1　背景と目的

原子構造を決定する事は物性を理解する根本的なプロセスである。本課題では、X 線回折による結晶構造の決定の仕組みを体験するとともに、逆格子の概念やフーリエ変換を理解することを目的とする。

2　実験原理

2.1　ブラッグの公式

ブラッグの公式は n を自然数としたとき、以下の式で与えられる。

$$2d\sin\theta = n\lambda$$

これは X 線の波に対して原子配列は回折結晶の役割を果たし、各原子からの散乱波が強め合う条件を表している。強度の強い反射は n=1 の時だけである。

2.2　距離 d と面指数の関係

結晶面の面指数が (hkl) であり、格子定数が a であるとき、面間隔 d は以下の式で与えられる。

$$d = \frac{a}{\sqrt{h^2+k^2+l^2}}$$

2.3　フーリエ変換

電子密度マップを求めるためのフーリエ変換の公式は以下の式で与えられる。

$$\rho(\vec{r}) = \frac{1}{(2\pi)^3}\int d\vec{q}F\vec{q}\exp(-i\vec{q}\cdot\vec{r})$$

ここで逆格子の概念を導入すると、以下のように書き換えることができる。

$$\rho(\vec{r}) = \sum F(hkl)\exp\left(-2\pi i(hx+ky+lz)\right)$$

今回の実験において使用したコードは巻末に記載する。

3　実験方法

3.1　実験試料

1. NaCl 粉末
2. KCl 粉末
3. Al 粉末

3.2　実験装置

- 乳鉢・乳棒
- 試料ホルダー
- X 線発生装置
- イメージングプレート (IP)
- 粉末回折ホルダー
- ビームキャプチャー
- コリメータ
- X 線管球部

3.3　解析ソフトウェア

- TRY-IP-READER
- TRY-VIEWER
- Python 3.6.1

3.4　実験手順

(1) 実験試料を丹念にすり潰し、試料ホルダーを作成した。

(2) X 線回折装置を用いて IP に X 線強度を記録した。

(3) TRY-IP-READER を用いて IP を読み取った。

(4) TRY-VIEWER を用いてデータを書き出した。

(5) Python を用いて結晶構造を求めた。ソースコードは巻末に記載した。

4　実験結果

4.1　実験 1 NaCl 結晶構造の特定

2θ-強度グラフを図 4.1、各データの一覧を表 1、z=0,z=0.5 における電子密度マップを図 2 に示した。

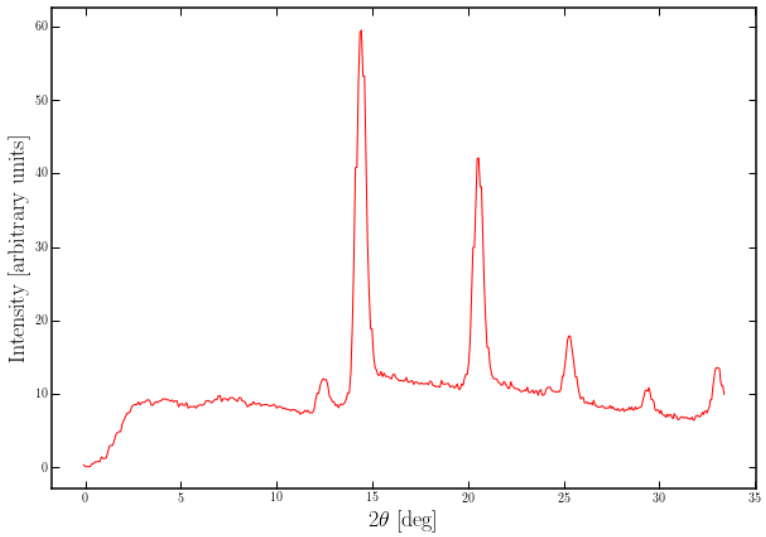
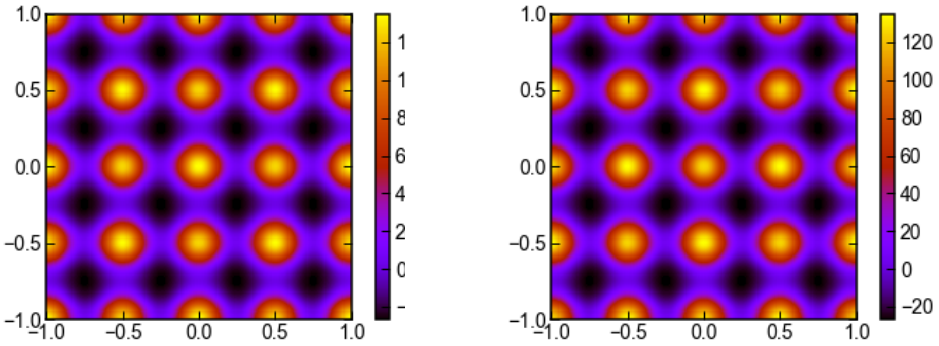


