

Name: Marius Pfeiffer

Matrikel-Nr.: 4188573

E-Mail: marius.pfeiffer@stud.uni-heidelberg.de

Betreut durch: Valentin Krems

13.02.2025

Versuch 255: Röntgenspektrometer



Abbildung 1: Versuchsaufbau (Quelle: Praktikumsanleitung).

Inhaltsverzeichnis

1 Einleitung	2
1.1 Physikalische Grundlagen	2
1.2 Versuchsdurchführung	5
2 Messprotokoll	6
3 Auswertung	8
3.1 Röntgenspektrum mit LiF-Kristall	8
3.2 Bestimmung der Wellenlängen der $K_{\alpha,\beta}$ -Linien	10
3.3 Zählrate als Funktion der Beschleunigungsspannung	11
3.4 Röntgenspektrum, ausgemessen mit NaCl-Kristall.	12
4 Zusammenfassung und Diskussion	14

1 Einleitung

In Versuch 255 setzen wir uns mit der Funktionsweise einer Röntgenröhre, sowie dem charakteristischen Spektrum der Röntenstrahlung auseinander. Neben quantitativen Untersuchungen des Röntgenspektrums selbst, nutzen wir das Prinzip der Bragg-Reflexion, um unter anderem die Gitterkonstante eines NaCl-Kristalls zu bestimmen.

1.1 Physikalische Grundlagen

Die Röntgenröhre

Eine Röntgenröhre ist aufgebaut aus einer Glühkathode und einer Anode, welche sich in einem evakuierten Glaskolben befinden. Durch Glühemission werden aus der Kathode Elektronen freigesetzt, welche durch eine Beschleunigungsspannung von 10 bis 100kV, welche zwischen Kathode und Anode anliegt beschleunigt werden. Der kontinuierliche Teil des Röntgenspektrums wird durch die ausgehende Bremsstrahlung beim Abbremsen der Elektronen im Anodenmaterial verursacht. Diese Strahlung setzt bei einer bestimmten Grenzwellenlänge λ_{gr} ein, welche sich nach

$$\lambda_{gr} = \frac{hc}{eU} \quad (1)$$

berechnen lässt. Hierbei sind h , c , e das Plank'sche Wirkungsquantum, die Lichtgeschwindigkeit und die Elementarladung. U ist die an der Röntgenröhre anliegende Beschleunigungsspannung. Durch frei werdende Strahlung bei der Ionisation des Anodenmaterials ist dem kontinuierlichen Spektrum ein diskretes Spektrum überlagert, zu sehen in Abbildung (2), welches charakteristisch für das jeweilige Anodenmaterial ist.

