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

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

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  
.. ... ...

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')

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()

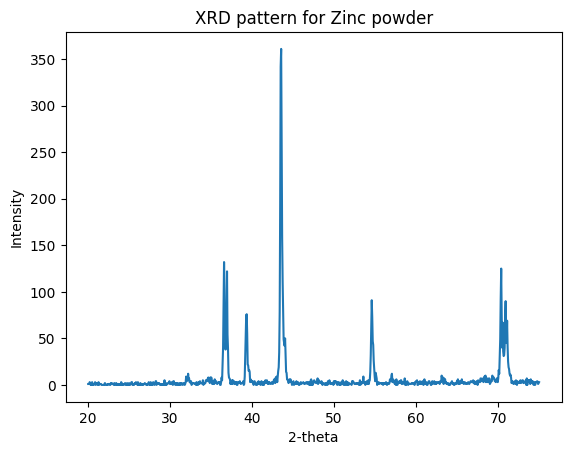


Fig.1  
  
  
  
  
  
  
Detect the maximum intensity peaks from the above plot here we set the limit of 60 unit for intensity. We mark the detected maximum intensity peaks with \* mark which is show in below fig.

#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()

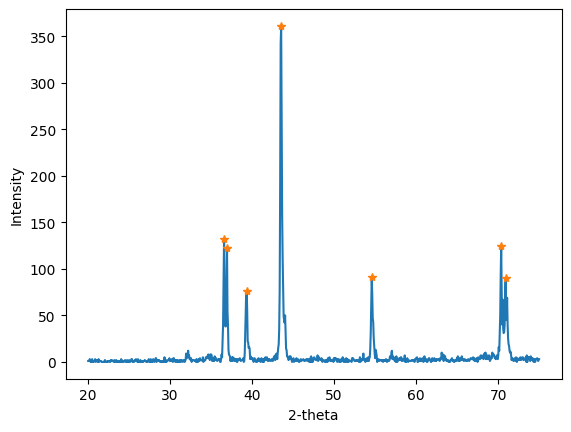


fig.2

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

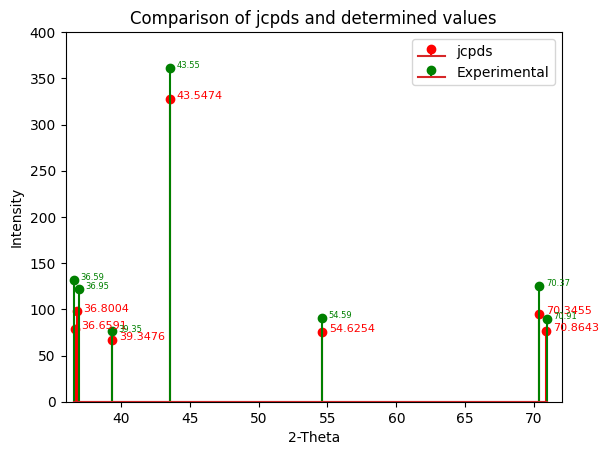
Read the reference data:

reference\_data=pd.read\_csv('G:\msc project sources\XRD\zinc powder.csv', encoding='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

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()



Calculation for 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,

**2dsin(Ɵ)=nƛ**

Where, d is inter-planer spacing, **Ɵ** is angle of incident, **ƛ** 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 Ao 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

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/d2hkl = (h2+k2+l2)/a2**

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, square 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** | |

**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)

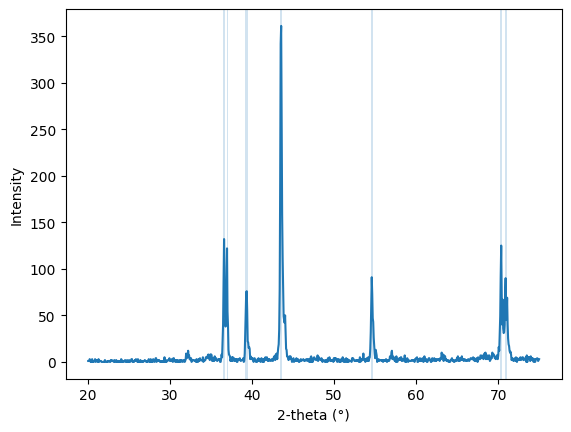


Fig.



**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')

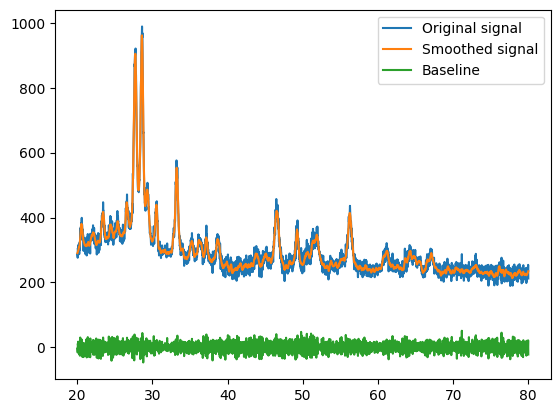
2-theta intensity  
 -------- --------  
 20.02 286  
 20.04 281  
 20.06 276  
 20.08 305  
 20.10 302  
 ... ... ...  
 [3000 rows x 2 columns]

# Load the XRD data from a CSV file  
xrd\_data = data  
x = data.iloc[:,0]  
y = data.iloc[:,1]  
  
# Extract the X and Y data from the CSV file  
  
# 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  
 ... ... ...  
[3000 rows x 2 columns]

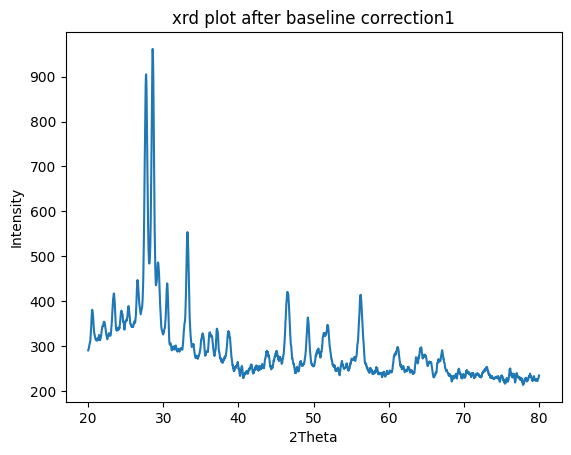
# Apply Savitzky-Golay filter to smooth the data  
y\_smooth = savgol\_filter(y, window\_length=21, polyorder=2)  
  
# Subtract the smoothed signal from the original signal to obtain the baseline  
baseline = y - y\_smooth  
  
# Plot the original signal and the baseline-corrected signal  
import matplotlib.pyplot 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.**

**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



**Pymatgen**:

!pip install pymatgen

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   
weighted\_surface\_energy\_EV\_PER\_ANG2=0.027623588865645848,  
weighted\_surface\_energy=0.44257866583614763,  
surface\_anisotropy=0.21740376274172288,  
pretty\_formula='Zn',  
shape\_factor=5.85949104225898,  
task\_id='mp-79',  
structure=Structure Summary

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],

from pymatgen.core import Lattice, Structure

We can create latttice

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

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

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

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

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

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]