図 1: 2θ-強度グラフ

表 1: NaCl における各データの一覧

hkl	111	200	220	222	400	420
2θ	12.584072	14.540293	20.630895	25.354746	29.373919	32.965019
distance	3.250482	2.815000	1.990506	1.625241	1.407500	1.258906
I(hkl)	2.483306	646.186757	164.468498	9.917789	1.469235	6.885137



[1]z=0　　[2]z=0.5

図 2: 電子密度マップ

4.2　実験 2 KCl 結晶構造の特定

2θ-強度グラフを図 4.2、各データの一覧を表 2、z=0,z=0.5 における電子密度マップを図 4 に示した。

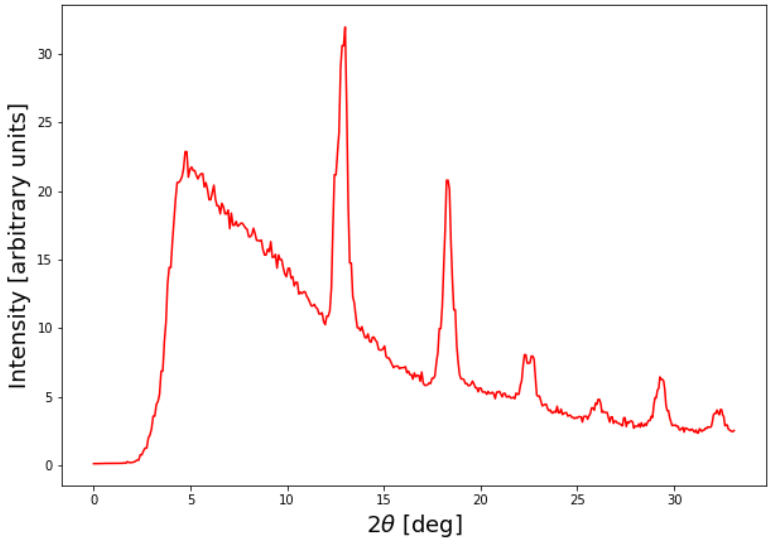


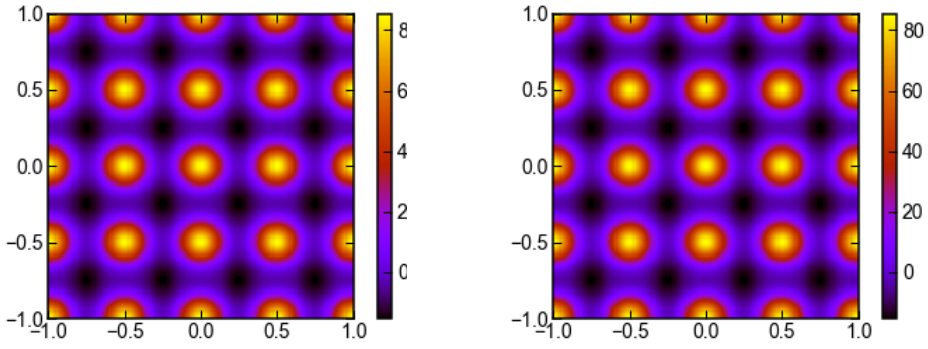
図 3: 2θ-強度グラフ

表 2: KCl における各データの一覧

hkl	200	220	222	400	420	422
