Estimating Accuracy of X-Ray Diffraction Peak Separation

Lennon F. Seiders
University of Michigan

Abstract

This report investigates the effectiveness of Lab_HEDM.py, a Python script developed by Seunghee Oh for preprocessing X-Ray Diffraction data by removing unwanted secondary peaks. The primary goal of the project is to calculate the success rate of secondary peak removal for the given dataset, and second, to improve the script's ability to replace these peaks with background pixels. The methodology involved generating synthetic data to test the current filtering approach and developing a new algorithm for peak segmentation and removal. This new algorithm was then compared with the preprocessed data to assess the script's performance in removing secondary peaks. Preliminary results have identified instances where Lab_HEDM.py fails to effectively eliminate secondary peaks, though the new algorithm is not yet refined enough to provide an accurate success rate. Further development and testing are required to enhance the precision and reliability of the testing algorithm. Additionally, significant progress was made to develop an effective background replacement method.

Introduction

The first dataset for this project consists of 3600 diffraction images, taken over the course of a complete rotation around the sample. Each image is 4096 x 4096 pixels, and consists of both primary and secondary sets of background peaks, with a considerable amount of background noise. The two sets of peaks are a result of two different wavelengths of light being emitted by the Lab's new MetalJet E1+ 160 kV X-Ray source. Because there are two sets of peaks, a diffraction image analysis on these data would be ineffective, and as a result of this, the second set of peaks must be removed.

The second dataset for this project is the output of a preprocessing algorithm, Lab_HEDM.py. This algorithm removes the set of secondary peaks found in the initial diffraction dataset and replaces each of the peaks with background noise. The success rate of this secondary peak removal must be assessed in order to ensure that the preprocessed dataset may be accurately analyzed.

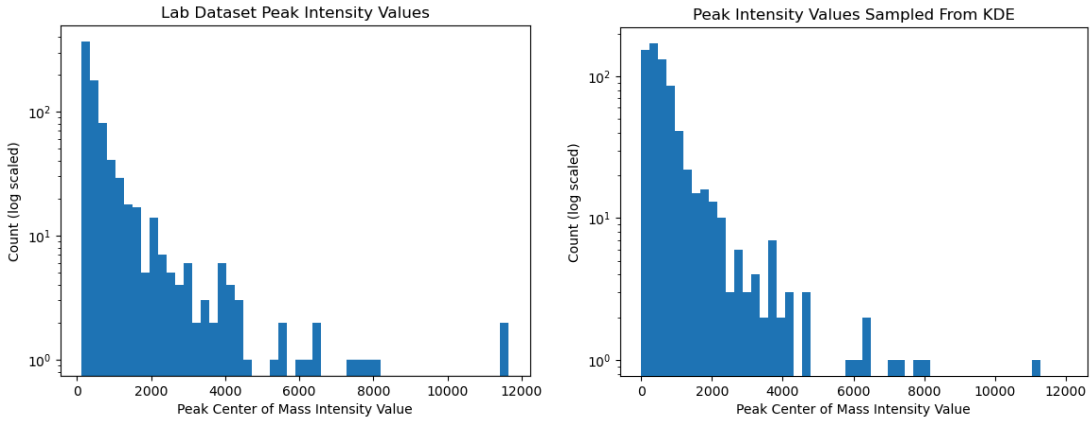
Methodology

In order to estimate the success rate of secondary peak removal between the two datasets, the task was separated into multiple parts, with the end goal of creating a peak-segmentation algorithm to perform the estimation.

First, the design of synthetic data to imitate the data before preprocessing with both primary and secondary peaks. The purpose of the synthetic data was to primarily promote a better understanding of the diffraction images and the preprocessing algorithm. Through statistical methods and analysis of the datasets, the script test_generator.py (Appendix B) was developed to model diffraction images with a number of primary and secondary peaks of size, count, and features defined by customizable parameter values. While some of the peak parameters were determined by heuristic techniques, both peak intensity

values and image background noise were determined each by a gaussian kernel density estimate, which involves fitting a gaussian sampling distribution to data gathered from the original diffraction image sets.

Fig. 1 Lab Dataset vs Synthetic Dataset Peak Intensity Values



As observed in Fig. 1, aspects of the synthetic data such as peak intensity were able to effectively model (although not with complete precision) the diffraction image data. Because of this, the synthetic data was used to test difficult edge cases with different filtering methods.

