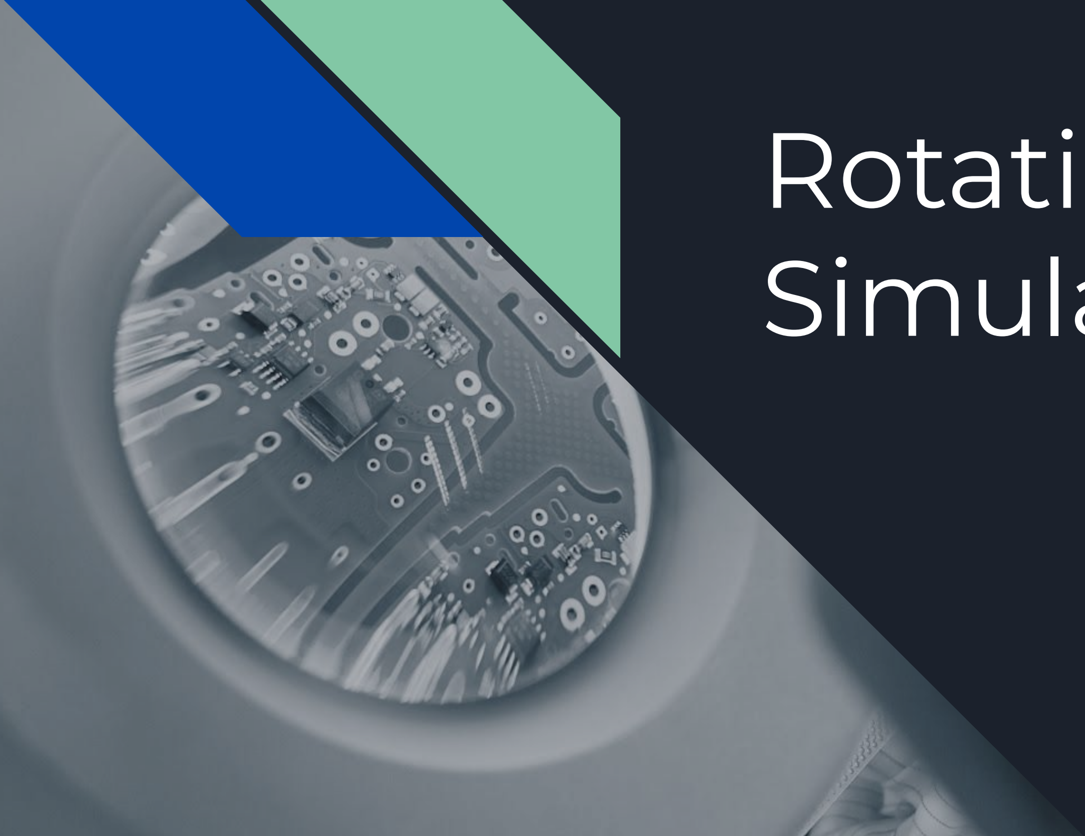
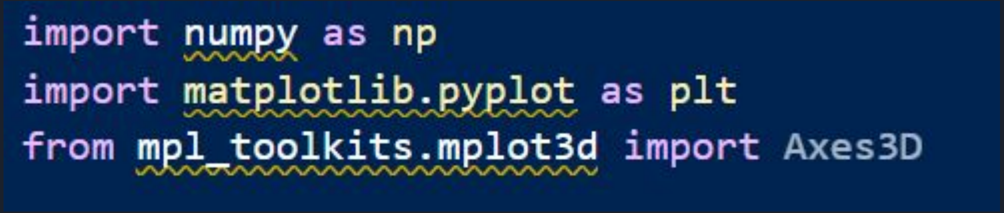


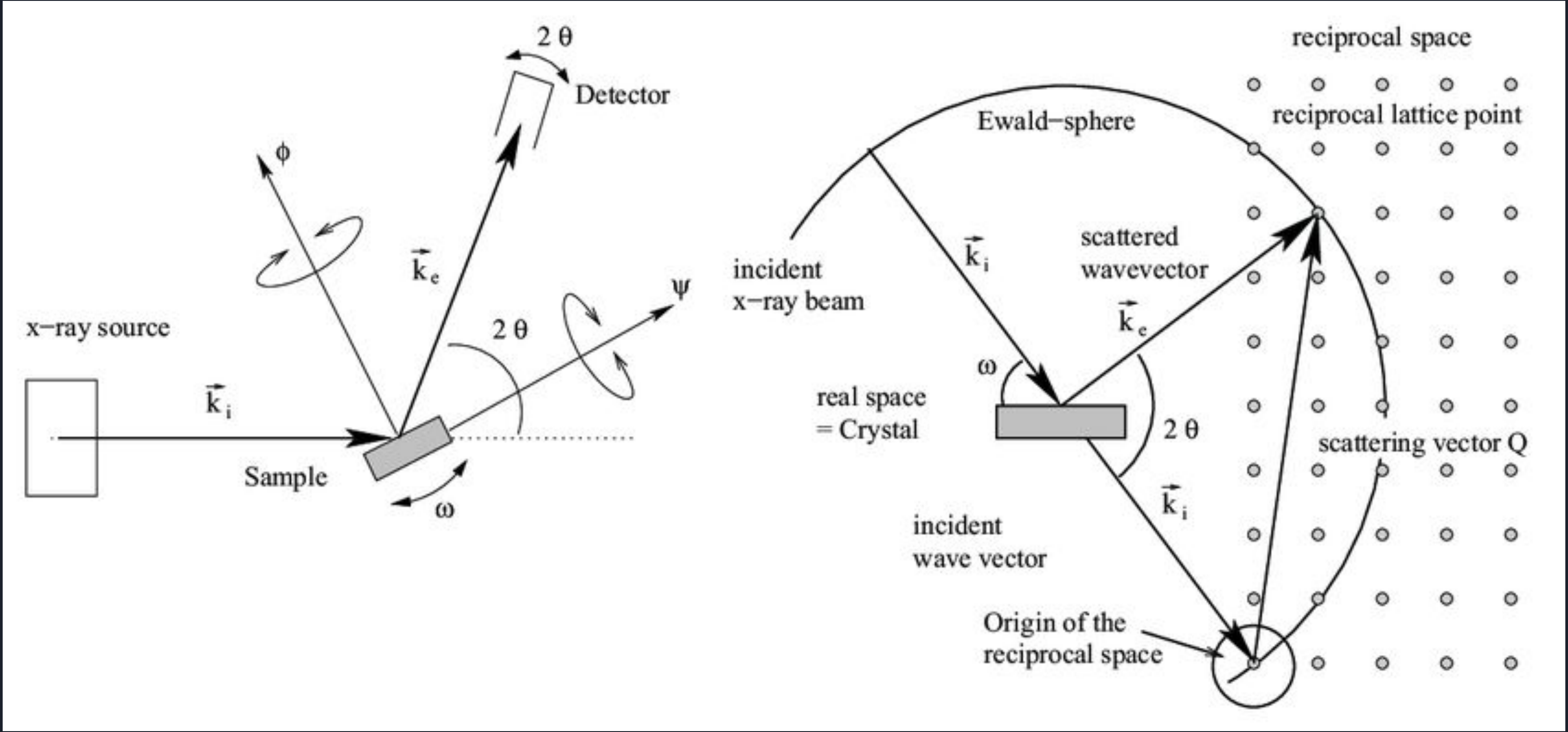
Rotating Crystal Simulation

Abdallah W. Mahmoud

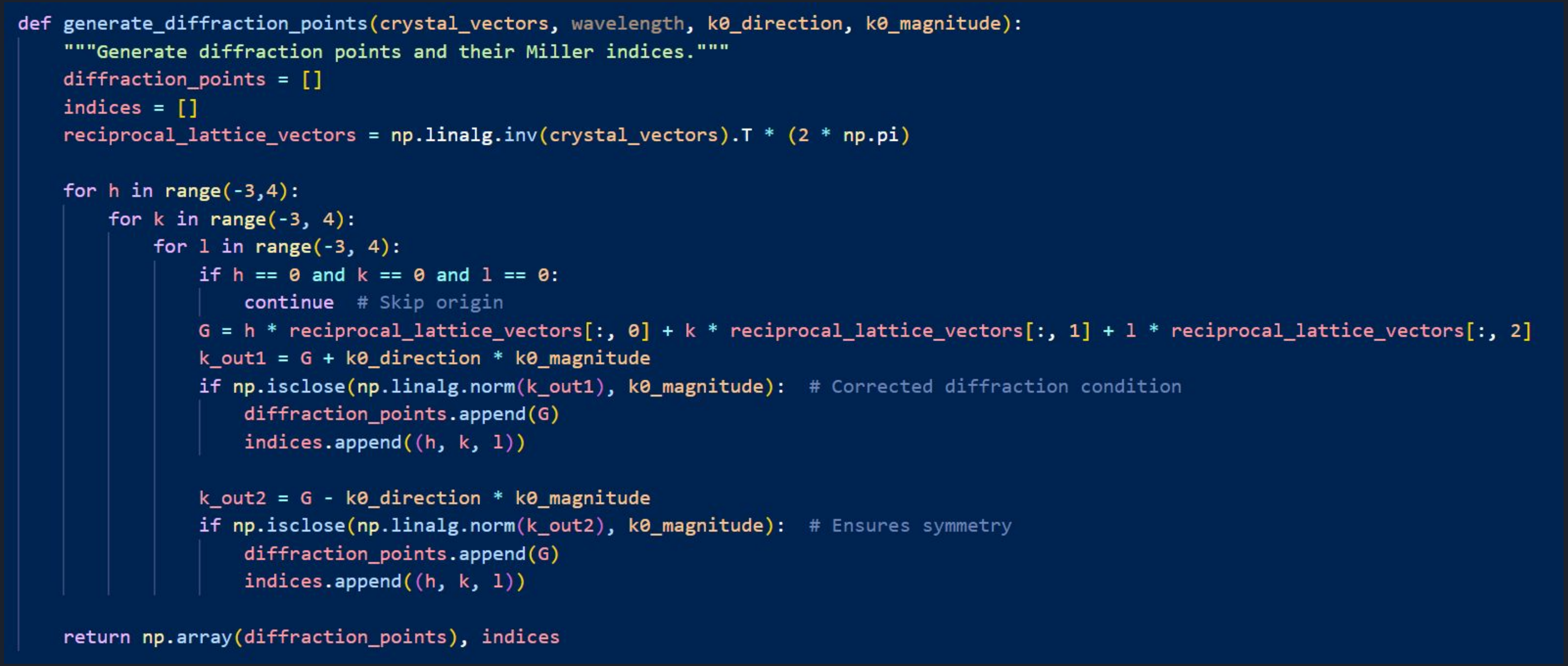
Important libraries



```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```


2 θ
Detector
φ
$\vec{k}_e$
ψ
x−ray source
2 θ
$\vec{k}_i$
Sample
ω
reciprocal space
reciprocal lattice point
Ewald−sphere
incident
x−ray beam
$\vec{k}_i$
scattered
wavevector
$\vec{k}_e$
ω
real space
= Crystal
2 θ
scattering vector Q
$\vec{k}_i$
incident
wave vector
Origin of the
reciprocal space

The module for generating the diffraction points



```python
def generate_diffraction_points(crystal_vectors, wavelength, k0_direction, k0_magnitude):
    """Generate diffraction points and their Miller indices."""