2θ	12.848988	18.314135	22.483611	26.070175	29.320779	32.312745
distance	3.140000	2.220315	1.812880	1.570000	1.404251	1.281900
I(hkl)	211.822263	60.561554	7.789260	0.297566	3.369641	1.135681

4.3　実験 3 Al 結晶構造の特定

2θ-強度グラフを図 4.3、各データの一覧を表 3、z=0,z=0.5 における電子密度マップを図 6 に示した。



[1]z=0　　[2]z=0.5

図 4: 電子密度マップ

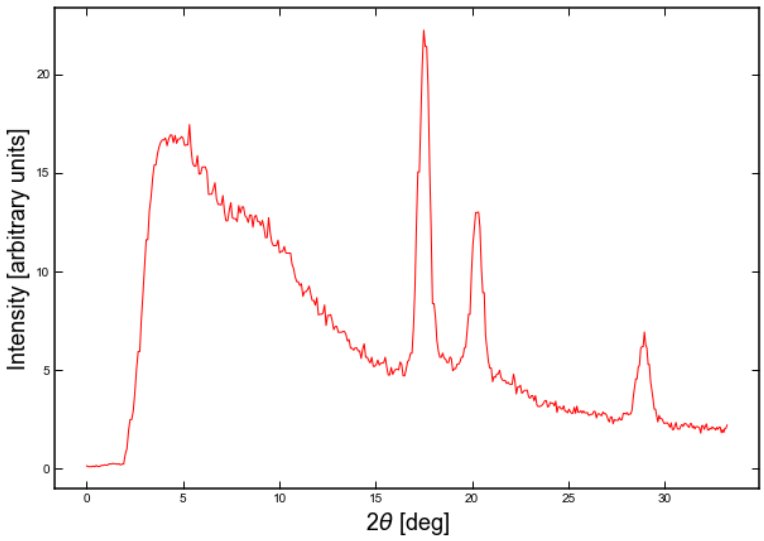
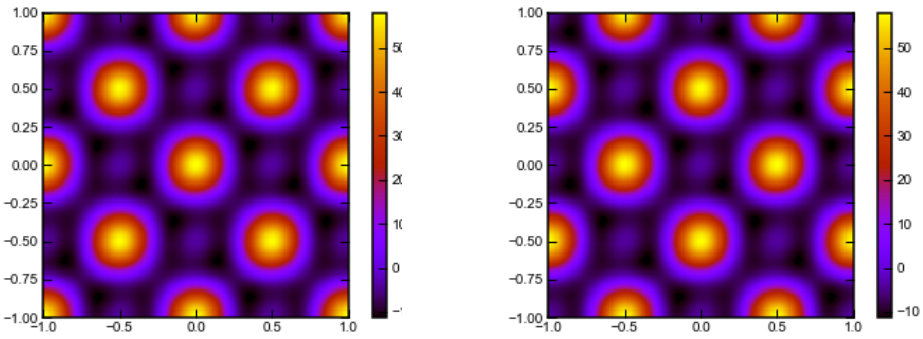


図 5: 2θ-強度グラフ

表 3: Al における各データの一覧

hkl	111	200	220
2θ	17.518690	20.242201	28.896402
distance	2.338269	2.025000	1.4318913
I(hkl)	119.543988	41.426174	10.754482



