XRD DATA ANALYSIS BY USING PYTHON AND APPLICATION OF PYMATGEN IN MATERIAL SCIENCE

A Project Report Submitted by

DEPARTMENT OF PHYSICS FERGUSSON COLLEGE,PUNE

In Partial Fulfilment of the Requirement for the Degree

MASTER OF SCIENCE IN PHYSICS
By

Mr. DIPAK BABASAHEB KHEDKAR

Under the guidance of

DR. KAILASH B. SAPNAR

Project work done at

DEPARTMENT OF PHYSICS FERGUSSON COLLEGE, PUNE

Deccan Education Society's

Fergusson College (Autonomous), Pune.

Department of Physics

CERTIFICATE

This is to certify that the project entitled

"XRD data analysis by using Python and application of Pymatgen in Material science" is submitted by:

Roll Number: **211296**

Name: **Dipak B. Khedkar**

In partial fulfillment of the requirement of the completion of M.Sc. Semester IV, has been carried out satisfactorily by the team under my guidance during the academicyear **2022-2023**.

Place: Pune

Date: 2 9 / 04/ 2023

Prof. Dr. Kailash Sapnar

Project Guide

Prof. Dr. Nandkumar T. Mandlik

Head and Professor,

Department of Physics,

Fergusson College (Autonomous), Pune-4

ACKNOWLEDGEMENT

I am extremely thankful to Principal Dr. R. G. Pardeshi and Prof. Nandakumar T. Mandlik, Head, Department of Physics, Fergusson College, Pune, who poured every inch of the knowledge about the subject in me and for providing me all necessary college facilities and conductive environment for the subject.

I am greatly thankful to my research supervisors **Dr. Kailash B. Sapnar** for his endless guidance, encouragement, motivations, patience and continuous support throughout my M.Sc. project.

I record my sincere thanks to Department of Physics of Savitribai Phule Pune University. I would like to acknowledge the Physics Department of Fergusson College. I would like to thanks to my project guide Dr. Kailash B. Sapnar for their valuable time and suggestions. I am also grateful to *Aniket Bembale* and *Lotous IT institute*, *pune* for helping me a lot. I would like to thanks to all faculty members of Department of Physics who have made director indirect contribution towards the completion of my project.

It gives me an enormous pleasure to thank all my friends and all the research scholars of Department of Physics. In particular, I would like to thank my parents for their love and patience.

Dipak Babasaaheb Khedkar

ABSTRACT

X-ray diffraction (XRD) analysis is a key tool in materials science research for identifying crystal structures, determining inter-planner spacing d, and measuring grain sizes. In this project, we utilized the Python programming language and its libraries, such as pandas, matplotlib, scipy, numpy. Specifically the pandas library for data analysis and the pymatgen library for materials analysis and reference details. To automate and accelerate XRD data analysis using python, We loaded XRD data into pandas DataFrames, preprocessed the data with pandas functions, and visualized the patterns with Matplotlib and by processing data. In this project we also try to correct the baseline. After plotting xrd data we detect the maximum intensity peaks and record the corresponding 2theta values and perform number of operations from that we calculate d-spacing and approx. values of a=b and c that is 2.43 and 4.90 resp. We also determine the grain size for Zn powder which lies in range 5.9 to 17.6 nm. We also used the pymatgen library to perform crystalline phase identification, grain size calculation, reference properties and crystal lattice operations. From pymatgen we also take reference for number of structural parameter such as a, b, c values, surface energy, band gap, etc. Our results showed that this approach increases the efficiency and accuracy of XRD data analysis, it will reduce the dependency of paid software's and also reduce the human error by making it a valuable tool in materials science research.

INDEX

SR. NO.	CONTENT	PAGE NO.
CHAPTER 1	**Introduction & Theoretical Background** 1.1 X-Ray Diffraction 1.2 Pandas Library 1.3 Numpy Library 1.4 Matplotlib Library 1.5 Pymatgen	6-12
CHAPTER 2	**Programing work** **2.1 XRD Analysis:** 2.1.1 Installation 2.1.2 Import Libraries 2.1.5 Plot XRD data 2.1.7 Read reference data 2.1.8 Comparison of data 2.1.9 Calculate d-spacing 2.1.10 Calculate Approx. a, b, c values 2.1.11 Calculate Grain Size (D) 2.1.12 Baseline correction **2.2 Pymatgen Application** 2.2.1 Installation 2.2.2 Surface properties 2.2.3 Structure 2.2.4 Fcc Lattice 2.2.5 NaCl Lattice 2.2.6 Crystal Lattice From Composition 2.2.7 Replace Atoms in Lattice	13-26
CHAPTER 3	**Results and Discussion** 3.1 Peak Detection 3.2 d-spacing 3.3 comparison 3.4 Grain Size 3.5 Structure Parameters 3.6 Use of Pymatgen	27-33
CHAPTER 4	**Conclusion**	34
CHAPTER 5	**Reference**	35

CHAPTER :1

Introduction & Theoretical background

1.1 X-Ray Diffraction

X-rays are electromagnetic radiation They have same nature of light and having very short wavelength (1-5 A^0). As compared to the light radiations (visible) they are much more penetrating than light.. X-ray diffraction is used to obtain the crystal structure because the crystal works as three- dimensional grating (this is due to the fact that the inter planner distances in a given crystal are of the order of incident wavelengths). When an x-ray beamis passed through a crystal, the scattering can be treated on the basis of the interferencesof the waves reflected from the crystal planes.

The X- ray diffraction technique is one of the most commonly used structure characterization technique for studying powder form as well as thin form. It is used uniquely to identify the crystalline phase of the material, the out of

the orientation of the film grain and relatively plane alignment of the thin film. With the substrate or between different layers in the case of hetero structures in normal θ-2θ scanned mode.

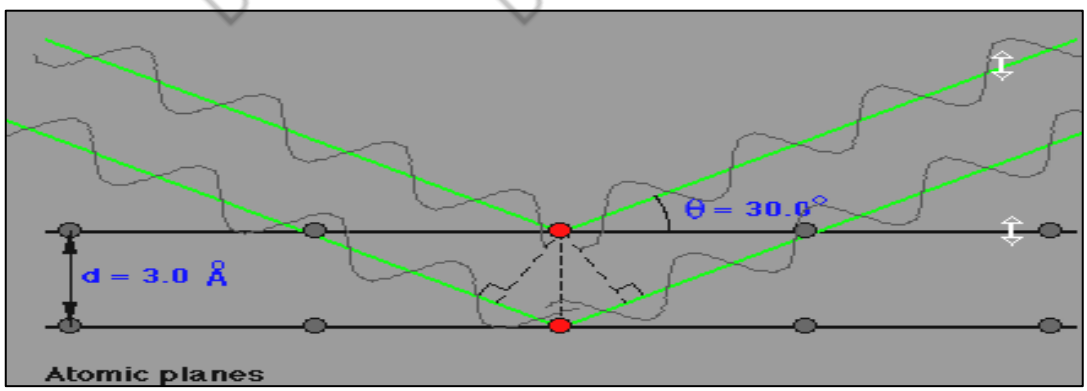

Fig: Schematic ray diagram for XRD

(*http://hyperphysics.phy-astr.gsu.edu/hbase/quantum/bragg.html)*

A beam of X- ray is incident on sample. Making angle θ (angle between the surface of thin film and the incident beam) this ray is penetrated deep inside the sample and reflected by the different plane of the sample. As shown in the fig. the scattered beamreinforced each other when their path difference is integral multiple of wavelength. Path difference between the two scattered rays is 2dsinθ. The condition for constructive interference is-

$$\mathbf{2dsin\theta = n\lambda}$$

(This is Bragg's condition.)

