displays the summed diffraction pattern along with the User selected bounding boxes

sxdm.roi.**display_summed_ims_reload**(*val*, *figure_class*, *user_class*)
Allows the summed diffraction images to be displayed

**Parameters**

- **val** ((matplotlib val)) – matplotlib variable
- **figure_class** ((ROI_FigureClass)) – the roifigureclass object

- **user_class** ((SXDMFrameset)) – the sxdmframeset object

**Returns**

**Return type** Nothing - reloads the summed diffraction images in the figure

sxdm.roi.**grab_int_v_scan**(*user_class*, *types='scan'*)
Grab the data needed for the gaus plot

**Parameters**

- **user_class** ((SXDMFrameset)) – the sxdmframeset object
- **types** ((str)) – either 'scan' or 'bounding' - determines which plots the User would like to get information for

**Returns**

**Return type** Nothing - sets up the x and y values for the top plot of either the Scan ROI or the Bounding Box ROI

sxdm.roi.**initiate_roi_viewer**(*user_class*)
Sets up all the ROI viewer figures

**Parameters** **(SXDMFrameset)** (*user_class*) – the sxdmframeset object

**Returns**

**Return type** Nothing

sxdm.roi.**on_scan_click**(*event*, *figure_class*, *user_class*, *types='scan'*)
Obtains all the raw scan data for a user clicked pixel

**Parameters**

- **(matplotlib event)** (*event*) – the matplotlib event
- **(ROI_FiguresClass)** (*figure_class*) – the roi_figuresclass object
- **(SXDMFrameset)** (*user_class*) – the sxdmframeset object
- **(str)** (*types*) – either 'scan' or 'bounding'

**Returns**

**Return type** Nothing - obtains all data and diplays data to appropriate location

sxdm.roi.**right_roi**(*user_class*, *types='scan'*)
Plots the right roi plot

**Parameters**

- **user_class** ((SXDMFrameset)) – the sxdmframeset object
- **types** ((str)) – either 'scan' or 'bounding' - depends on which figure is being called

**Returns**

**Return type** The entire summed region of interest for either the scans or the bounding boxes

sxdm.roi.**scan_roi_figure_setup**(*figure_class*)
Initiates the scan roi figure with proper axes locations current_roi_slider_val

**Parameters** **(ROI_FiguresClass)** (*figure_class*) – the roi_figuresclass object

**Returns**

**Return type** Nothing

sxdm.roi.**scan_slider_setup**(*figure_class*, *user_class*)
Sets up the scan figure slider inside the roi figures

**Parameters**

- **(ROI_FiguresClass)** (*figure_class*) – the roi_figuresclass object
- **(SXDMFrameset)** (*user_class*) – the sxdmframeset object

**Returns**

**Return type** Nothing

sxdm.roi.**start_bounding_roi**(*user_class*)
Displays the bounding box roi with aproptiate textboxes

**Parameters** **(SXDMFrameset)** (*user_class*) – the sxdmframeset object

**Returns**

**Return type** Nothing

sxdm.roi.**start_scan_roi**(*user_class*)
Displays the scan roi figure with appropriate settings

**Parameters** **(SXDMFrameset)** (*user_class*) – the sxdmframeset object

**Returns**

**Return type** Nothing

sxdm.roi.**textbox_setup**(*figure_class*)
Sets up textboxes for the scan roi and the bounding box roi figures

**Parameters** **(ROI_FiguresClass)** (*figure_class*) – the roi_figuresclass object

**Returns**

**Return type** Nothing

sxdm.roi.**top_plot_reload**(*val*, *figure_class*, *user_class*, *types='scan'*)
Reloads the gaus plot

**Parameters**

- **val** ((matplotlib val)) – the value for matplot lib events
- **figure_class** ((ROIFigure_Class)) – the roifigure_class objects
- **user_class** ((SXDMFrameset)) – the sxdmframeset object
- **types** ((str)) – either ‘scan’ or ‘bounding’ - determines which plots need to be reloaded

**Returns**

**Return type** Nothing - reloads the top plot in the roi figures

sxdm.roi.**top_plot_start**(*figure_class*, *user_class*, *types='scan'*)
Initiate the gaus plot