[1]z=0　　[2]z=0.5

図 6: 電子密度マップ

5　結果の考察

NaCl の電子密度マップは、原子 1 つ毎に濃淡がある様子が見て取れる。またその濃淡が z=0 と z=0.5 では入れ替わることが分かる。イオンにおける電子数を考えると、電子密度マップで濃い原子は塩素であり、電子密度マップで淡い原子はナトリウムであると考えられる。以上のことから NaCl 結晶が面心立方の結晶構造を持つ事が確かめられた。
一方、KCl の電子密度マップは、原子 1 つ毎の濃淡が見て取れない。これはカリウムイオン及び塩化物イオンがアルゴンの電子配置を取るため、X 線では見分ける事ができないためであると考えられる。KCl 結晶の原子を見分けるためには、陽子数が違うことから中性子線での散乱を測定すべきであると考えられる。
Al においては z=0 と z=0.5 において原子の配列が入れ替わることが分かる。このことから Al 結晶が面心立方の結晶構造を持つ事が確かめられた。

6　結論

各結晶においてフーリエ変換を用いて電子密度マップを求めることが出来た。これらを通して結晶構造の決定の仕組みについて理解する事が出来た。またコンピュータを用いたフィッティングの技術を習得することが出来た。

7　マニュアルの問題

(a) (d) においては、結果の項に記載したのでそちらに代える。(e) においては別紙 1 に記載した。

8　教科書の問題

演習問題は別紙 2 に記載した。

9　appendix

【ソースコード】

```python
# function_define
def linear(x, a, b):
    return b*(a + x)

def gauss(x, a, sigma, mean):
    return a / np.sqrt(2.0*np.pi) / sigma * np.exp(-((x-mean)/sigma)**2/2)


def fitting(x, *paramaters):
    a0, b0, a1, sigma1, mean1, a2, sigma2, mean2, a3, sigma3, mean3, a4, sigma4, mean4, a5, sigma5, mean5, a6, sigma6, 
    return linear(x, a0, b0) + \
           gauss(x, a1, sigma1, mean1) + \
           gauss(x, a2, sigma2, mean2) + \
           gauss(x, a3, sigma3, mean3) + \
           gauss(x, a4, sigma4, mean4) + \
           gauss(x, a5, sigma5, mean5) + \
           gauss(x, a6, sigma6, mean6)

# initial
initial_gauss0 =  -30., -1.,
initial_gauss1 = 2.60587478,  0.24806175, 12.55341385 #a1, sigma1, mean1
initial_gauss2 = 32.12624409, 0.26151443, 14.49595295 #a2, sigma2, mean2
initial_gauss3 = 18.65372096, 0.24795276, 20.62008517 #a3, sigma3, mean3
initial_gauss4 = 5.00578413,  0.23956014, 25.37221518
initial_gauss5 = 2.,  0.2, 29.
initial_gauss6 = 2.33902937,  0.22952653, 32.97021728

initial_parameter = initial_gauss0 + initial_gauss1 + initial_gauss2 + initial_gauss3 + \
                    initial_gauss4 + initial_gauss5 + initial_gauss6
# action

one = 70
end= 453

paramater_optimal, covariance = curve_fit(fitting, two_thita_deg[one:], intensity[one:], initial_parameter)
print(paramater_optimal)
```

```
F = [paramater_optimal[i] for i in range(2,20,3)]
I = {'111': F[0]**2, '200': F[1]**2, '220': F[2]**2, '222': F[3]**2, '400': F[4]**2, '420': F[5]**2}
#I = {'111': 3.463, '200': 43.25, '220': 21.25, '222': 5.498, '400': 2.025, '420': 3.898}

points = np.arange(-1,1.01,0.01,dtype= np.complex)
dx,dy = np.meshgrid(points,points)

data = []

for dz in points:

    p1_r = np.exp(-2*pi*1.j*(dx+dy+dz)) +\
           np.exp(-2*pi*1.j*(dx+dy-dz)) +\