As this condition is satisfied then we get peak. For the sharp peak in intensity there is condition of scattered radiation is

1. The X-ray should be reflected by ions in any one plane.
2. The reflected rays should interfere constructively

From X-ray one can also calculate the grain size by using Scherer formula-

$$t = \frac{\mathbf{0.9\lambda}}{Q\boldsymbol{cos\theta}\mathbf{B}}$$

1.2 Pandas Library

Pandas is a Python library for data manipulation and analysis. It is built on top of the NumPy package and is a very popular library among data scientists and analysts. Pandas provides easy-to-use data structures and data analysis tools for handling structured data.

1.2.1 Installation

To install the Pandas library, you can use pip, the Python package manager. Open your command prompt or terminal and run the following command:

```
pip install pandas
```

1.2.2 Data Structures

Pandas provides two main classes for handling data: Series and DataFrame.

1.2.3 Series

A Series is a one-dimensional labeled array that can hold any data type (integer, float, string, etc.). It is similar to a column in a spreadsheet or a SQL table.

```
import pandas as pd
s = pd.Series([1, 3, 5, np.nan, 6, 8])
```

1.2.4 DataFrame

A DataFrame is a two-dimensional labeled data structure with columns of potentially different types. It is similar to a spreadsheet or a SQL table

.

```
import pandas as pd
import numpy as np
data = {'name': ['Alice', 'Bob', 'Charlie', 'Dave'],
        'age': [25, 32, 18, 47],
        'gender': ['F', 'M', 'M', 'M']}
```

1.2.5 Data Analysis Tools:

Pandas provides a variety of data analysis tools, such as:

- Filtering and sorting data
- Data aggregation and grouping
- Merging and joining datasets
- Reshaping and pivoting data
- Handling missing data
- Time series analysis

1.2.6 Reading CSV Files:

Pandas can be used to read CSV (Comma Separated Value) files, which are a common format for storing and sharing tabular data. The “read_csv” function can be used to read CSV files into a DataFrame object:

```
import pandas as pd
df = pd.read_csv('data.csv')
```

Apart from reading CSV files, Pandas provides functions to read various other file formats as well. Here are some commonly used functions for reading different types of files using Pandas:

- read_csv(): Reads a comma-separated values (CSV) file into a DataFrame.
- read_excel(): Reads an Excel file into a DataFrame.
- read_json(): Reads a JSON file into a DataFrame.
- read_sql(): Reads SQL query or database table into a DataFrame.
- read_html(): Reads an HTML file into a list of DataFrame objects.

These functions provide various parameters to customize the way the data is read into a DataFrame. For example, you can specify the column names, delimiter, encoding, header rows, and many other options depending on the file format and the structure of the data.

1.2.7 Conclusion:

Pandas is a powerful library for data manipulation and analysis in Python. It provides easy-to-use data structures and data analysis tools for handling structured data.

1.3 Numpy Library

Numpy is a Python library for scientific computing that provides support for arrays and matrices, as well as a large collection of mathematical functions to operate on them efficiently. Numpy arrays are faster and more memory-efficient than traditional Python lists and allow for easy manipulation of multi-dimensional arrays.

1.3.1 Arrays

The core object in Numpy is the ndarray, which is a multi-dimensional array of elements of the same type. Here's an example of creating a 2D ndarray:

```
import numpy as np
a = np.array([[1, 2, 3], [4, 5, 6]])
```

1.3.2 Mathematical Operations

Numpy provides a large collection of mathematical functions to operate on arrays, including:

- Trigonometric functions (sin, cos, tan, etc.)
- Exponential and logarithmic functions
- Basic arithmetic operations (addition, subtraction, multiplication, division, etc.)
- Linear algebra operations (dot product, matrix multiplication, etc.)
- Statistical functions (mean, standard deviation, variance, etc.)

Here's an example of using the sin() function on a Numpy array:

```
import numpy as np
a = np.array([0, np.pi/2, np.pi])
np.sin(a)
```

1.4 Matplotlib Library

Matplotlib is a Python library for creating static, animated, and interactive visualizations in Python. It provides a wide variety of customizable plots, including line plots, scatter plots, bar plots, histograms, 3D plots, and more.

Matplotlib provides many ways to customize plots, including adding titles, axes labels, legends, and changing colors and line styles. It also makes it easy to create multiple plots in the same figure using subplots.

Matplotlib is widely used in scientific computing, data analysis, and machine learning. It is an essential tool for anyone working with data in Python. Key features of Matplotlib include:

- Create a wide variety of customizable plots, including line plots, scatter plots, bar plots, histograms, 3D plots, and more
- Customize plots with titles, axes labels, legends, and changing colors and line styles
- Create multiple plots in the same figure using subplots
- Support for LaTeX formatted text and mathematical expressions in plot text
- Works well with NumPy, Pandas, and other Python libraries

Matplotlib is a mature and reliable library that has been in development for over 15 years. It is actively maintained and has a large and supportive community of users and developers.

1.4.1 Basic Line Plot

Here's an example of creating a basic line plot using Matplotlib:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 100)
y = np.sin(x)
plt.plot(x, y)
plt.show()
```

1.5 Pymatgen

Pymatgen is a Python library for materials analysis. It provides tools for generating, analyzing, and manipulating materials structures. Pymatgen is built on top of the Python scientific computing ecosystem, including NumPy, SciPy, and matplotlib.

1.5.1 Materials Analysis

Pymatgen provides a variety of tools for analyzing materials structures, such as:

- Parsing and generating structures in various file formats, such as CIF, POSCAR, and XYZ
- Analyzing bond lengths and angles
- Calculating density and porosity
- Generating surface slabs and interfaces
- Calculating band structures and density of states

1.5.2 Crystallography

Pymatgen provides tools for working with crystallographic data, such as:

- Calculating and plotting X-ray diffraction patterns
- Analyzing crystal symmetry and generating symmetry operations
- Performing Fourier transforms and structure factors
- Converting between crystallographic and Cartesian coordinates

1.5.3 Electronic Structure

Pymatgen provides tools for working with electronic structure data, such as:

- Parsing electronic structure data from VASP and other DFT codes
- Generating input files for DFT calculations
- Calculating band structures and density of states
- Visualizing electronic structure data using matplotlib

CHAPTER: 2
Programing Work

2.1 XRD Analysis:
2.2 Pymatgen Application:

2.1 XRD Analysis:

2.1.1 Install Required Python Libraries:

for which we used pip command as package management system which used to install required package in the python environment.