**Parameters**

- **figure_class** ((ROI_FigureClass)) – the roi_figureclass object
- **user_class** ((SXDMFrameset)) – the sxdmframeset object
- **types** ((str)) – either ‘scan’ or ‘bounding’ allow the User to choose which figure they are working on

**Returns**

**Return type** Nothing

### 5.1.18 sxdm.roi_bounding module

**class** sxdm.roi_bounding.**ROI_FiguresClass**
Bases: object

A class function which will make it easier to move variable in and out of functions

sxdm.roi_bounding.**bounding_figure_setup**(*figure_class*)
Set up the viewer figure

**Parameters** **(ROI_FigureClass)** (*figure_class*) – the roi_figureclass object

**Returns**

**Return type** Nothing - sets figure_class.axes

sxdm.roi_bounding.**contbtn_click**(*event*, *figure_class*, *user_class*)
Close the bounding box figure once complete

**Parameters**

- **(matplotlib event)** (*event*) – the matplotlib event
- **(matplotlib figure)** (*fig*) – the matplotlib figure to close

**Returns**

**Return type** Nothing

sxdm.roi_bounding.**line_select_callback**(*eclick*, *erelease*, *figure_class*)
Creates interactive bounding boxes for roi analysis

**Parameters**

- **(event)** (*erelease*) – the click event
- **(event)** – the release event
- **(ROI_FigureClass)** (*figure_class*) – the roi_figureclass object

**Returns**

**Return type** Nothing

sxdm.roi_bounding.**rec_select_setup**(*figure_class*)
Initiate the rectangle interactive

**Parameters** **(ROI_FigureClass)** (*figure_class*) – the roi_figureclass object

**Returns**

**Return type** Nothing

sxdm.roi_bounding.**rm_box_click**(*event*, *figure_class*)
Removed the last bounding box the user made

**Parameters**

- **(matplotlib event)** (*event*) – the matploblib event
- **(ROI_FigureClass)** (*figure_class*) – the roi_figureclass object

**Returns**

**Return type** Nothing

sxdm.roi_bounding.**start_bounding_box**(*summed_diff_pattern*, *user_class*)
Starts the ROI bounding box GUI

**Returns**

**Return type** Nothing

sxdm.roi_bounding.**textbox_btn_dropdown_setup**(*figure_class*)
Sets up all the textboxes and buttons

**Parameters** **(ROI_FigureClass)** (*figure_class*) – the roi_figureclass object

**Returns**

**Return type** Nothing

sxdm.roi_bounding.**vs_change**(*text*, *figure_class*)
Change the vmin and vmax of a plot

**Parameters**

- **(matplotlib textbox)** (*text*) – the matplotlib textbox
- **(ROI_FigureClass)** (*figure_class*) – the roi_figureclass object

**Returns**

**Return type** Nothing

### 5.1.19 sxdm.summed2d module

sxdm.summed2d.**summed2d_all_data**(*self*, *bkg_multiplier=0*)
Loads the summed diffraction patter without using a large amount of RAM

**Parameters**

- **(SXDMFrameset)** (*self*) – the sxdmframeset object
- **(int)** (*bkg_multiplier*) – the multiplier for the background images

**Returns**

**Return type** the 2D summed diffraction pattern

sxdm.summed2d.**summed2d_all_data_v2**(*self*, *bkg_multiplier=0*)
Loads the summed diffraction patter without using a large amount of RAM

**Parameters**

- **(SXDMFrameset)** (*self*) – the sxdmframeset object
- **(int)** (*bkg_multiplier*) – the multiplier for the background images

**Returns**

**Return type** the 2D summed diffraction pattern

### 5.1.20 sxdm.SXDM module

**class** sxdm.SXDM.**SXDMFrameset**(*file*, *dataset_name*, *scan_numbers=False*, *fill_num=4*, *restart_zoneplate=False*, *median_blur_algorithm='scipy'*)

Bases: object

**alignment**(*reset=False*)

Allows the user to align all scans based on Fluorescence Maps or ROI Maps

**Parameters** **(bool)** (*reset*) – if True this will clear all the alignment data. Use this when adding new scans to a group