           np.exp(-2*pi*1.j*(dx-dy+dz)) +\
           np.exp(-2*pi*1.j*(-dx+dy+dz)) +\
           np.exp(-2*pi*1.j*(dx-dy-dz)) +\
           np.exp(-2*pi*1.j*(-dx+dy-dz)) +\
           np.exp(-2*pi*1.j*(-dx-dy+dz)) +\
           np.exp(-2*pi*1.j*(-dx-dy-dz))
    p1 = np.sqrt(I['111']/8)*p1_r


    p2_r = np.exp(-4*pi*1.j*dx) + np.exp(4*np.pi*1.j*dx) +\
           np.exp(-4*pi*1.j*dy) + np.exp(4*np.pi*1.j*dy) +\
           np.exp(-4*pi*1.j*dz) + np.exp(4*np.pi*1.j*dz)
    p2 = np.sqrt(I['200']/6)*p2_r


    p3_r = np.exp(-4*pi*1.j*(dx+dy))+\
           np.exp(-4*pi*1.j*(dx+dz))+\
           np.exp(-4*pi*1.j*(dx-dy))+\
           np.exp(-4*pi*1.j*(dx-dz))+\
           np.exp(-4*pi*1.j*(-dx+dy))+\
           np.exp(-4*pi*1.j*(-dx+dz))+\
           np.exp(-4*pi*1.j*(-dx-dy))+\
           np.exp(-4*pi*1.j*(-dx-dz))+\
           np.exp(-4*pi*1.j*(dy+dz))+\
           np.exp(-4*pi*1.j*(-dy+dz))+\
           np.exp(-4*pi*1.j*(dy-dz))+\
           np.exp(-4*pi*1.j*(-dy+dz))

    p3 = np.sqrt(I['220']/12)*p3_r
    p4_r = np.exp(-4*pi*1.j*(dx+dy+dz))+\
           np.exp(-4*pi*1.j*(dx+dy-dz))+\
           np.exp(-4*pi*1.j*(dx-dy+dz))+\
           np.exp(-4*pi*1.j*(-dx+dy+dz))+\
           np.exp(-4*pi*1.j*(dx-dy-dz))+\
           np.exp(-4*pi*1.j*(-dx+dy-dz))+\
           np.exp(-4*pi*1.j*(-dx-dy+dz))+\
           np.exp(-4*pi*1.j*(-dx-dy-dz))

    p4 = np.sqrt(I['222']/8)*p4_r



    p5_r = np.exp(-8*pi*1.j*dx) + np.exp(8*np.pi*1.j*dx) +\
           np.exp(-8*pi*1.j*dy) + np.exp(8*np.pi*1.j*dy) +\
           np.exp(-8*pi*1.j*dz) + np.exp(8*np.pi*1.j*dz)
    p5 = np.sqrt(I['400']/6)*p5_r


    p6_r = np.exp(-4*pi*1.j*(2*dx+dy))+\
           np.exp(-4*pi*1.j*(2*dx+dz))+\
           np.exp(-4*pi*1.j*(2*dx-dy))+\
           np.exp(-4*pi*1.j*(2*dx-dz))+\
           np.exp(-4*pi*1.j*(dx+2*dy))+\
           np.exp(-4*pi*1.j*(dx+2*dz))+\
           np.exp(-4*pi*1.j*(dx-2*dy))+\
           np.exp(-4*pi*1.j*(dx-2*dz))+\
           np.exp(-4*pi*1.j*(dy+2*dz))+\
           np.exp(-4*pi*1.j*(dy-2*dz))+\
           np.exp(-4*pi*1.j*(-dy+2*dz))+\
           np.exp(-4*pi*1.j*(-dy-2*dz))+\
           np.exp(-4*pi*1.j*(2*dy+dz))+\
           np.exp(-4*pi*1.j*(2*dy-dz))+\
           np.exp(-4*pi*1.j*(-2*dy+dz))+\
           np.exp(-4*pi*1.j*(-2*dy-dz))

    p6 = np.sqrt(I['420']/12)*p6_r


    p = p1 + p2 + p3 + p4 + p5 + p6


    data.append(p)
```

```
number = int(float(input("-1=< z =< 1 : "))*100+100)
plt.imshow(np.real(data[number]), cmap="gnuplot")

plt.colorbar()
```