    diffraction_points = []
    indices = []
    reciprocal_lattice_vectors = np.linalg.inv(crystal_vectors).T * (2 * np.pi)

    for h in range(-3,4):
        for k in range(-3, 4):
            for l in range(-3, 4):
                if h == 0 and k == 0 and l == 0:
                    continue  # Skip origin
                G = h * reciprocal_lattice_vectors[:, 0] + k * reciprocal_lattice_vectors[:, 1] + l * reciprocal_lattice_vectors[:, 2]
                k_out1 = G + k0_direction * k0_magnitude
                if np.isclose(np.linalg.norm(k_out1), k0_magnitude):  # Corrected diffraction condition
                    diffraction_points.append(G)
                    indices.append((h, k, l))

                k_out2 = G - k0_direction * k0_magnitude
                if np.isclose(np.linalg.norm(k_out2), k0_magnitude):  # Ensures symmetry
                    diffraction_points.append(G)
                    indices.append((h, k, l))

    return np.array(diffraction_points), indices
```

np.isclose(a,b)

Checks if a is approximately equal to b

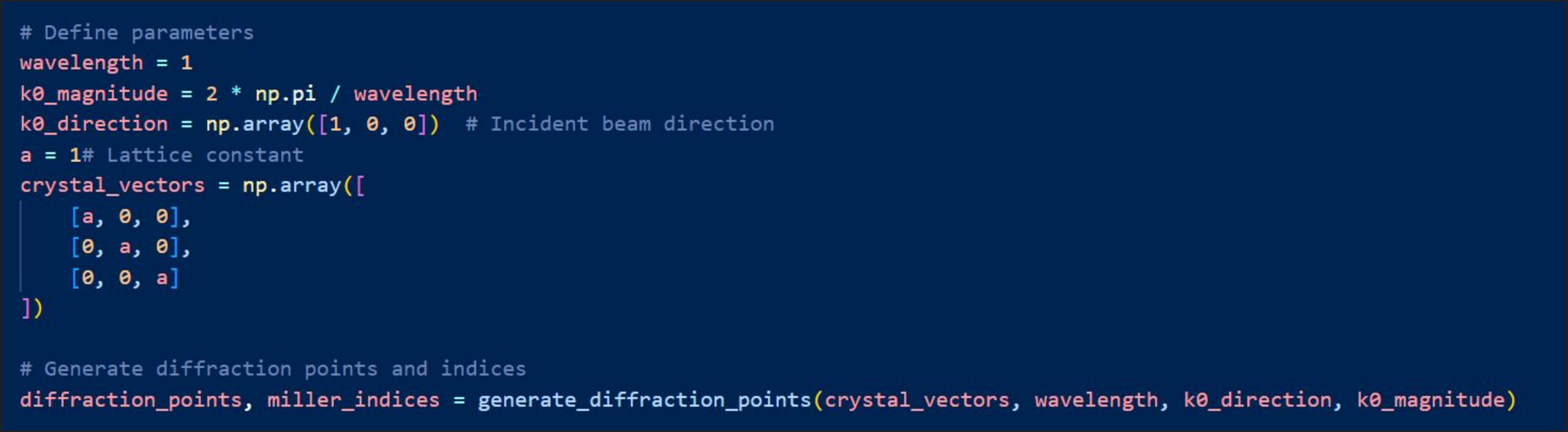
Np.linalg.norm(a)

Calculates the magnitude of a vector a