```
!pip install pandas
!pip install numpy
!pip install matplotlib
!pip install scipy
!pip install peakutils
```

2.1.2 Import required libraries:

Import required libraries such as pandas for data analysis, Numpy for working on array to perform mathematical operations, matplotlib for the data visualization and Scipy for additional support to perform array operations.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import find_peaks
```

2.1.3 Read XRD data csv file:

Read the xrd data by pandas in which we use the command **read_csv** and provide the file location as argument, which will read the file and convert it into pandas DataFrame.

```
data = pd.read_csv('G:\msc project sources\XRD\Group 1 - Zinc Powder.csv')
data
```

	Angle (2theta)	Intensity (Counts)
0	20.03	1
1	20.09	1
2	20.15	1
3	20.21	3
4	20.27	2
..	...	...

2.1.4 Collect information about dataset such as data types:

```
data.info()
```

Column	Dtype
Angle (2theta)	float64
Intensity (Counts)	int64

Checking the data to filter out null spaces to avoid errors in array operations:

```
data.isnull().sum()
```

Angle (2theta) 0 –null values
Intensity (Counts) 0 –null values
dtype: int64

In above output we can see the dtype is int64 we must have convert it into the float data type to avoid further error in the mathematical ops. So,

```
data['Angle (2theta)'] = data['Angle (2theta)'].astype('float')
```

2.1.5 Plot XRD data:

Now we are going to plot the XRD data for Zinc powder as Intensity vs 2-theta, for which we assume x variable as x axis and take its value as 2-theta

```
x=data['Angle (2theta)']
y=data['Intensity (Counts)']
plt.plot(data['Angle (2theta)'], data['Intensity (Counts)'])
plt.xlabel('2-theta')
plt.ylabel('Intensity')
plt.title('XRD pattern for Zinc powder')
plt.show()
```

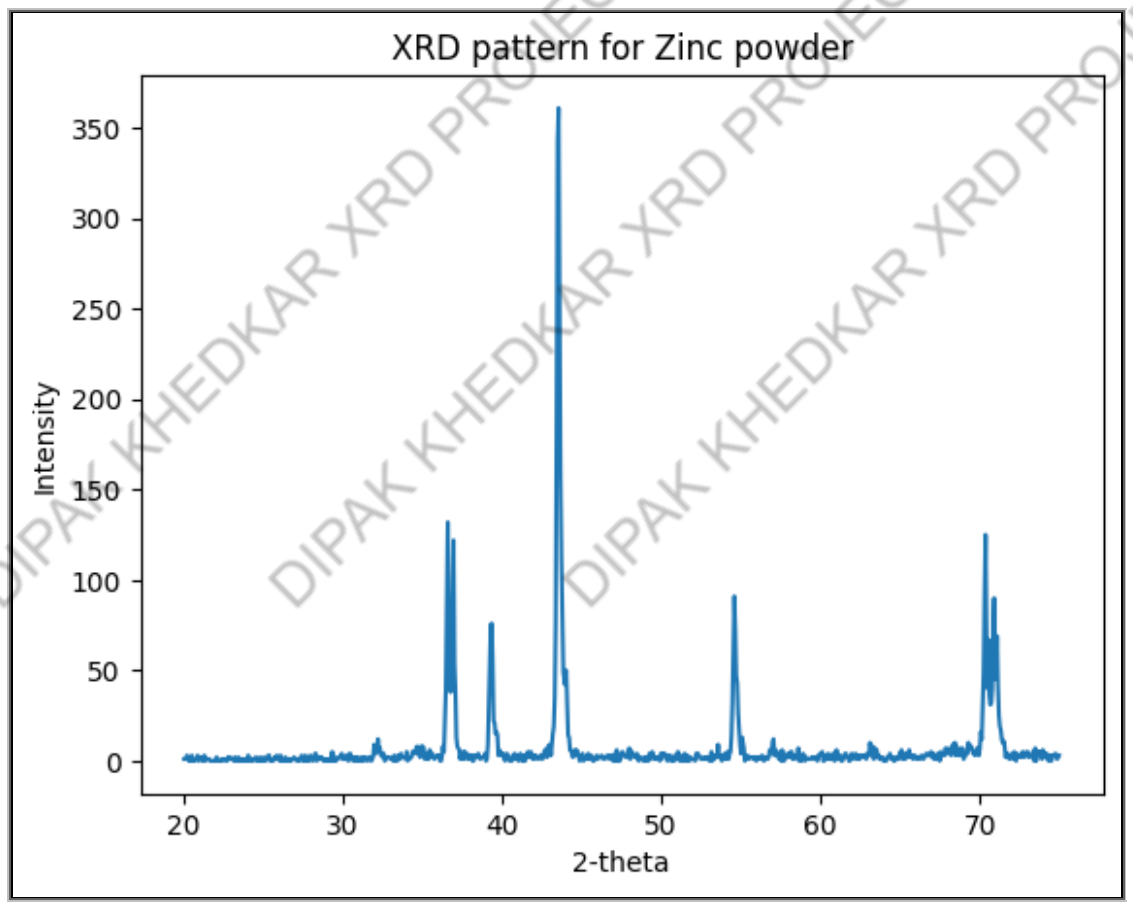

Fig.2.1

2.1.6 Peak detection and save peak data:

Detect the maximum intensity peaks from the above plot here we set the limit of 60 unit for intensity. We mark the detected maximum intensity peaks with * mark which is show in below fig.

```
#to detect the maximum intensity peaks
peaks, _ = find_peaks(data['Intensity (Counts)'], height=60, distance=5)

plt.plot(data['Angle (2theta)'], data['Intensity (Counts)'])
plt.plot(data['Angle (2theta)'][peaks], data['Intensity (Counts)'][peaks], '*')

plt.xlabel('2-theta')
plt.ylabel('Intensity')
plt.show()
```