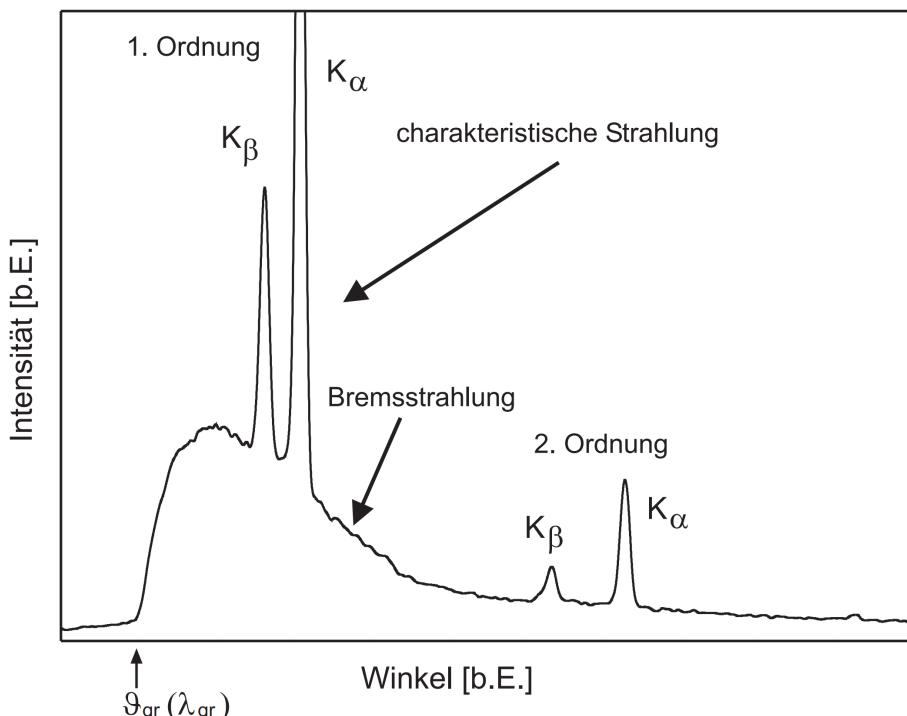


Abbildung 2: Kontinuierliches und diskretes Röntgenspektrum (Quelle: Praktikumsanleitung).

Die Positionen der Linien im diskreten Spektrum sind abhängig von der Ursprungs- und Ziel-Schale von bzw. zu welcher der Übergang des Elektrons stattfindet. Beispielsweise

bezeichnen wir die Strahlung der Übergänge von der L - auf die K -Schale als K_{α} -Strahlung, die für die Übergänge der M - auf die K -Schale als K_{β} -Strahlung. Die freiwerdende Energie eines Übergangs von der n -ten zur m -ten Schale lässt sich durch das Moseley'sche Gesetz

$$E_{n \rightarrow m} = hcR_{\infty}(Z - A)^2 \left(\frac{1}{m^2} - \frac{1}{n^2} \right) \quad (2)$$

berechnen. Hier gehen die Rydbergkonstante R_{∞} , sowie die Kernladungszahl Z und die Abschirmung der Kernladung als Abschirmungskonstante A mit ein. Nähert man die Abschirmungskonstante als $A \approx 1$ an, so lässt sich mit dem Moseley'schen Gesetz eine Näherung der Energie für die K_{α} -Strahlung abhängig von der Kernladungszahl angeben:

$$E_{2 \rightarrow 1} = hcR_{\infty}(Z - 1)^2 \left(\frac{1}{1} - \frac{1}{2^2} \right) = \frac{3}{4}hcR_{\infty}(Z - 1)^2. \quad (3)$$

Es ist zu beachten, dass bei genauerer Betrachtung, neben der Hauptquantenzahl, noch eine Entartung der Drehimpuls- und Spinquantenzahl in die freigesetzte Energie der Übergänge eingeht. Das Moseley'sche Gesetz gibt somit nur eine Näherung an.

Bragg-Reflexion

Als Bragg-Reflexion bezeichnet man die Beugung von Röntgenstrahlung durch die Gitterstruktur von Kristallen. Die Atomabstände in der Kristallstruktur befinden sich in der gleichen Größenordnung wie die Wellenlängen der Röntgenstrahlung, weshalb sich die Bragg-Reflexion zur Untersuchung des Röntgenspektrums eignet. Die Bragg-Reflexion beruht darauf, dass auf den Kristall treffende Röntgenstrahlung sowohl an der Oberfläche, als auch den tieferliegenden Netzebenen der Kristallstruktur reflektiert wird. Beträgt der Gangunterschied $\Delta s = 2d \sin(\vartheta)$ zweier Teilbündel, die unter dem Winkel ϑ eintreffen, siehe Abbildung (3), ein Vielfaches der Wellenlänge λ , so interferieren diese konstruktiv, andernfalls destruktiv. Dies ist festgehalten durch das Bragg'sche Gesetz