Second, the methodology involves the selection of a filtering function used in the new testing algorithm (Algorithm.py) for the segmentation of any peaks missed by Lab_HEDM.py. Find_peaks_2d() (found in Appendix C), a function from Python library HEXRD, is used by Lab_HEDM.py in order to identify diffraction peaks by filtering peaks from the background. This function, in combination with OpenCV's morphological transformation cv.MORPH_OPEN (also known as Opening) was used in the latest iteration of Algorithm.py (Appendix A) in order to find missed peaks. This method was motivated by cv.MORPH_OPEN's ability to remove larger, higher intensity pieces of background noise, which then enabled find_peaks_2d() to operate at a lower threshold value for identifying peaks.

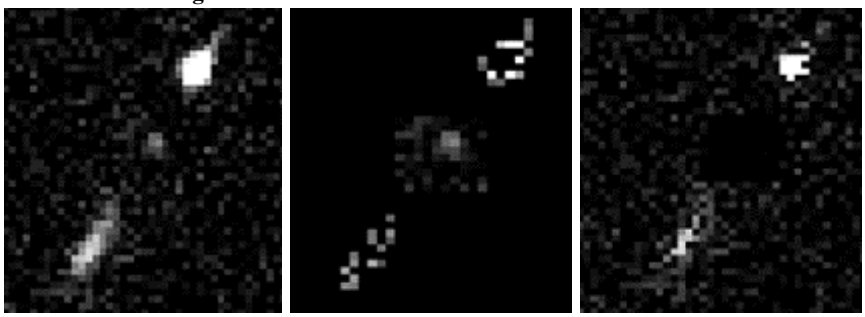
The algorithm created for testing the preprocessed dataset, Algorithm.py, in its current state:

1. For each image in initial dataset, find all peaks and store as polar coordinates
2. Compare each image with the three frames before it and three frames after in order to differentiate between primary and secondary peaks
3. Replace each secondary peak with background noise sampled from gaussian KDE fit to background surrounding the peak using background_generator.py (Appendix E)
4. Save new preprocessed image
5. Compare secondary peaks found by Algorithm.py to secondary peaks removed from diffraction dataset by Lab_HEDM.py, as identified by image_dev.py (Appendix D)

Results and Discussion

Test results were taken from groups of 100 diffraction images, as computing resources were limited. The estimated success rate of secondary peak removal as determined by Algorithm.py was 96.38%, although this result is not accurate. Upon manual inspection of individual peak removal cases for Algorithm.py, it was not uncommon to find misidentified secondary peaks. The majority of observed cases flagged by Algorithm.py, however, were instances where the secondary peak appeared to have been identified by Lab_HEDM.py but it had not been removed properly.

Fig. 2 Incorrect Peak Removal Case in Lab Dataset



In this example of incorrect peak removal, Fig. 2 shows the raw image (left), the preprocessed image (right), and the difference between the two images (center). The primary peak, located in the upper corner of each of the images, should be the only peak showing after preprocessing.

These residuals indicate that while the script performs well in most cases, it struggles with certain edge cases, particularly those involving low-intensity secondary peaks or peaks that are close to primary peaks in intensity and position. This misidentification can be attributed to the limitations in the current peak segmentation approach, particularly in distinguishing between peaks that are closely or distantly positioned, or have overlapping intensity distributions.

Conclusions and Future Outlook

The success rate of 96.38% for secondary peak removal, as roughly estimated by Algorithm.py, suggests that the existing preprocessing script is generally effective at identifying and removing unwanted peaks from the X-Ray Diffraction data. However, the accuracy of this success rate remains questionable due to the observed misidentification of secondary peaks by the new testing algorithm.

The implementation of cv.MORPH_OPEN in combination with the find_peaks_2d() function provided a valuable improvement in isolating true peaks from background noise. This approach allowed for a lower threshold in peak detection, increasing the sensitivity of the algorithm. However, this increased sensitivity also led to a higher rate of false positives, indicating the need for further refinement in distinguishing between true and false peaks.

The use of synthetic data for testing proved beneficial, though not without limitations. Future work should focus on refining the synthetic data generation process to better emulate the complexities of real-world diffraction data. Additionally, the peak segmentation and filtering methods in Algorithm.py need to be further developed to improve accuracy and reduce false positives.