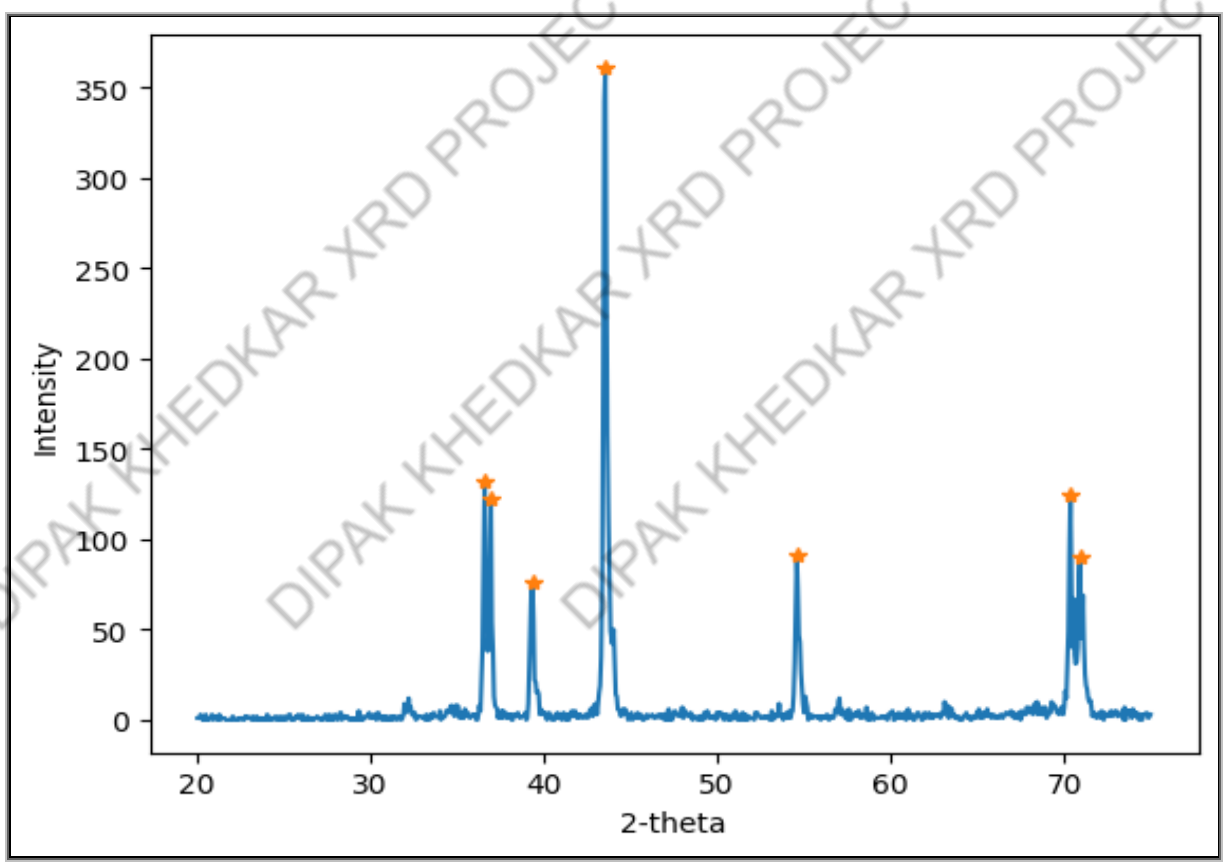

fig.2.2

Save the coordinates of the detected peaks in a new csv file. Read the saved file and access the content as x1=detected_theta and y1=detected_peak intensity for the future operations.

```
peak_coords = np.column_stack((x[peaks], y[peaks])).astype(float)
np.savetxt('peak_file.csv', peak_coords, delimiter=',', header='Theta,Intensity',
comments='')

detected_peak=pd.read_csv('peak_file.csv')
x1=detected_peak['Theta'].tolist()
y1=detected_peak['Intensity'].tolist()
print(detected_peak)
```

Theta	Intensity
36.59	132.0
36.95	122.0
39.35	76.0
43.55	361.0
54.59	91.0
70.37	125.0
70.91	90.0

2.1.7 Read the reference data:

```
reference_data=pd.read_csv('G:\msc project sources\XRD\zinc powder.csv', enc
oding='unicode_escape')
reference_data1 = reference_data.drop(jcpds_data.
  .columns[[2,3,4,5,6,7,8,9,10,11,12,13,15,16,17,18,20,21]], axis=1)

x=reference_data.iloc[:,1].tolist()  #jspds_theta
y=reference_data.iloc[:,15].tolist()  #jcpds intensity
print(reference_data1)
```

No.	Pos. [°2Th.]	d-spacing [Å]	Rel. Int. [%]
1	36.6591	2.44942	24.10
2	36.8004	2.44640	29.86
3	39.3476	2.28803	20.52
4	43.5474	2.07661	100.00
5	54.6254	1.67877	23.00
6	70.3455	1.33722	29.03
7	70.8643	1.32870	23.40

2.1.8 Comparison of reference data with experimental data:

```
plt.stem(x,y,'ro',label='jcpds')
plt.stem(x1,y1,'green',label='Experimental')
# Add 2-theta values at the top of the peak
for i, j in zip(x, y):
    plt.annotate(str(i), xy=(i, j), xytext=(5, 0), textcoords="offset points", ha='left'
, fontsize=8, color='red')
for i, j in zip(x1, y1):
    plt.annotate(str(i), xy=(i, j), xytext=(5, 0), textcoords="offset points", ha='left'
, fontsize=6, color='green')
plt.legend()

# Set x-axis and y-axis limits
plt.xlim(36,72)
plt.ylim(0,400)

plt.xlabel('2-Theta')
plt.ylabel('Intensity')
plt.title('Comparison of jcpds and determined values')
plt.show()
```

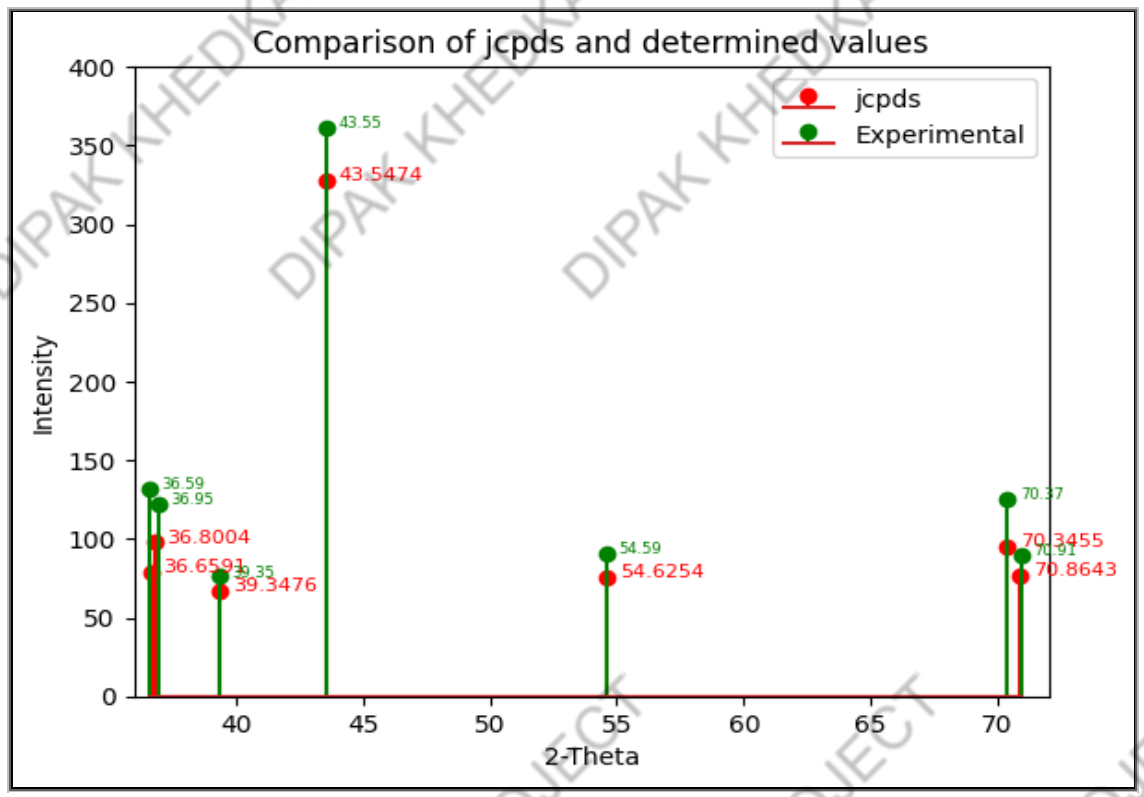

Fig.2.3

2.1.9 Calculate d-spacing:

For each detected theta value we calculate the inter-planer spacing i.e. d-spacing or distance between two consecutive planes. For that we use the bragg's law. It is state that,