$$2d \sin(\vartheta) = n\lambda, \quad n \in \mathbb{N}. \quad (4)$$

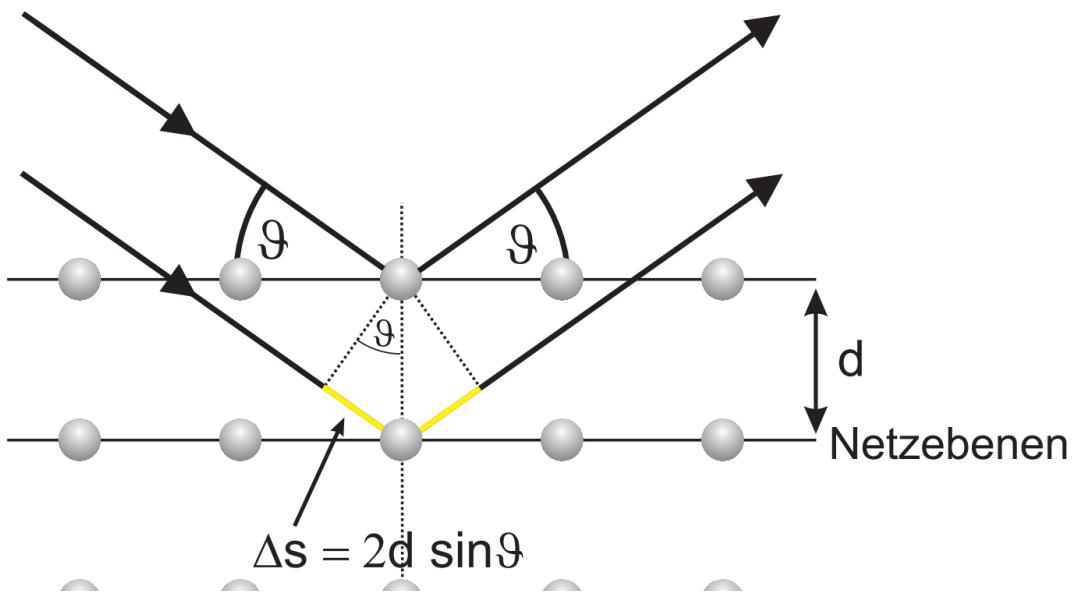


Abbildung 3: Bragg-Reflexion von Röntgenstrahlung an einem Kristall (Quelle: Praktikumsanleitung).

Bei der sogenannten Drehkristallmethode wird diese Gesetzmäßigkeit angewandt. Dabei trifft Röntgenstrahlung auf einen Kristall, welcher um die Achse senkrecht zu dieser gedreht wird. Welche Wellenlänge reflektiert wird, hängt dann gerade von der Winkelstellung des Kristalls ab. Die Intensität der reflektierten Strahlung kann dann mit einem Zählrohr gemessen und so Spektren wie in Abbildung (2) aufgezeichnet werden. Ist der Drehkristall so justiert, dass gerade die Wellenlänge einer bekannten diskreten Linie reflektiert wird, lässt sich mit dem Bragg'schen Gesetz der Netzebenenabstand d berechnen.

Kristallstruktur

Kristalle sind aus sich periodisch wiederholenden Elementarzellen (EZ) aufgebaut, wie sie Beispielsweise in Abbildung (4) zu sehen sind. Bei NaCl- und LiF-Kristallen sind die drei Gitterkonstanten, also die Seitenlängen a einer Elementarzelle, gleich groß.

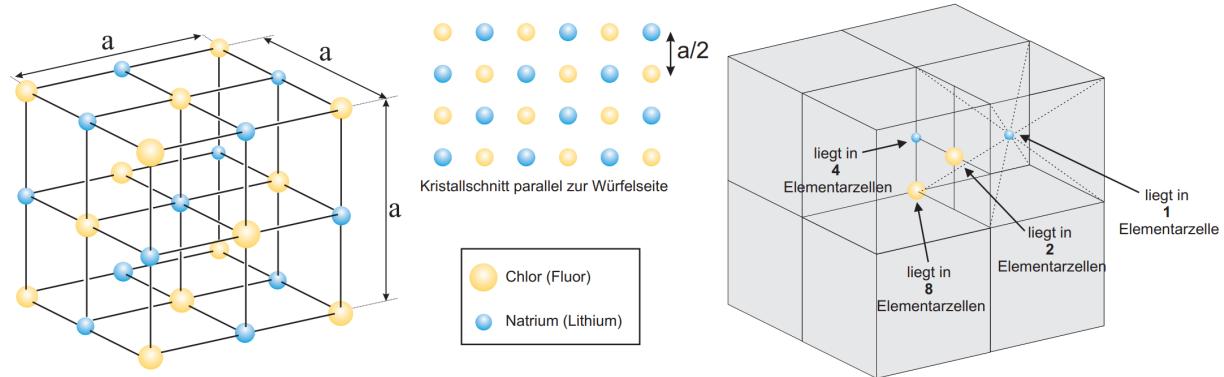


Abbildung 4: Elementarzelle eines NaCl-Kristalls, deren periodische Anordnung, sowie der Kristallschnitt (Quelle: Praktikumsanleitung).