Significant progress has been made in developing an effective background replacement method, which could be integrated into Lab_HEDM.py to enhance its performance. Moving forward, the primary focus should be on refining the testing algorithm to provide a more accurate estimation of the success rate and on improving the peak identification process to reduce errors and increase reliability in the preprocessing of X-Ray Diffraction data.

References

Joel Bernier, Patrick Avery, Saransh, Chris Harris, Zack, Donald Boyce, Brianna Major, darrencpagan, Rachel Lim, john, Óscar Villellas Guillén, & Ryan Rygg. (2024). HEXRD/hexrd: Release 0.9.6 (0.9.6). Zenodo. https://doi.org/10.5281/zenodo.12095606

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. (2020) SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17(3), 261-272.

Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2. (Publisher link).

Walt, S., Schönberger, J., Nunez-Iglesias, J., Boulogne, F., Warner, J., Yager, N., Gouillart, E., Yu, T., & the scikit-image contributors (2014). scikit-image: image processing in Python. PeerJ, 2, e453.

Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools.

Hunter, J. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90–95.

Andrew Collette (2013). Python and HDF5. O'Reilly.

Appendix A: Algorithm.py

```
'''
Author: Lennon Seiders
'''

import h5py
import numpy as np
from scipy import ndimage
import skimage
import imaging
from math import isclose
import os
import math
import cv2

'''
Algorithm for second peak removal. Run this script and modify settings in Main()
in order to segment and remove peaks from diffraction images and/or compare with lab's algorithm.

'''
np.set_printoptions(formatter={'float': '{: 0.3f}'.format})
radii_first = [1068, 1160, 1220, 1599, 1904]
radii_second = [1083, 1171, 1232, 1623, 1932]
r_ranges = [(rad1-10, rad2+10) for rad1, rad2 in zip(radii_first, radii_second)]
filter_thres = 10
filter_radius = 3
theta_margin = 0.04
rho_margin = 30

# Convert cartesian coordinates to polar coordinates with center (center_x, center_y)
def cart2pol(x, y, center_x=2063, center_y=2059):
    dx = x - center_x
    dy = y - center_y
    rho = math.sqrt(dx**2 + dy**2)
    theta = math.degrees(math.atan2(dy, dx))
    return (rho, theta)

# Convert polar coordinates with center (center_x, center_y) to cartesian coordinates
def pol2cart(radius, angle, center_x=2063, center_y=2059):
    angle_rad = math.radians(angle)
    x = center_x + radius * math.cos(angle_rad)
    y = center_y + radius * math.sin(angle_rad)
    return (x, y)

# Iterate through directory and save diffraction image data to a numpy array
def load_images_from_h5_folder(folder, num=3600):
    def load_h5_file(file_path):
        f = h5py.File(file_path, 'r')
        img = np.array(f['imageseries']['images'])
        if img.ndim == 3:
            img = img[0]
        return img
    images = []
    i = 0
```

```
    for filename in sorted(os.listdir(folder)):
        if filename.endswith(".h5"):
            file_path = os.path.join(folder, filename)
            images.append(load_h5_file(file_path))
            i += 1
            if i == num: break
            print (i, end="\r")
    return np.array(images)

# Filter a given image and return a list of peaks for each of the five radii ranges
# 'filter' parameter determines filtering method used: (see imaging.py for more info)
#   'laplace': hexrd's find_peaks_2d()
#   'open': erosion followed by dilation technique
#   'both': opening (erosion and dilation) followed by find_peaks_2d()
def get_pks(img, filter):
    if filter == 'laplace':
        numpks, pks = imaging.find_peaks_2d(img, 'label',{'filter_radius':filter_radius,
'threshold':filter_thres})
    elif filter == 'open':
        numpks, pks = imaging.find_peaks_2d_open(img,
cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3)))
    elif filter == 'both':
        img = cv2.morphologyEx(img, cv2.MORPH_OPEN, cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3)))
        numpks, pks = imaging.find_peaks_2d(img, 'label',{'filter_radius':filter_radius,
'threshold':filter_thres})
    else:
        print(filter, 'is not a valid filtering method')
        exit(1)

    newpk_lists = [[], [], [], [], []]
    for pk in pks:
        pcord = cart2pol(pk[1], pk[0])
        for i in range(0,5):
            if pcord[0] >= r_ranges[i][0] and pcord[0] <= r_ranges[i][1]:
                newpk_lists[i].append(pcord)
                break
    return newpk_lists

# Compares peak lists of two scans and creates a list of peaks from scan_a that should be removed
def find_2pks(scan_a, scan_b):
    # pk_a: candidate for removal