np.linalg.inv(a)

Calculates the transpose of the inverse of a vector a

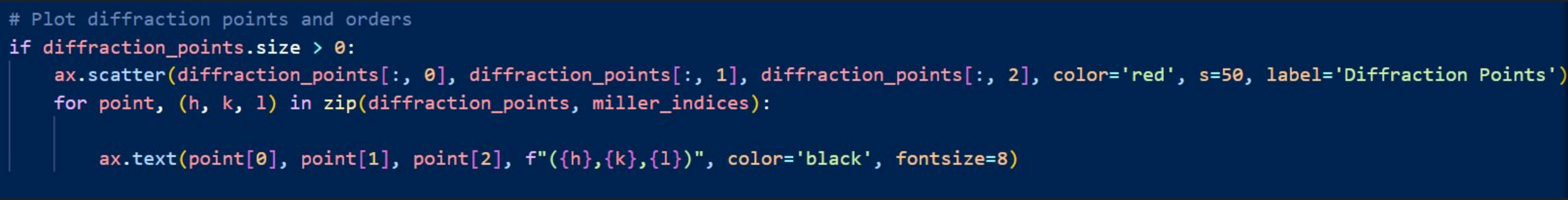
Defining the parameters



```python
# Define parameters
wavelength = 1
k0_magnitude = 2 * np.pi / wavelength
k0_direction = np.array([1, 0, 0])  # Incident beam direction
a = 1# Lattice constant
crystal_vectors = np.array([
    [a, 0, 0],
    [0, a, 0],
    [0, 0, a]
])

# Generate diffraction points and indices
diffraction_points, miller_indices = generate_diffraction_points(crystal_vectors, wavelength, k0_direction, k0_magnitude)
```