Zur Ermittlung der Anzahl an Atomen, die einer Elementarzelle angehören, muss man beachten, zu wie vielen benachbarten Elementarzellen die Atome jeweils außerdem zählen. Dies ist ebenfalls in Abbildung (4) dargestellt. Trägt ein Atom zu x Elementarzellen bei, so zählt es für eine einzelne Elementarzelle nur zu $1/x$. Tabelle (1) zeigt, wie sich die Anzahl an Atomen am Beispiel der NaCl- (LiF-) Kristalle bestimmen lässt.

Tabelle 1: Bestimmung der Beiträge der Atome zu einer EZ bei NaCl- (LiF-) Kristallen.

Element	Position	Beitrag zu x EZ	Beitrag zu einer EZ	Anzahl	Beitrag
Cl (F)	Ecken	8	1/8	8	1
Cl (F)	Mitte Fläche	2	1/2	6	3
Na (Li)	Mitte Kante	4	1/4	12	3
Na (Li)	Mitte Zelle	1	1/1	1	1

Aus den Beiträgen können ablesen, dass zu jeder Elementarzelle 4 Chlor- (Fluor-) Atome und 4 Natrium- (Lithium-) Atome, also 4 NaCl- (LiF-) Moleküle, beitragen. Die Zahl 4 geht bei der Berechnung der Avogadrokonstante wie folgt mit ein:

$$N_A = 4 \frac{V_{Mol}}{V}. \quad (5)$$

Dabei ist V_{Mol} das Molvolumen eines einzelnen Moleküls und V das Volumen einer Elementarzelle. Dieses lässt sich aus dem Netzebenenabstand d , welcher hier $a/2$ beträgt, berechnen. Somit gilt

$$N_A = 4 \frac{V_{Mol}}{(2d)^3} = 4 \frac{M_{Mol}}{\rho(2d)^3} = \frac{1}{2} \frac{M_{Mol}}{\rho d^3}. \quad (6)$$

1.2 Versuchsdurchführung

Zur Durchführung der Messungen verwenden wir ein Röntgengerät, welches mit einem Zählrohr-Goniometer ausgestattet ist (Abbildung (5)). Die Röntgenstrahlung wird durch eine Röntgenröhre mit Molybdänanode erzeugt und über einen Kollimator auf den Drehkristall fokussiert. Das Zählrohr am Goniometer dreht sich im Verhältnis 2:1 mit dem Drehkristall, sodass es immer genau die Strahlung, welche im Winkel ϑ reflektiert wird erfassen kann.