    # pk_b: comparison peak
    def is2nd(pk_a, pk_b):
        return isclose(pk_b[1], pk_a[1], abs_tol=theta_margin) and (pk_b[0] + 5 < pk_a[0])

    removepks = []
    for rad_a, rad_b in zip(scan_a, scan_b):
        for pk in rad_a:
            for pk_b in rad_b:
                if is2nd(pk, pk_b):
                    removepks.append(pol2cart(pk[0], pk[1]))
    return removepks

# Adds 2d array image to file.
#   If mkdir = True: create new '<filename>.h5' file for image data
#   If mdir = False: add new 'removed' image to existing '<filename>.h5'
```

```
def im2file(img, filename, mkdir=False):
    if mkdir:
        if os.path.isfile(filename):
            os.remove(filename)
        f = h5py.File(filename, 'w')
        f.create_group("imageseries")
        f['imageseries']['images'] = img
    else:
        f = h5py.File(filename, 'r+')
        f['imageseries']['removed'] = img

# Given a list of peaks to remove, replace each with background sampled from gaussian kde
def remove_2pks(toRemove, img):
    f = h5py.File('background.h5', 'r')
    bg = np.array(f['synthetic_bg'])
    for pk in toRemove:
        img[pk[0]-4:pk[0]+5, pk[1]-4:pk[1]+5] = bg[pk[0]-4:pk[0]+5, pk[1]-4:pk[1]+5]

    return img

# Removes second peaks from images in directory.
#   If overwrite = True: replace each file with a new .h5 file containing image data with peaks removed
#   If overwrite = False: add peaks removed image to each existing file in directory
def find_to_remove(raw_scans):
    scan_pks = [get_pks(scan, 'both') for scan in raw_scans]
    remove_pks = []
    for i in range(len(scan_pks)):
        scan_remove = []
        for j in range(i-3, i+4):
            if j >= 0 and j < len(scan_pks):
                toRemove = find_2pks(scan_pks[i], scan_pks[j])
                scan_remove.extend(toRemove)
        remove_pks.append(scan_remove)
    remove_unique=[np.unique(scan, axis=0) for scan in remove_pks]
    remove_unique = [[[round(pk[0]), round(pk[1])]for pk in pks] for pks in remove_unique]

    return remove_unique

# Runs peak removal algorithm on n='images_to_process' images in a user-defined folder.
# If used with lab dataset, set img_dev = True in order to compare this algorithm to the
# algorithm used to create the preprocessed image dataset
def main():
    images_to_process = 100
    folder = 'nobg_2024-03-13-21-19-01-scan Ti7Al_z25p5_deg360_step0p1_rot45'
    #folder = 'synthetic_images'

    write_to_file = True # set to true for creating new preprocessed files
    img_dev = True # set to true when testing lab data
    verbose = True # output second peaks removed in each scan by algorithm.py and not removed by lab
algorithm

    raw_scans = load_images_from_h5_folder(folder, images_to_process)
    to_remove = find_to_remove(raw_scans)

    # Compare results of lab algorithm and algorithm.py.
    # Requires user to run image_dev.py in order to generate deviation images.
```

```
    if img_dev:
        lab_found = 0
        algorithm_found = 0
        diff_found = 0
        dev_dir = 'img_dev'
        lab_removed = []
        for i, file in enumerate(os.listdir(dev_dir)):
            if i > images_to_process: break
            f = os.path.join(dev_dir, file)
            f1 = h5py.File(f, 'r+')
            lab_removed.append(np.array(f1['removed_peaks']))

        i = 0
        for lab_scan, found_scan in zip(lab_removed, to_remove):
            i += 1
            scan_diff_lab = []
            scan_diff_algorithm = []
            lab_found += len(lab_scan)
            algorithm_found += len(found_scan)
            # found in this algorithm but not found by lab's
            for pk in found_scan:
                if pk not in lab_scan:
                    scan_diff_algorithm.append(pk)
            diff_found += len(scan_diff_algorithm)
            if verbose:
                print('image', i)
                print('found in algorithm.py but not img_dev: ', scan_diff_algorithm)

        print()
        print(images_to_process, 'images tested')
        print('approximate secondary peaks identified by LabHEDM.py:', lab_found)
        print('estimated secondary peaks not removed: ', diff_found, ' (',
diff_found/(lab_found+diff_found), '%)', sep='')

    # Write new files with secondary peaks removed
    if write_to_file:
        if not os.path.exists('processed_images'):
            os.makedirs('processed_images')
        for i in range(0, len(raw_scans)):
            img = remove_2pks(to_remove[i], raw_scans[i])
            fname = os.path.join('processed_images', 'removed_'+ format(i+1, '04') +'.h5')
            im2file(img, fname, mkdir=True)

    return



if __name__ == "__main__":
    main()
```