**Returns**

**Return type** Nothing - sets alignment data

**centroid_analysis**(*rows*, *columns*, *med_blur_distance=9*, *med_blur_height=10*, *stdev_min=25*, *bkg_multiplier=1*, *slow=False*, *change_center=False*, *cores=0*)

Calculates spot diffraction and data needed to make 2theta/chi/roi maps

**Parameters**

- **rows** ((int, tup)) – the total number of rows you want to iterate through or a tuple of the rows to iterate through
- **columns** ((int, tup)) – the total number of columns you want to iterate through or a tuple of the columns to iterate through
- **med_blur_distance** ((int)) – the amount of values to scan for median blur
- **med_blur_height** ((int)) – the height cut off for the median blur
- **stdev_min** ((int)) – standard deviation above the mean of signal to ignore
- **bkg_multiplier** ((int)) – multiplier for the background signal to be subtracted

**Returns**

**Return type** the analysis results in the form of self.results

**centroid_viewer**(*default_extents=False*)

Allow the user to view the data in a convenient format

**Parameters** **(bool)** (*diffraction_load*) – if True this will load all necessary data for the diffraction will take up at least 2gb of RAM

**Returns**

**Return type** Nothing - displays viewer

**chi_determination**()

Determine the axis bounds for the diffraction images

**frame_shape**()

Returns the total imported scans shape (scan_number, rows, columns)

**Parameters** **(SXDMFrameset)** (*self*) –

**Returns**

**Return type** the frame set scan dimensions

**gaus_checker**(*center_around=False*, *default=False*)

Plots total diffraction intensity vs scan angle for a set ROI

**Parameters**

- **(bool)** (*default*) – if True this will default the centerind index to the first index (0)
- **(bool)** – if True this will default the roi to check the gaus of to the first user defined ROI

**Returns**

**Return type** Nothing - displays the total intensity vs scan theta rocking curve

**image_data_dimensions**()
Determine the CCD image dimensions

**Parameters** **– SXDMFrameset** (*self*) – the sxdmframeset

**Returns**

**Return type** the image dimensions of the CCD camera

**ims_array**(*amount2ave=3*, *multiplier=1*, *center_around=False*)
Creates the Scan Background and the Image Array

**Parameters**

- **– int** (*amount2ave*) – Take the first (amount2ave) to the last (amount2ave) pixels and use them as a background
- **– float** (*multiplier*) – A number to be applied to the background image

**Returns**

**Return type** Creates the background image and the image array used for many proceses

**region_of_interest**(*rows*, *columns*, *med_blur_distance=9*, *med_blur_height=100*, *bkg_multiplier=1*, *diff_segmentation=True*, *slow=False*, *cores=0*)
Create a region of interest map for each scan and center the region of interest maps. If the diff segmentation is True this will also create a region of interest map based on a user defined sub region of interests

**Parameters**

- **(int) or (tup)** (*columns*) – the total amount of rows the user wants to do analysis on
- **(int) or (tup)** – the total amount of columns the user wants to do analysis on
- **(int)** (*bkg_multiplier*) – the amount of values to scan for median blur
- **(int)** – the height cut off for the median blur
- **(int)** – multiplier for the background signal to be subtracted
- **(bool)** (*diff_segmentation*) – if set to True this will determine region of interests maps for sub roi's set by the user

**Returns**

**[(row, column), idxs,** raw_scan_data, corr_scan_data, scan_data_roi_vals, summed_data, corr_summed_data, summed_data_roi_vals]

**Return type** the roi results in self.roi_results

**reload_save**(*summed_dif_return=True*)
Allow the user to reload Centroid saved data

**Parameters** **(bool)** (*summed_dif_return*) – if True this will load all necessary data for the diffraction will take up at least 2gb of RAM

**Returns**

**Return type** Nothing

**roi_segmentation**(*bkg_multiplier=1*, *restart=False*)
Allows the User to create a summed diffraction pattern bounding box for region_of_interest analysis

**Parameters**

- **(int)** (*bkg_multiplier*) – the scaler applied to the background images
- **(bool)** (*restart*) – if True this clears all bounding boxes when loading the segmentation data