$$2d\sin(\Theta)=n\lambda$$

Where, d is inter-planer spacing, Θ is angle of incident, λ is wavelength of x-ray and n is integer. For further calculation we convert theta into radians and take value for wavelength is 1.5406 A^{o} and n is equal to 1. By performing operations for each theta value we get corresponding d-spacing value and its save as new list for further calculations.

```
wavelength = 1.5406  # Cu K-alpha radiation
value=Theta['Theta'].to_numpy()#[36.59, 36.95, 39.35, 43.55, 54.59, 70.37, 70.91]
print("{0} \t {1} ".format("2theta","d-spacing"))
d_list=[ ]
theta_list=[ ]

for i in value:
  theta=i/2
  d_spacing = wavelength / (2 * np.sin(np.deg2rad(theta)))     --#Bragg's law
  d_list.append(d_spacing)
  theta_list.append(i)

d_data={'2-Theta':theta_list,'d-spacing':d_list}
print(d_data)
```

2theta	**d-spacing**
36.59	**2.453891**
36.95	**2.430805**
39.35	**2.287899**
43.55	**2.076489**
54.59	**1.679780**
70.37	**1.336820**
70.91	**1.327958**

2.1.10 Calculate approx. abc values:

From reference data we take corresponding hkl planes for each peak so, we are going to use those hkl values for calculation and perform operations on it and calculate the abc values, for that we use simple cubic lattice formula that is,

$$1/d^2_{hkl} = (h^2+k^2+l^2)/a^2$$

```
hkl_data = ['002', '100', '101', '102', '110', '112', '201']
data1 = {'2theta':theta_list,'d_spacing':d_list,'h': [], 'k': [], 'l': [], '(h^2+k^2+l^2)':
[],'root':[],'abc':abc}

# Iterate over each three-digit number and split it into its individual digits, squa
re each digit, and calculate the sum of the squares
for num in hkl_data:
    h, k, l = str(num)
    h, k, l = int(h), int(k), int(l)
    squared = h**2 + k**2 + l**2
    root=(squared)**(0.5)
    data1['h'].append(h)
    data1['k'].append(k)
    data1['l'].append(l)
    data1['(h^2+k^2+l^2)'].append(squared)
    data1['root'].append(root)

df = pd.DataFrame(data1)
df.to_csv('output.csv', index=False)

d2=df['d_spacing'].tolist()
root=df['root'].tolist()
d2=np.array(d2)
root=np.array(root)
abc=np.multiply(d2,root).tolist()

# highlight abc column
highlight_abc = lambda x: 'background-color: gray'
styled_df = df.style.applymap(highlight_abc, subset=['abc'])

print(styled_df)
```

2theta	d_spacing	h	k	l	(h^2+k^2+l^2)	root	abc
36.590000	2.453891	0	0	2	4	2.000000	**4.907782**
36.950000	2.430805	1	0	0	1	1.000000	**2.430805**
39.350000	2.287899	1	0	1	2	1.414214	**3.235578**
43.550000	2.076489	1	0	2	5	2.236068	**4.643171**
54.590000	1.679780	1	1	0	2	1.414214	**2.375567**
70.370000	1.336820	1	1	2	6	2.449490	**3.274526**
70.910000	1.327958	2	0	1	5	2.236068	**2.969405**

2.1.11 To Calculate FWHM and Grain size (D):

```
xrd_data = data
plt.plot(xrd_data['Angle (2theta)'], xrd_data['Intensity (Counts)'])
plt.xlabel('2-theta (°)')
plt.ylabel('Intensity')

# Find peaks
peaks, _ = find_peaks(xrd_data['Intensity (Counts)'], height=70)

#save fwhm and grain size in new list
FWHM=[ ]
Grain_size_D=[ ]

for peak_index in peaks:
    # Find left and right indices where intensity drops to half of the maximum

    half_max_height = xrd_data['Intensity (Counts)'][peak_index] / 2
    left_index = peak_index - 1
    while xrd_data['Intensity (Counts)'][left_index] > half_max_height:
        left_index -= 1
    right_index = peak_index + 1
    while xrd_data['Intensity (Counts)'][right_index] > half_max_height:
        right_index += 1

    # Calculate FWHM
    fwhm = xrd_data['Angle (2theta)'][right_index] - xrd_data['Angle    (2theta)']
[left_index]
    FWHM.append(fwhm)

    # Calculate grain size using Scherrer's formula
    k = 0.9  --# shape factor
    lambda_ = 1.5406
    theta = xrd_data['Angle (2theta)'][peak_index] * pi / 180
    grain_size = k * lambda_ / (cos(theta) * fwhm) # in nanometers
    Grain_size_D.append(grain_size)
    #print('Grain size:', grain_size, 'nm')

    # Highlight peak on XRD plot
    plt.axvspan(xrd_data['Angle (2theta)'][left_index], xrd_data['Angle (2theta)'][
right_index], alpha=0.2)
```

```
plt.show()
df.insert(loc=2,column='FWHM',value=FWHM)
df.insert(loc=3,column='Grain_size (D)',value=Grain_size_D)
# Define the style for index column
styled_df = df.style.set_properties(subset=pd.IndexSlice[:, ['FWHM','Grain_size (D)']], **{'background-color': 'gray'})
display(styled_df)
```