Appendix B: test_generator.py

```
'''
Author: Lennon Seiders
'''

import h5py
import random
import numpy as np
from scipy import ndimage
from scipy.stats import gaussian_kde
import imaging
import os
import cv2
import skimage
import background_generator
import math

'''
Functions for generating and testing synthetic image data. Images are saved in .h5 files
along with a list of primary and secondary peak coordinates.

Set parameter values in main() and run this script in order to generate synthetic images.

'''

# Load peak value data from lab dataset
pkvals = np.loadtxt('pkvals.csv', dtype=np.int16, converters=float)
ints_kde = gaussian_kde(pkvals)

def sample(kde):
    x = kde.resample(1)[0]
    while x < np.min(pkvals):
        x = kde.resample(1)[0]
    return x[0]

# Generate synthetic based on input parameters
def generate_radial_peaks_image(peaks_per_level, newImageFile, p_second, p_tail, size_mult):
    size = (4096, 4096)
    image = np.zeros(size, dtype=np.int16)
    center = (2063, 2059)
    radii_first = [1068, 1160, 1220, 1599, 1904]
    radii_second = [1083, 1171, 1232, 1623, 1932]

    # Create a 2d peak array. Higher peak intensity correlated with larger size.
    def create_peak_structure(coord, size, peakIntensity):
        mdl = int(size/2)
        chunk = np.zeros((9,9))
        peak = np.random.randint(0, peakIntensity, (3,3), dtype=np.int16)
        peak[1][1] = peakIntensity
        if size > 4: peak = np.pad(peak, (mdl-1,), mode='linear_ramp', end_values=np.random.randint(0,
np.min(peak)+1, dtype=np.int16))
        peak = peak * cv2.getStructuringElement(cv2.MORPH_ELLIPSE, ksize=(size,size))
        chunk[4-(mdl):5+(mdl),4-(mdl):5+(mdl)] = peak
        image[coord[0]-4:coord[0]+5, coord[1]-4:coord[1]+5] = chunk
```

```
    # Place peaks on each radius. Peaks_per_level primary peaks + secondary peaks generated with
probability p_second.
    primary_peaks = []
    secondary_peaks = []
    for i, radius in enumerate(radii_first):
        for _ in range(peaks_per_level):
            angle = np.random.uniform(0, 2 * np.pi)
            x = int(center[0] + radius * np.cos(angle))
            y = int(center[1] + radius * np.sin(angle))
            intensity = sample(ints_kde)
            size = 3 + int(ints_kde.integrate_box_1d(0, intensity) * 2 * size_mult) * 2
            create_peak_structure([x,y], size, intensity)
            primary_peaks.append((x, y))
            if random.random() < p_second:
                dx = int(center[0] + radii_second[i] * np.cos(angle))
                dy = int(center[1] + radii_second[i] * np.sin(angle))
                create_peak_structure([dx,dy], size, intensity)
                secondary_peaks.append((dx, dy))
                if (random.random() < p_tail) and math.dist([x,y],[dx,dy]) < 13:
                    rr, cc = skimage.draw.line(x,y,dx,dy)
                    mid_x = int((rr[0] - rr[-1]) / 2)
                    mid_y = int((cc[0] - cc[-1]) / 2)
                    image[mid_x, mid_y] = intensity/(math.dist([x,y],[dx,dy])+(7-size))
                    a = np.geomspace(image[x,y], image[mid_x, mid_y], num=int((len(rr)/2) + 1))
                    b = np.geomspace(image[mid_x, mid_y], image[dx, dy], num=int(len(rr)/2))
                    tailvalues = np.hstack((a, b))
                    for j in range(0, len(rr)):
                        image[rr[j], [cc[j]]] = tailvalues[j]

    # Gaussian smoothing and adding  background noise
    image = np.maximum(ndimage.gaussian_filter(image, sigma=0.7), image)
    try: bg_f = np.array(h5py.File('background.h5','r')['synthetic_bg'])
    except:
        background_generator.main()
        bg_f = np.array(h5py.File('background.h5','r')['synthetic_bg'])
    finally: image += bg_f

    # Set synthetic image file
    newImageFile.create_group("imageseries")
    newImageFile['imageseries']['images'] = image
    newImageFile['imageseries']['primaryPeaks'] = primary_peaks
    newImageFile['imageseries']['secondaryPeaks'] = secondary_peaks
    return

# Create synthetic image file, generate image and peak lists
def generate_images(num_images, peaks_per_level=4, p_second=0.5, p_tail=0.75, size_mult=1):
    print("generating", num_images, "diffraction images...")
    if not os.path.exists('synthetic_images'):
        os.makedirs('synthetic_images')
    for i in range(num_images):
        filename = 'synthetic_image_' + format(i+1, '04') + '.h5'
        if os.path.isfile('synthetic_images/' + filename):
            os.remove('synthetic_images/' + filename)
        newImageFile = h5py.File('synthetic_images/' + filename,'a')
        generate_radial_peaks_image(peaks_per_level, newImageFile, p_second, p_tail, size_mult)
```

```
# Test to run different filtering methods on synthetic data. Keeps track of peaks missed and false
positives.
def test_synthetic_images(thres, directory='synthetic_images'):
    missed = 0
    false_positive = 0
    correct = 0
    total_pks = 0
    i = 0
    for filename in os.listdir(directory):
        i += 1
        f = os.path.join(directory, filename)
        x, numpks = imaging.find_peaks_2d_test(f, thres)
        total_pks += numpks
        if x > 0: false_positive += x
        elif x < 0: missed -= x
        else: correct += 1
    print(total_pks, 'total peaks tested in', i, 'images')
    print('correct images:', correct)
    print('missed:', missed)
    print('incorrectly classified:', false_positive)

def main():
    num_images = 10 # number of synthetic diffraction images to be generated
    peaks_per_level = 4 # peaks per radius level
    p_second = 0.5 # probability of secondary peaks being generated for each primary peak
    p_tail = 0.5 # probaility of a pair of peaks having additional "tail" noise
    size_mult = 1 # size multiplier. higher value (ex. 1.3) results in a higher chance of generating
large peaks
    generate_images(num_images, peaks_per_level, p_second, p_tail, size_mult)

if __name__ == "__main__":
    main()
```