**Returns**

**Return type** Nothing

**roi_viewer**()
Takes the self.roi_results variable and displays it in a GUI format

**Parameters** **(SXDMFrameset)** (*self*) – the sxdmframeset object

**Returns**

**Return type** Nothing

**save**()
Save self.results (Centroid Data) to _savedata.h5

Unable to efficiently save ROI data

### 5.1.21 sxdm.viewer module

**class** sxdm.viewer.**FiguresClass**
Bases: object

A class function which will make it easier to move variable in and out of functions

sxdm.viewer.**analysis_change**(*text*, *self*)
When analysis values change, make the plots change as well to fit analysis parameters :param text (textbox text event): textbox text event :param self (SXDMFrameset): the sxdmframeset

**Returns**

**Return type** Nothing - loads figure data

sxdm.viewer.**btn_setup**(*reproocessbtn_ax*, *savingbtn_ax*)
Set up the reprocess and saving button

**Parameters**

- **reproocessbtn_ax** ((matplotlib axis)) – the reproocess button figure axis
- **savingbtn_ax** ((matplotlib axis)) – the saving button figure axis

**Returns**

**Return type** reprocess and saving button

sxdm.viewer.**convert_x_axis**(*intensity_values*, *extents*, *centroid_value*, *type='ttheta'*)

sxdm.viewer.**figure_setup**()
Set up the viewer figure

**Returns**

**Return type** All axes for the figure

sxdm.viewer.**image_array_bkg_check**(*self*)

Warns the User if they have not started the create_imagearray() or scan_background() functions

**Parameters** **self** –

**Returns**

sxdm.viewer.**load_dynamic_data**(*results*, *vmin_spot*, *vmax_spot*, *spot_dif_ax*, *ttheta_centroid_ax*, *chi_centroid_ax*, *med_blur_distance*, *med_blur_height*, *stdev_min*, *row*, *column*, *self*)

Load the dynamic data in the viewer based on the user inputs :param results (nd.array): the output of self.analysis or the self.results value :param vmin_spot (int): the vmin for the summed diffraction spot image :param vmax_spot (int): the vmax for the summed diffraction spot image :param spot_dif_ax (matplotlib axis): the figure axis for the diffraction spot image :param ttheta_centroid_ax (matplotlib axis): the figure axis for the 1D two theta plot :param chi_centroid_ax (matplotlib axis): the figure axis for the 1D chi plot :param med_blur_distance (int): the integer value used in the median_blur() function :param med_blur_height (int): the integer value used in the median_blur() function :param stdev_min (int): the standard deviation value used in the median_blur() segmentation function :param row (int): the row the user would like to load the data for :param column (int): the bolumn the user would like to load the data for :param self (SXDMFramset): the sxdmframset

**Returns**

**Return type** Nothing - loads figure data

sxdm.viewer.**load_static_data**(*results*, *vmin_sum*, *vmax_sum*, *fluor_ax*, *roi_ax*, *summed_dif_ax*, *ttheta_map_ax*, *chi_map_ax*, *fluor_image*, *user_class=False*)

Load all the static data for the viewer :param results (nd.array): the output of the self.analysis function or the self.results value :param vmin_sum (int): the vmin for the summed diffraction image :param vmax_sum (int): the vmax for the summed diffraction image :param fluor_ax (matplotlib axis): the figure axis for the fluorescence image :param roi_ax (matplotlib axis): the figure axis for the region of interest image :param summed_dif_ax (matplotlib axis): the figure axis for the summed diffraction image :param ttheta_map_ax (matplotlib axis): the figure axis for the 2theta centroid map :param chi_map_ax (matplotlib axis): the figure axis for the chi centroid map :param fluor_image (image array): the fluorescence image the user would like to display in the figure :param user_class (SXDMFrameset): the sxdmframeset object or False

**Returns**

**Return type** Nothing - displays figure images

sxdm.viewer.**make_black**(*self*)

Reset figures to black when everything is up to date :param self (SXDMFrameset): the sxdmframeset

**Returns**

**Return type** Nothing - loads figure formatting

sxdm.viewer.**make_pink**(*self*)

Show the user which values have not been saved by turning them pink :param self (SXDMFrameset): the sxdmframeset