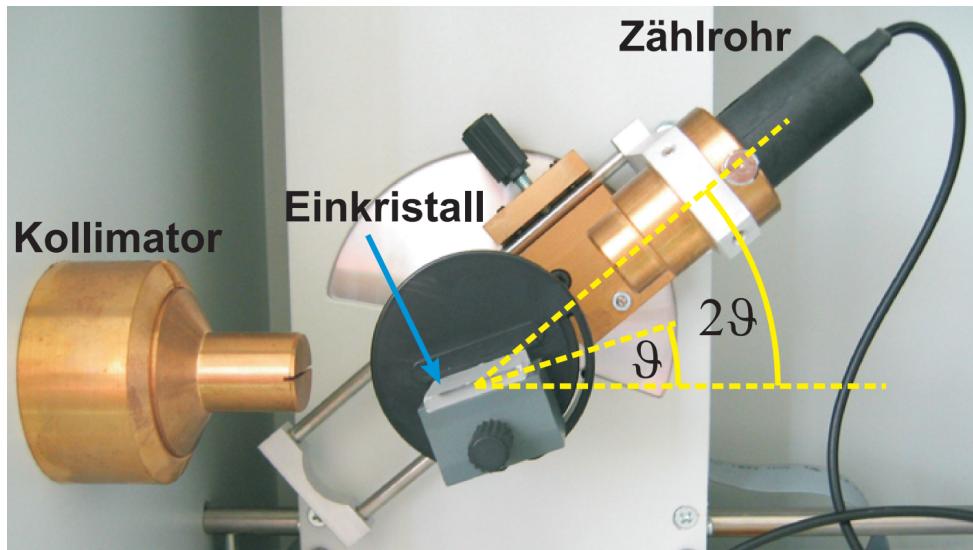


Abbildung 5: Montierung des Drehkristalls und des Zählrohr-Goniometers im Winkelverhältnis 2:1 (Quelle: Praktikumsanleitung).

Messung des Röntgenspektrums mit einem LiF-Kristall. Bei einer Beschleunigungsspannung von $U = 35\text{kV}$, einem Strom von $I = 1\text{mA}$ und einer Torzeit von $t = 5\text{s}$ zeichnen wir das Röntgenspektrum in einem Winkelbereich von 3° bis 22° in 0.2° -Schritten auf.

Vermessen der K_α und K_β -Linien des Anodenmaterials. Aus unseren Messdaten der vorherigen Aufgabe entnehmen wir die ungefähren Positionen der K_α - und K_β -Linien. Mit den gleichen Einstellungen für Beschleunigungsspannung und Strom nehmen wir in deren Umgebung das Spektrum noch einmal in Winkelschritten von 0.1° und einer Torzeit von 20s auf.

Zählrate als Funktion der Beschleunigungsspannung. Bei einem festen Winkel von 7.5° Zeichnen wir die Zählrate für Beschleunigungsspannungen im Bereich von 20 bis 35kV in 1kV -Schritten jeweils über 20s auf.

Messung des Röntgenspektrums mit einem NaCl-Kristall. Hier führen wir die Messung aus Aufgabe 1 noch einmal mit einem NaCl-Kristall in einem Winkelbereich von 3° bis 18° durch.

2 Messprotokoll

Versuchsprotokoll 255

Marius Pfeiffer
Robert Grossch

13.02.2025

Aufgabe 1 (LiF)

$$U = 35 \text{ kV} \quad t = 5 \text{ s}$$

$$I = 1 \text{ mA} \quad \Delta\beta = 0.2^\circ$$

$$\beta \in [3^\circ, 22^\circ]$$

Aufgabe 2 (LiF)

$$U = 35 \text{ kV} \quad t = 20 \text{ s}$$

$$I = 1 \text{ mA} \quad \Delta\beta = 0.1^\circ$$

β -Bereiche:

K_β 1. Ordnung ca. 8.5° bis 9.5°

K_α 1. Ordnung ca. 9.6° bis 10.7°

K_β 2. Ordnung ca. 17.5° bis 18.9°

K_α 2. Ordnung ca. 20.1° bis 21.2°

Aufgabe 3 (LiF)

$$I = 1 \text{ mA} \quad t = 20 \text{ s} \quad \beta = 7.5^\circ$$

Spannung [kV]	Zählrate [$\frac{1}{s}$]
20	0.65
21	1.60
22	1.40
23	10.35
24	123.9
25	296.5
26	425.3
27	551.3
28	646.8
29	740.4
30	841.4
31	937.4
32	1005
33	1092
34	1155
35	1213