Appendix C: imaging.py

```
'''
Author: Lennon Seiders
'''

import h5py
import numpy as np
from scipy import ndimage
import cv2

'''
This file contains hexrd's (https://github.com/HEXRD/hexrd) find_peaks_2d() function for filtering
diffraction images,
as well as an alternate method and a test to be used with synthetic data.

These functions are to be used in other scripts for filtering and testing.

'''

sigma_to_fwhm = 2.*np.sqrt(2.*np.log(2.))
fwhm_to_sigma = 1. / sigma_to_fwhm # = 0.42....

# Finds peak structures by using scipy.ndimage.gaussian_laplace().
# gaussian_laplace() uses a gaussian filter followed by a second derivative measurement using a
laplacian kernel.
def find_peaks_2d(img, method, method_kwargs):
    if method == 'label':
        # labeling mask
        structureNDI_label = ndimage.generate_binary_structure(2, 1)

        # First apply filter if specified
        filter_fwhm = method_kwargs['filter_radius']
        if filter_fwhm:
            filt_stdev = fwhm_to_sigma * filter_fwhm
            img = -ndimage.filters.gaussian_laplace(
                img, filt_stdev
            )

        labels_t, numSpots_t = ndimage.label(
            img > method_kwargs['threshold'],
            structureNDI_label
            )
        coms_t = np.atleast_2d(
            ndimage.center_of_mass(
                img,
                labels=labels_t,
                index=np.arange(1, np.amax(labels_t) + 1)
                )
            )

    return numSpots_t, coms_t

# Finds peak structures by eroding foreground with given kernel and then dilating
def find_peaks_2d_open(img, kernel=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3)), thres=40):
```

```
    img = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
    structureNDI_label = ndimage.generate_binary_structure(2, 1)

    labels_t, numSpots_t = ndimage.label(img > thres, structureNDI_label)
    coms_t = np.atleast_2d(ndimage.center_of_mass(
        img,
        labels=labels_t,
        index=np.arange(1, np.amax(labels_t) + 1)
        )
    )
    return numSpots_t, coms_t

# Tests find_peaks_2d() on a synthetic diffraction image. Returns peaks missed or false positives.
def find_peaks_2d_test(filename, thres=38):
    print("testing", filename)
    f = h5py.File(filename, 'r+')
    img = np.array(f['imageseries']['images'])
    if img.shape == (3,):
        img = img[0,:,:]
    img_primary_peaks = np.array(f['imageseries']['primaryPeaks'])
    img_secondary_peaks = np.array(f['imageseries']['secondaryPeaks'])
    true_pks =np.append(img_primary_peaks, img_secondary_peaks, axis=0)
    true_pks = np.sort(true_pks, axis=0)

    found_pks = find_peaks_2d(img, 'label',{'filter_radius':3, 'threshold':thres})[1]
    found_pks = np.sort(found_pks, axis=0)

    # Check for shared pairs
    try:
        identical = np.allclose(true_pks, found_pks, atol=1)
        # if identical: print('correctly identified')
        # else: print('true peaks and peaks found not identical')
        return 0, len(true_pks)
    except:
        identical = 0
        # if len(true_pks) > len(found_pks):
        #     print('missing', len(true_pks) - len(found_pks), 'peaks')
        # else:
        #     print('incorrectly classified', len(found_pks) - len(true_pks), 'peaks')
        return len(found_pks) - len(true_pks), len(true_pks)
```