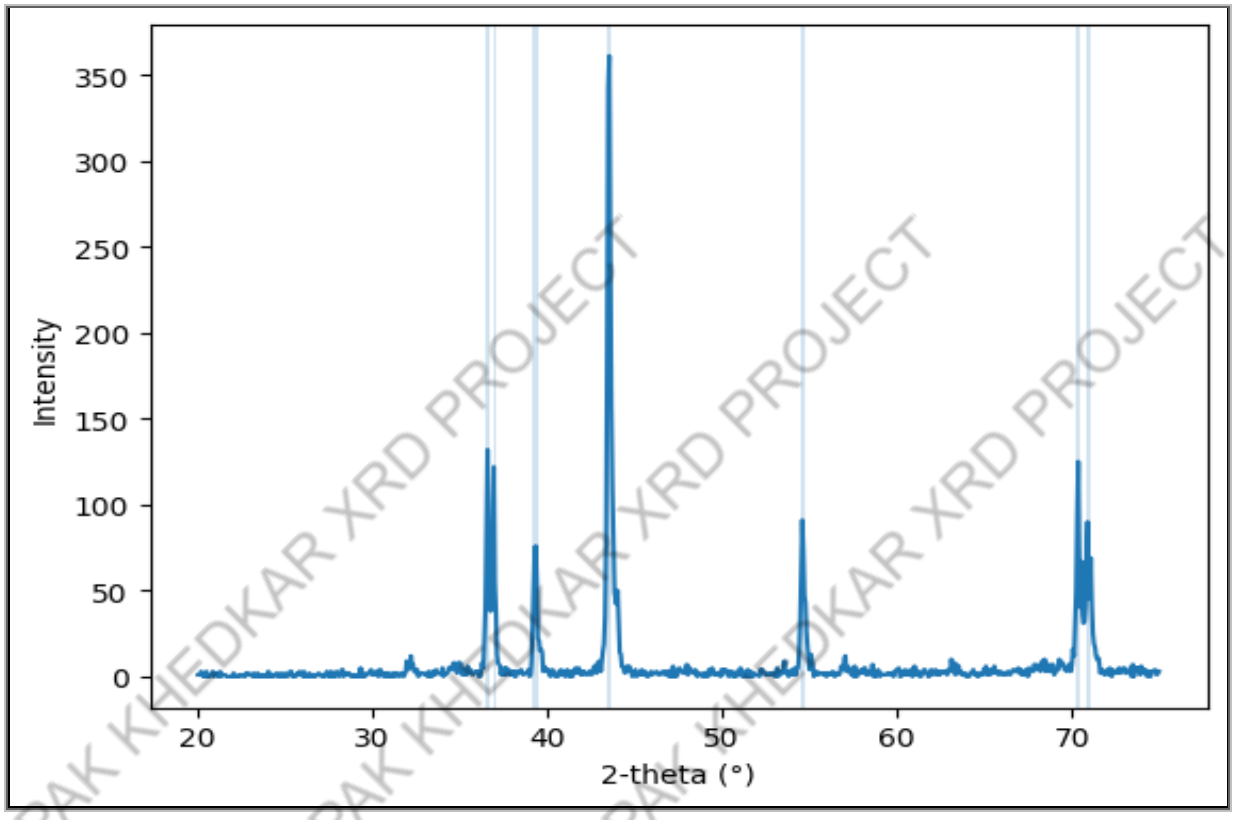

Fig.2.4

2theta	d_spacing	FWHM	Grain_size (D)	h	k	l	(h^2+k^2+l^2)	root	abc
36.59	2.453891	0.24	7.195286	0	0	2	4	2	4.907782
36.95	2.430805	0.12	14.4583	1	0	0	1	1	2.430805
39.35	2.287899	0.3	5.976822	1	0	1	2	1.414214	3.235578
43.55	2.076489	0.3	6.37689	1	0	2	5	2.236068	4.643171
54.59	1.67978	0.3	7.97655	1	1	0	2	1.414214	2.375567
70.37	1.33682	0.24	17.19702	1	1	2	6	2.44949	3.274526
70.91	1.327958	0.24	17.66457	2	0	1	5	2.236068	2.969405

2.1.12 Baseline correction:

```
import pandas as pd

# Read the CSV file and store it in a dataframe
df = pd.read_csv(r'C:\Users\HP\Downloads\bfo_xrd1.csv')
```

2-theta	intensity
20.02	286
20.04	281
20.06	276
20.08	305
20.10	302 ...

```
# Load the XRD data from a CSV file
xrd_data = data
x = data.iloc[:,0]
y = data.iloc[:,1]

# Apply the Savitzky-Golay filter to smooth the data and estimate the baseline
baseline = savgol_filter(y, 100, 2)

# Subtract the baseline from the original data to obtain the adjusted data
adjusted_data = y - baseline

# Save the adjusted data to a new CSV file
adjusted_data_df = pd.DataFrame({'X': x, 'Y': adjusted_data})
adjusted_data_df.to_csv('adjusted_xrd_data.csv', index=False)

base_data=pd.read_csv(r'C:\Users\HP\Downloads\adjusted_xrd_data.csv')
base_data
```

X	Y
20.02	-31.074438
20.04	-36.611751
20.06	-42.138685
20.08	-13.655242
20.10	-17.161420...

```
# Apply Savitzky-Golay filter to smooth the data
y_smooth = savgol_filter(y, window_length=21, polyorder=2)

# Subtract the smoothed signal from the original signal to obtain the baseline
baseline = y - y_smooth
# Plot the original signal and the baseline-corrected signal
import matplotlib.pyplot as plt
plt.plot(x, y, label='Original signal')
plt.plot(x, y_smooth, label='Smoothed signal')
plt.plot(x, baseline, label='Baseline')
plt.legend()
plt.show()

# Save the baseline-corrected data to a new CSV file
data['Baseline'] = baseline
data['y_smooth']=y_smooth
data.to_csv('xrd_data_baseline_corrected.csv', index=False)
peaks, _ = find_peaks(y, height=230, distance=50)
baseline = savgol_filter(y, window_length=51, polyorder=3)
y_flat = y – baseline
```