**Returns**

**Return type** Nothing - loads figure formatting

sxdm.viewer.**make_red**(*self*)

Show the user which figures are not current by turning them red :param self (SXDMFrameset): the sxdmframeset

**Returns**

**Return type** Nothing - loads figure formatting

sxdm.viewer.**reload_some_static_data**(*results*, *roi_ax*, *ttheta_map_ax*, *chi_map_ax*)
Reloading some of the static data when you save/reload image :param results (nd.array): the output of the self.analysis function or the self.results value :param roi_ax (matplotlib axis): the figure axis for the region of interest map :param ttheta_map_ax (matplotlib axis): the figure axis for the 2theta centroid map :param chi_map_ax (matplotlib axis): the figure axis for the chi centroid map

**Returns**

**Return type** Nothing - displays figure images

sxdm.viewer.**reprocessbtn_click**(*event*, *user_class*, *figure_class*)
Calls the self.analysis function once the reprocessed button is clicked :param event (matplotlib event): matplotlib event :param user_class (SXDMFrameset): the sxdmframeset :param figure_class (FigureClass): the figureclass

**Returns**

**Return type** Nothing - loads figure formatting

sxdm.viewer.**run_viewer**(*user_class*, *fluor_image*)
Function that compiles functions to get the viewer working :param user_class (SXDMFrameset): the sxdmframeset :param fluor_image (image): the fluorescence image the user would like to load into the viewer

**Returns**

**Return type** Nothing - loads figure data

sxdm.viewer.**savingbtn_click**(*event*, *user_class*, *figure_class*)
Calls the self.save function and reloads data to the viewer :param event (matplotlib event): matplotlib event :param user_class (SXDMFrameset): the sxdmframeset :param figure_class (FigureClass): the figureclass

**Returns**

**Return type** Nothing - loads figure formatting

sxdm.viewer.**spot_change**(*text*, *self*)
When the user changes texbox value spots on the image, reload all the data for the new spot :param text (texbox text event): textbox text event :param self (CurrentFigure Class): the currecnt figure class

**Returns**

**Return type** Nothing - loads figure data

sxdm.viewer.**spot_dif_ram_save**(*self*)
Allows the program to take all the user_class.image_array images and sum them together to get a single summed diffraction image

**Parameters** **(SXDMFrameset)** (*self*) – the sxdmframeset object

**Returns**

**Return type** the summed diffraction image for the entire FOV set by the create_imagearray() function

sxdm.viewer.**sum_error**()
If there is an error loading data, import psyduck

**Returns**

**Return type** psyduck image

sxdm.viewer.**tb_setup**(*vmin_spot_ax*, *vmax_spot_ax*, *vmin_sum_ax*, *vmax_sum_ax*, *med_blur_dis_ax*, *med_blur_h_ax*, *stdev_ax*, *multiplier_ax*, *self*)
Set up all textboxes :param vmin_spot_ax (matplotlib axis): the figure axis that goes to the vmin change for the spot diffraction :param vmax_spot_ax (matplotlib axis): the figure axis that goes to the vmax change for the spot diffraction :param vmin_sum_ax (matplotlib axis): the figure axis that goes to the vmin change for the

summed diffraction :param vmax_spot_ax (matplotlib axis): the figure axis that goes to the vmax change for the summed diffraction :param med_blur_dis_ax (matplotlib axis): the figure axis that goes to the median blur distance value :param med_blur_h_ax (matplotlib axis): the figure axis that goes to the median blur height value :param stdev_ax (matplotlib axis): the figure axis that goes to the standard deviation value :param multiplier_ax (matplotlib axis): the figure axis that goes to the background multiplier value :param self (SXDMFrameset): the sxdmframset

**Returns**

**Return type** All textboxes associated with these axis

sxdm.viewer.**viewer_mouse_click**(*event*, *self*)

Based on what is clicked in the figure change the viewer accordingly :param event (matplotlib event): matplotlib event :param self (FigureClass): the figureclass object

**Returns**

**Return type** Nothing - loads figure formatting

CHAPTER

SIX

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## S

# INDEX

## F



## G



## H



## I



## L



## M

## O



## P



## R



## S

## T



## V



## Z