Appendix D: image_dev.py

```
'''
Author: Lennon Seiders
'''

import os
import h5py
import numpy as np
import cv2
import imaging

'''
Creates deviation images by subtracting preprocessed images from raw scans.
Resulting images are peaks and noise removed by preprocessing algorithm:
  'image_dev': img_pre - img_post
  'dev_filtered': image_dev masked to only show peaks near radii, eroded and then dilated
  'dev_peaks': list of peaks found in image_dev; peaks that have been removed by preprocessing
algorithm

Set directory paths and number of images to generate prior to running this script.

'''
num_files = 20 # choose how many deviation images to create from lab dataset

raw_directory = 'nobg_2024-03-13-21-19-01-scan Ti7Al_z25p5_deg360_step0p1_rot45'
prep_directory =
'Ti7Al_deg360_step0p1_after_preprocessing\Ti7Al_z25p5_deg360_step0p1_rot45_0to3600_ver3p1_mult1p6_3600_
mimg9'
result_directory = 'img_dev/'

size = (4096, 4096)
center = (2063, 2059)
radii_first = [1068, 1160, 1220, 1599, 1904]
radii_second = [1083, 1171, 1232, 1623, 1932]
mask = np.zeros(size, dtype=bool)
mask2 = np.zeros(size, dtype=bool)
x = np.arange(size[0])
y = np.arange(size[1])
xx, yy = np.meshgrid(x, y)
distances = np.sqrt((xx - center[0])**2 + (yy - center[1])**2)
for r1, r2 in zip(radii_first, radii_second):
    mask |= (distances >= r1 - 10) & (distances <= r2 + 10)

# optional function to draw dotted lines along each radius
def set_radii_elements(array, center, radii, value):
    for radius in radii:
        for angle in range(360):
            radian = np.deg2rad(angle)
            x = int(center[0] + radius * np.cos(radian))
            y = int(center[1] + radius * np.sin(radian))
            if 0 <= x < array.shape[0] and 0 <= y < array.shape[1]:
                array[x, y] = value

# Create num_files amount of img_dev images
```

```
files1 = sorted(os.listdir(raw_directory))
files2 = sorted(os.listdir(prep_directory))
count = 0
for raw_f, prep_f in zip(files1[:num_files], files2[:num_files]):
    raw = os.path.join(raw_directory, raw_f)
    prep = os.path.join(prep_directory, prep_f)

    f1 = h5py.File(raw, 'r')
    r_img = np.array(f1['imageseries']['images'][0,:,:])
    f1.close()
    f2 = h5py.File(prep, 'r')
    p_img = np.array(f2['imageseries']['images'][0,:,:])
    f2.close()

    new_filename = result_directory + 'dev_' + format(count+1, '04') + '.h5'
    img_dev = r_img - p_img

    if os.path.isfile(new_filename):
        os.remove(new_filename)

    result = h5py.File(new_filename, 'a')
    result['image_dev'] = img_dev

    count += 1

# Filter img_dev to isolate peaks
for dev_f in os.listdir(result_directory):
    f = os.path.join(result_directory, dev_f)
    f1 = h5py.File(f, 'r+')
    dev_img = np.array(f1['image_dev'])

    masked_image = np.where(mask, dev_img, 0)
    masked_image = cv2.morphologyEx(masked_image, cv2.MORPH_OPEN,
cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3)))
    numpks, pks = imaging.find_peaks_2d(masked_image, 'label',{'filter_radius':3, 'threshold':5})
    coords = [np.array([round(pk[1]), round(pk[0])]) for pk in pks]
    f1['dev_filtered'] = masked_image
    f1['removed_peaks'] = coords
```