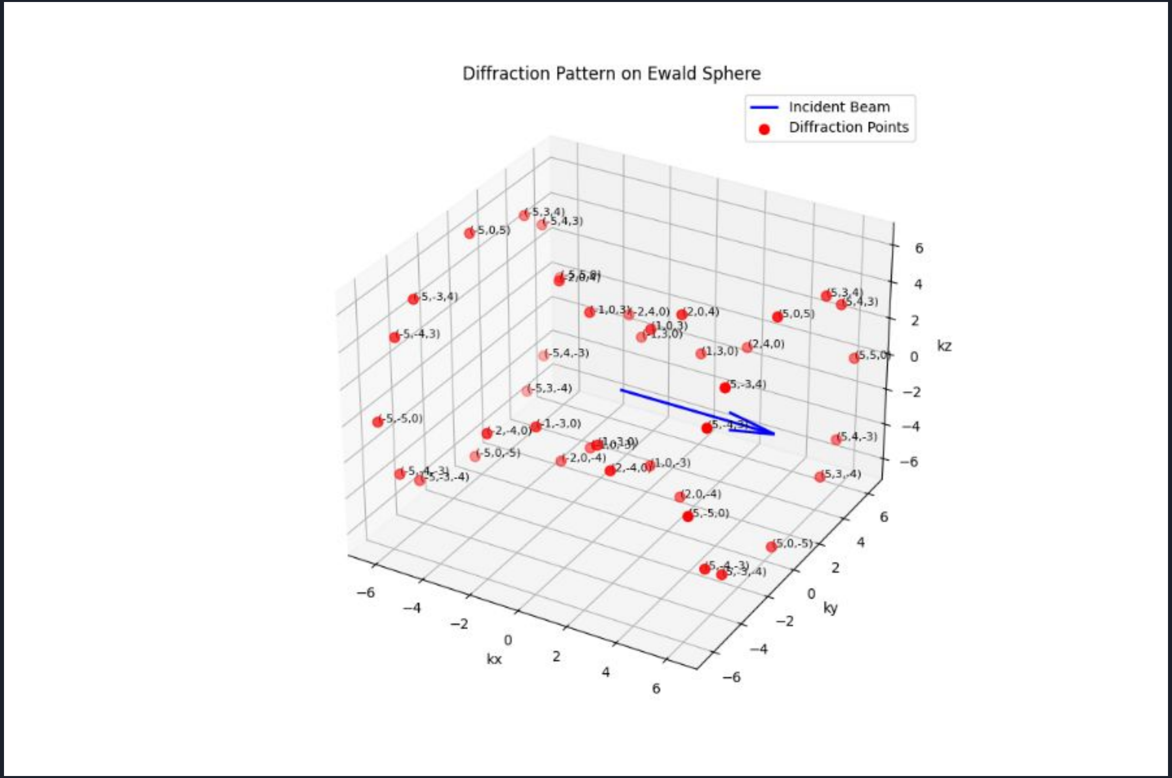
We can easily generalise to other non-cubic structures

Plotting the diffraction points

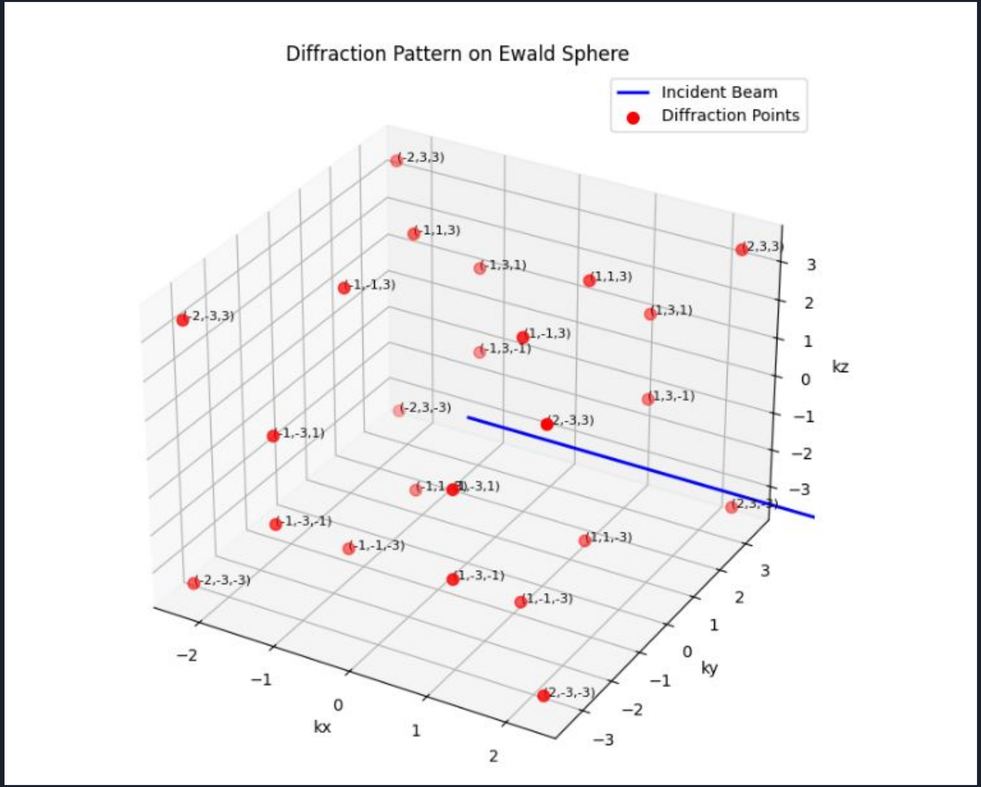


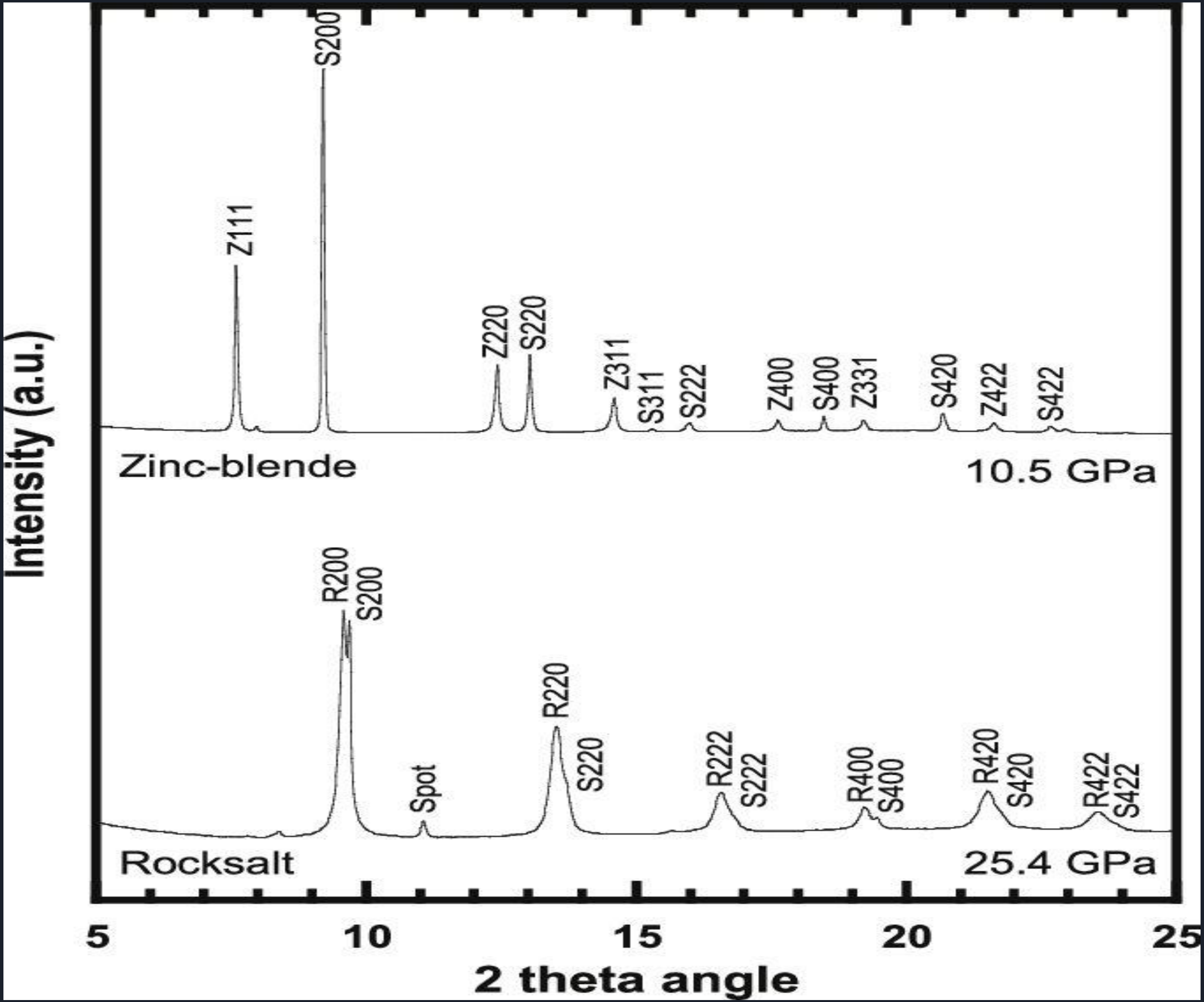
```
# Plot diffraction points and orders
if diffraction_points.size > 0:
    ax.scatter(diffraction_points[:, 0], diffraction_points[:, 1], diffraction_points[:, 2], color='red', s=50, label='Diffraction Points')
    for point, (h, k, l) in zip(diffraction_points, miller_indices):

        ax.text(point[0], point[1], point[2], f"({h},{k},{l})", color='black', fontsize=8)