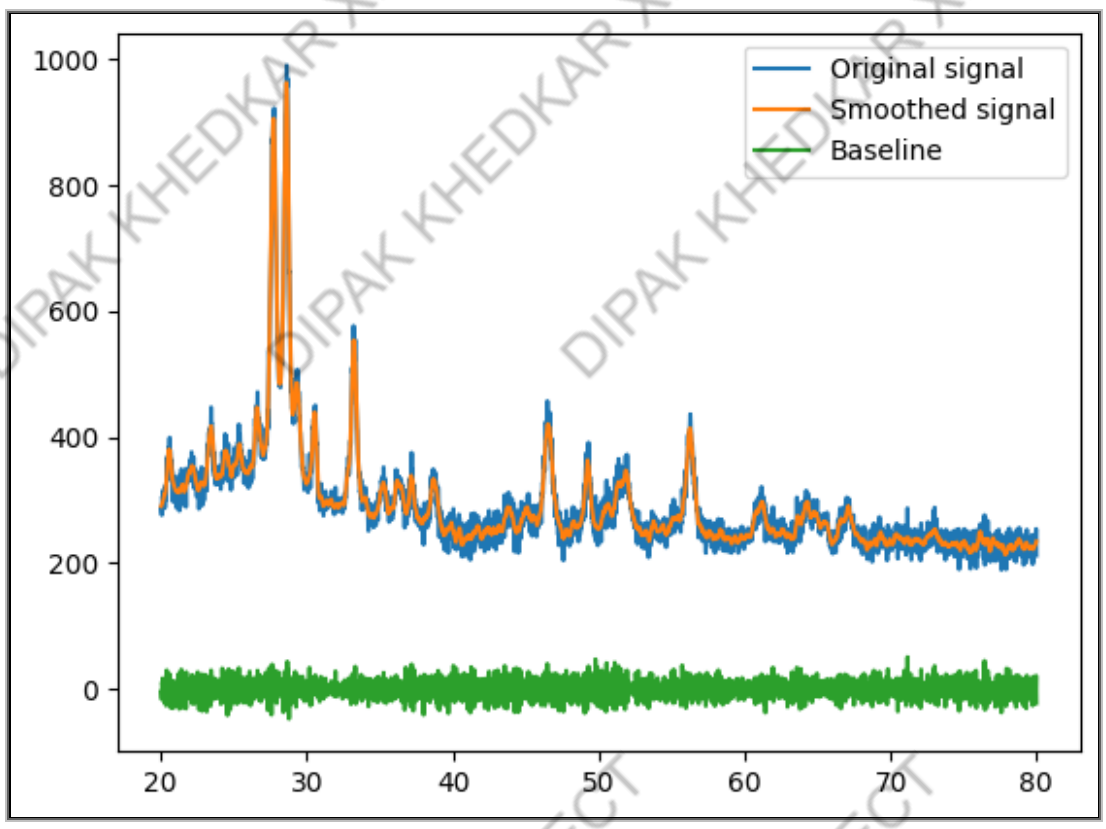

Fig.2. 5

2.1.13 Data plot after correction of baseline

```
plt.plot(x, y_smooth, label='Smoothed signal')
plt.title('xrd plot after baseline correction1')
plt.xlabel("2Theta")
plt.ylabel("Intensity")
plt.show()


baseline = savgol_filter(y_smooth, window_length=250, polyorder=2)
y_data_corrected = y_smooth – baseline
```

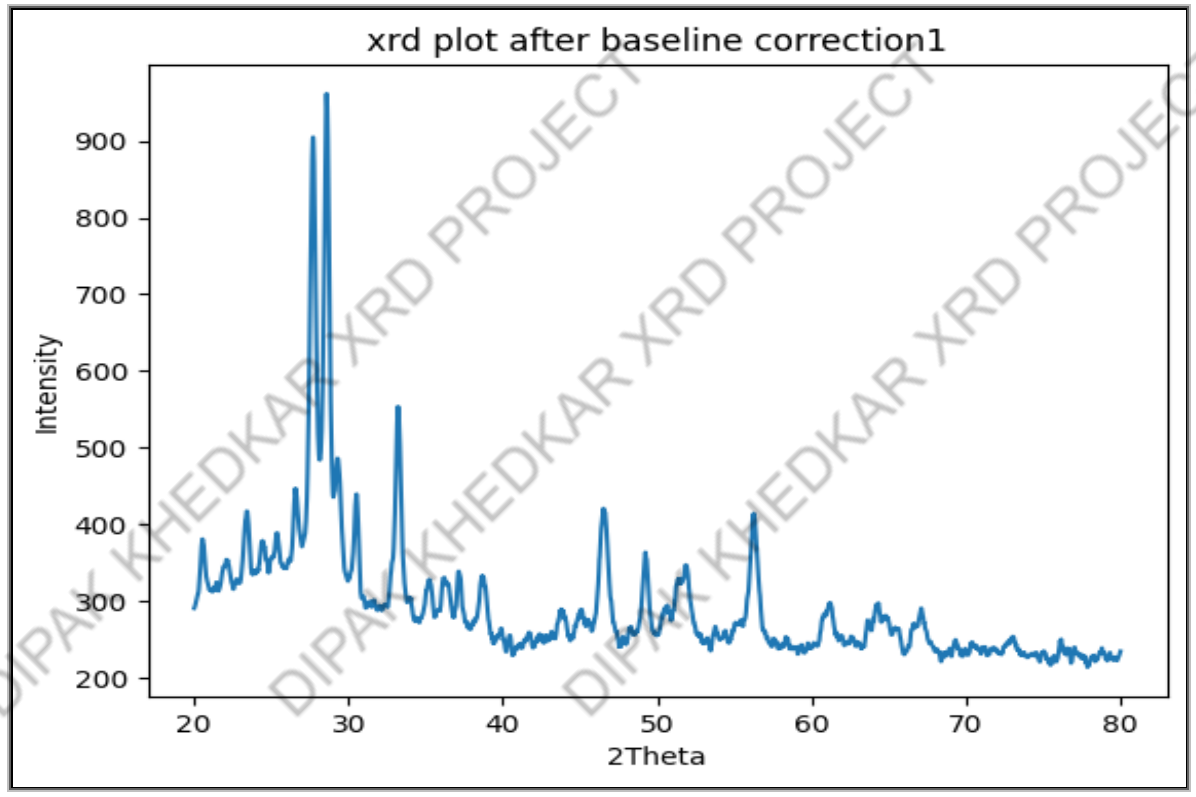

Fig.2.6

2.2 Pymatgen Application

2.2.1 Install pymatgen:

```
!pip install pymatgen
```

2.2.2 Access material surface properties using api_key:

```
from mp_api.client import MPRester

with MPRester(api_key='ygczzVeeUPfPlnBGRhMxZ0dRZWcX75df') as mpr:
    surface_properties_doc = mpr.surface_properties.get_data_by_id('mp-79')

surface_properties_doc
```

surface_energy_EV_PER_ANG2= 0.049858712834864614,
surface_energy= 0.7988246101579123
work_function= 4.155235970301791
efermi= -0.5666
area_fraction= 0.0
surface_anisotropy= 0.21740376274172288,
pretty_formula= 'Zn',
shape_factor= 5.85949104225898,
task_id= 'mp-79',

2.2.3 Get structure details:

```
mpr.get_structures
structure
```

Structure Summary
Lattice
abc : 2.626730472467679 2.626730472467679 5.207234
angles : 90.0 90.0 120.00000000000001
volume : 31.114924432303443
PeriodicSite: Zn (1.3134, -0.7583, 3.9054) [0.6667, 0.3333, 0.7500]
PeriodicSite: Zn (1.3134, 0.7583, 1.3018) [0.3333, 0.6667, 0.2500],

We can create lattice:

Here we provide the structure parameter for the crystal and from that we will gate the volume for crystal corresponding to given parameter.

```
from pymatgen.core import Lattice, Structure

Lattice_1=Lattice.from_parameters(5,5,5,90,90,90)

Lattice_1
```

Lattice
abc : 5.0 5.0 5.0
angles : 90.0 90.0 90.0
volume : 125.0

2.2.4 Create fcc crystal:

Here we set the crystal parameter for cubic structure is 2.64 and provide the coordinates of atoms. From that we get lattice details as shown below:

```
bcc_fe=Structure(Lattice.cubic(2.64),['Fe','Fe'],[[0,0,0],[1,1,1]])

bcc_fe
```

Structure Summary
Lattice
abc : 2.64 2.64 2.64
angles : 90.0 90.0 90.0
volume : 18.399744000000002