Appendix E: background_generator.py

```
'''
Author: Lennon Seiders
'''

import h5py
import os
import numpy as np
import scipy.stats as stats

'''
Script for obtaining accurate background noise by sampling windowed kernel desnsity estimates of lab
dataset.
Run this scripy to create a file titled 'background.h5', synthetic background noise mimicking that
of the lab dataset's images.

'''

directory = 'nobg_2024-03-13-21-19-01-scan Ti7Al_z25p5_deg360_step0p1_rot45'
num_images = 1
if os.path.isfile('gaussian_kde_full.h5'):
    os.remove('gaussian_kde_full.h5')

# Creates mask over peak radii in order to remove peaks from background data
def set_points_along_radii(img):
    size = img.shape
    center = (2063, 2059)
    radii_first = [1068, 1160, 1220, 1599, 1904]
    radii_second = [1083, 1171, 1232, 1623, 1932]
    x = np.arange(size[0])
    y = np.arange(size[1])
    xx, yy = np.meshgrid(x, y)

    distances = np.sqrt((xx - center[0])**2 + (yy - center[1])**2)
    for r1, r2 in zip(radii_first, radii_second):
        mask = (distances >= r1 - 15) & (distances <= r2 + 15)
        img[mask] = -1

    return img

# Get lab dataset file for background sampling
def get_background(path):
    file = h5py.File(path, 'r')
    img = np.array(file['imageseries']['images'], dtype=np.int16)
    if img.shape == (3,):
        img = img[0,:,:]
    img = np.squeeze(img, axis=0)
    return set_points_along_radii(img)

# Iterate over 2^windowing_exp windows of img,
# Create and sample a gaussian kernel density estimate at each window to get new background
def background_to_gaussians(img, windowing_exp=5):
    size_x = 4096
    kde_list = []
```

```
    num_windows = pow(2, windowing_exp)
    window_size = int(size_x/num_windows)

    for window_row in range(num_windows):
        for window_col in range(num_windows):
            start_row = window_size * window_row
            end_row = start_row + window_size
            start_col = window_size * window_col
            end_col = start_col + window_size
            sample_values = img[start_row:end_row, start_col:end_col].flatten().tolist()
            weights = [1 if x != -1 else 0 for x in sample_values]

            kde = stats.gaussian_kde(sample_values, weights = weights)
            kde_list.append(kde)
            new_samples = np.array(kde.resample((window_size)*(window_size)).flatten().tolist(),
dtype=np.int16)
            new_samples[new_samples < 0] = 0
            new_samples[new_samples > 100] = 0
            noise_image = new_samples.reshape(window_size, window_size)

            img[start_row:end_row, start_col:end_col] = noise_image

    return img

# Run script
def main():
    path = 'nobg_2024-03-13-21-19-01-scan Ti7Al_z25p5_deg360_step0p1_rot45/Raw_scan_0001.h5'
    img = get_background(path)
    if os.path.isfile('background.h5'):
        os.remove('background.h5')
    newImageFile = h5py.File('background.h5','a')
    newImageFile['synthetic_bg'] = background_to_gaussians(img)

    return

if __name__ == "__main__":
    main()
```