```

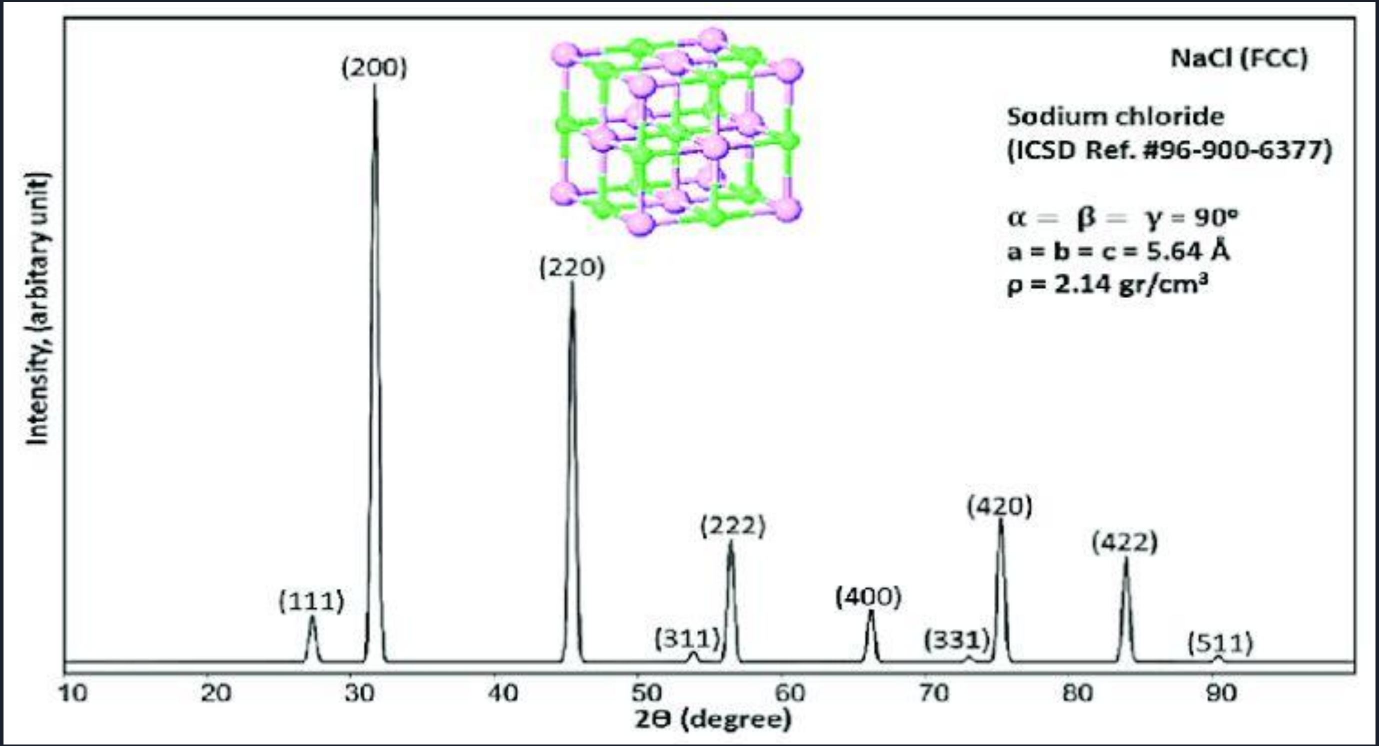
Zinc Blende → a=5



Sodium chloride → a=5.5




Intensity (a.u.)
Z111
S200
Z220
S220
Z311
S311
S222
Z400
S400
Z331
S420
Z422
S422
Zinc-blende
10.5 GPa
R200
S200
Spot
R220
S220
R222
S222
R400
S400
R420
S420
R422
S422
Rocksalt
25.4 GPa
5
10
15
20
25
2 theta angle



NaCl (FCC)
Sodium chloride
(ICSD Ref. #96-900-6377)
α = β = γ = 90°
a = b = c = 5.64 Å
ρ = 2.14 gr/cm³
Intensity, (arbitary unit)
2Θ (degree)
(111)
(200)
(220)
(311)
(222)
(400)
(331)
(420)
(422)
(511)
10
20
30
40
50
60
70
80
90

Thank you!