PeriodicSite: Fe (0.0000, 0.0000, 0.0000) [0.0000, 0.0000, 0.0000]
PeriodicSite: Fe (2.6400, 2.6400, 2.6400) [1.0000, 1.0000, 1.0000]

```
bcc_fe.volume
```

18.399744000000002

```
bcc_fe.atomic_numbers
```

(26, 26)

```
bcc_fe.composition
```

Comp: Fe2

2.2.5 Create NaCl lattice from spacegroup:

```
nacl1=Structure.from_spacegroup('Fm-3m',Lattice.cubic(5.6),['Na+','Cl-'],[[0,0,0],[0.5,0.5,0.5]])

print(nacl1)
```

Full Formula: (Na4 Cl4)
Reduced Formula: NaCl
abc : 5.600000 5.600000 5.600000
angles: 90.000000 90.000000 90.000000

Sites (8)

#	SP	a	b	c
0	Na+	0	0	0
1	Na+	0.5	0.5	0
2	Na+	0.5	0	0.5
3	Na+	0	0.5	0.5
4	Cl-	0.5	0.5	0.5
5	Cl-	0	0	0.5
6	Cl-	0	0.5	0
7	Cl-	0.5	0	0

2.2.6 We get crystal lattice by providing composition:

```
composition={'Cu':0.5,'Au':0.5}
cu_au=Structure.from_spacegroup('Fm-3m',Lattice.cubic(4.677),[composition],[[0,0,0]])

cu_au
```

Structure Summary
Lattice
abc : 4.677 4.677 4.677
angles : 90.0 90.0 90.0
volume : 102.30623673299998

2.2.7 Replace the Na site with 'I' element at 0 position:

```
nacl1.replace(0,'I ')

print(nacl1)
```

Structure Summary
Lattice
abc : 5.6 5.6 5.6
angles : 90.0 90.0 90.0
volume : 175.61599999999996

2.2.8 We dislocate the position of I atom in lattice and get the site info.

```
nacl1.translate_sites([0],[0,0.1,0])
nacl1
```

Structure Summary
Lattice
abc : 5.6 5.6 5.6
angles : 90.0 90.0 90.0
volume : 175.61599999999996
PeriodicSite: I (0.0000, 1.1200, 0.0000) [0.0000, 0.2000, 0.0000]
PeriodicSite: Na+ (2.8000, 2.8000, 0.0000) [0.5000, 0.5000, 0.0000]
PeriodicSite: Na+ (2.8000, 0.0000, 2.8000) [0.5000, 0.0000, 0.5000]
PeriodicSite: Na+ (0.0000, 2.8000, 2.8000) [0.0000, 0.5000, 0.5000]
PeriodicSite: Cl- (2.8000, 2.8000, 2.8000) [0.5000, 0.5000, 0.5000]
PeriodicSite: Cl- (0.0000, 0.0000, 2.8000) [0.0000, 0.0000, 0.5000]
PeriodicSite: Cl- (0.0000, 2.8000, 0.0000) [0.0000, 0.5000, 0.0000]
PeriodicSite: Cl- (2.8000, 0.0000, 0.0000) [0.5000, 0.0000, 0.0000]

CHAPTER: 3

Result and discussion

3.1 Peak Detection:

We perform certain operations and detect the maximum intensity peak from which we get the accurate 2-Theta values for each detected peak.as shown in fig.2

Theta values for each detected peak are as follows:

2-Theta	Intensity
36.59	132.0
36.95	122.0
39.35	76.0
43.55	361.0
54.59	91.0
70.37	125.0
***70*.91**	90.*0*

3.2 Determine the d-spacing:

Now we already have required 2-Theta values, from that theta values we calculate the inter-planer spacing by using Braggs law, from that we get the corresponding d-spacing values for each detected peak are as follows:

2-Theta	d-spacing
36.59	**2.453891**
36.95	**2.430805**
39.35	**2.287899**
43.55	***2*.076489**
54.59	**1.679780**
70.37	**1.336820**
70.91	**1.327958**

3.3 Comparison of Reference values and determine values:

We determined the corresponding 2-Theta values for each peak, now here we compared our determined values with reference values as shown in fig.3

Ref. 2-Theta	Determined 2-Theta
36.6591	36.59
36.8004	36.95
39.3476	39.35
43.5474	43.55
54.6254	54.59
70.3455	70.37
70.8643	*70.*91

3.4 Determine Grain size:

Now here we use the scherrer formula to calculate the grain size (D). for that we calculate the FWHM for the each peak and by putting those values in formula we get the grain size of zinc powder. We get the maximum grain size is 14.4583 nm. And minimum grain size is 5.9765 nm. The values are as follows:

FWHM	Grain_size (D)
0.24	7.195286
0.12	14.4583
0.3	5.976822
0.3	6.37689
0.3	7.97655
0.24	17.19702
0.24	17.66457

3.5 Determined crystal structure parameters:

Here we try to calculate the a,b,c values for crystal lattice, but we are unable to calculate it for hexagonal lattice. The expected values are a=b=2.64 nm. And c=4.91 nm. But we get an error by using cubic lattice formula we calculate some approximate a,b,c values are as follows:

a,b,c
4.907782
2.430805
3.235578
4.643171
2.375567
3.274526
2.969405

3.6 Use of pymatgen:

We us pymatgen for crystal lattice modification, in which we create the crystal lattice by providing composition, element, structure type and also rearrange and manipulate the position of atom in the lattice, we also planned for visualization of crystal lattice but there is certain machine errors occurring. By creating crystal structures in pymatgen helps to analyzed the crystal properties such as volume, band gap, surface energy, etc..

CHAPTER: 4

Conclusion

In this project we work on the xrd data analysis from which we successfully analyzed xrd data and calculate the required parameters of material which will required for material characterization such as Grain size, d-spacing, etc. by using programing we can reduce the dependency on various analysis software's which was time consuming and there is lots of human work is need to done and which leads to the human errors but by using programing skills we can perform the analysis on data its quit less time consuming and having high accuracy.

We also try same method for uv-visible analysis but there is certain limitations and thet we need to work on.

Also, the pymatgen is providing us the great source of reference data and properties of material and operations on the crystal lattice. There is lots of work need to be done in order to use pymatgen for material characterization.

CHAPTER: 5

Reference

1. *https://pandas.pydata.org/getting_started.html*
2. *https://numpy.org/doc/1.24*
3. *https://numpy.org/doc/1.23*
4. *https://pymatgen.org/*
5. *https://github.com/materialsproject/pymatgen*
6. *https://workshop.materialsproject.org/lessons/02_intro_pymatgen/1%20-%20pymatgen%20foundations/*
7. *https://notebook.community/PhasesResearchLab/prl-onboard/python-course/assignment-3/assignment-3-solution*
8. *https://www.youtube.com/watch?v=S7rF7abLK9E*
9. *https://www.youtube.com/watch?v=PSsNcvzJAqk*
10. *https://www.youtube.com/watch?v=hDkMjrNZKsw*