Aufgabe 4 (NaCl)

$$U = 35 \text{ kV} \quad t = 5 \text{ s}$$

$$I = 1 \text{ mA} \quad \Delta\beta = 0.2^\circ$$

$$\beta \in [3^\circ, 18^\circ]$$

V. LG

3 Auswertung

Vorbemerkungen

Sofern nicht anders angegeben, berechnen wir die Fehler zusammengesetzter Werte anhand der standardmäßigen Gauß'schen Fehlerfortpflanzung. Die σ -Abweichung zweier fehlerbehafteter Werte $x \pm \Delta x$ und $y \pm \Delta y$ berechnen wir anhand der Formel

$$\sigma = \frac{|x - y|}{\sqrt{\Delta x^2 + \Delta y^2}}. \quad (7)$$

Für eine Anzahl an Ereignissen N berechnet sich der Fehler gemäß $\Delta N = \sqrt{N}$. Zur Fehlerangabe für Zählraten $n = N/t$ leiten wir daraus die Formel

$$\Delta n = \frac{\Delta N}{t} = \frac{\sqrt{N}}{t} = \frac{\sqrt{nt}}{t} = \sqrt{\frac{n}{t}} \quad (8)$$

ab.

3.1 Röntgenspektrum mit LiF-Kristall

Abbildung (6) zeigt das Röntgenspektrums, welches wir mit der Drehkristallmethode mit einem LiF-Kristall im Winkelbereich von 3° bis 22° aufgezeichnet haben. Bereits hier ist zu erkennen, dass wir ab einem Winkel von etwa 5° beginnen, Bremsstrahlung zu detektieren.

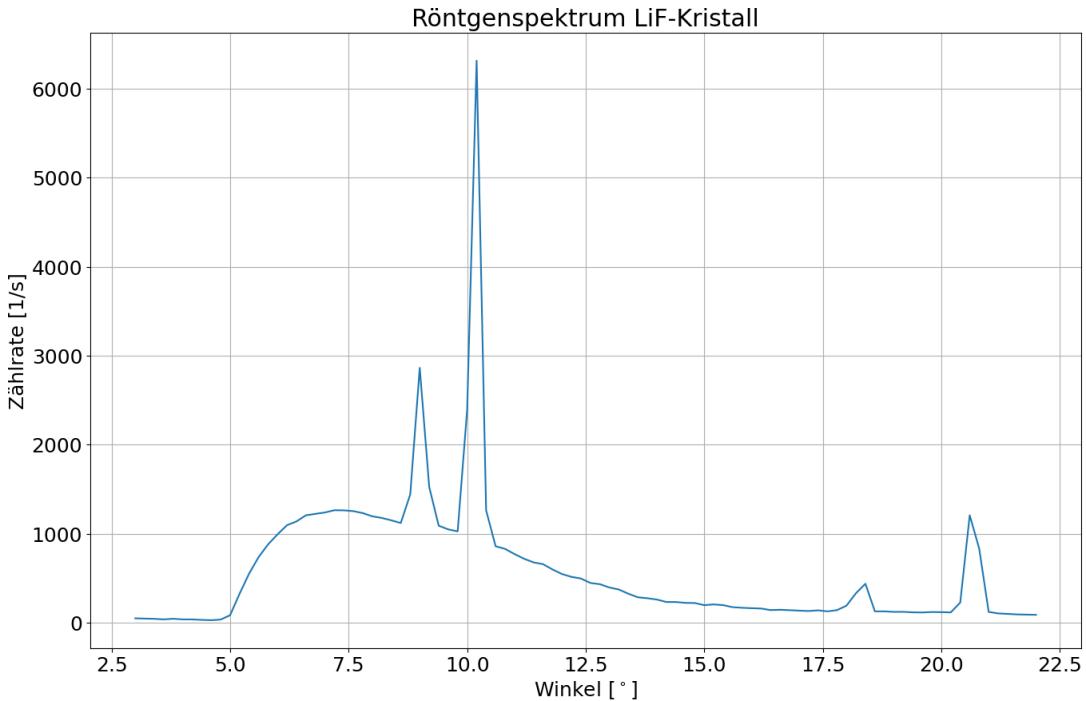


Abbildung 6: Röntgenspektrum, ausgemessen mit LiF-Kristall.

Um den Winkel und die zugehörige Grenzwellenlänge genauer zu bestimmen, betrachten wir den nahezu linearen Bereich des Spektrums am kurzweligen Ende und fitten an dieses eine lineare Funktion der Form $f(x; a, b) = ax + b$, wie in Abbildung (7) zu sehen.

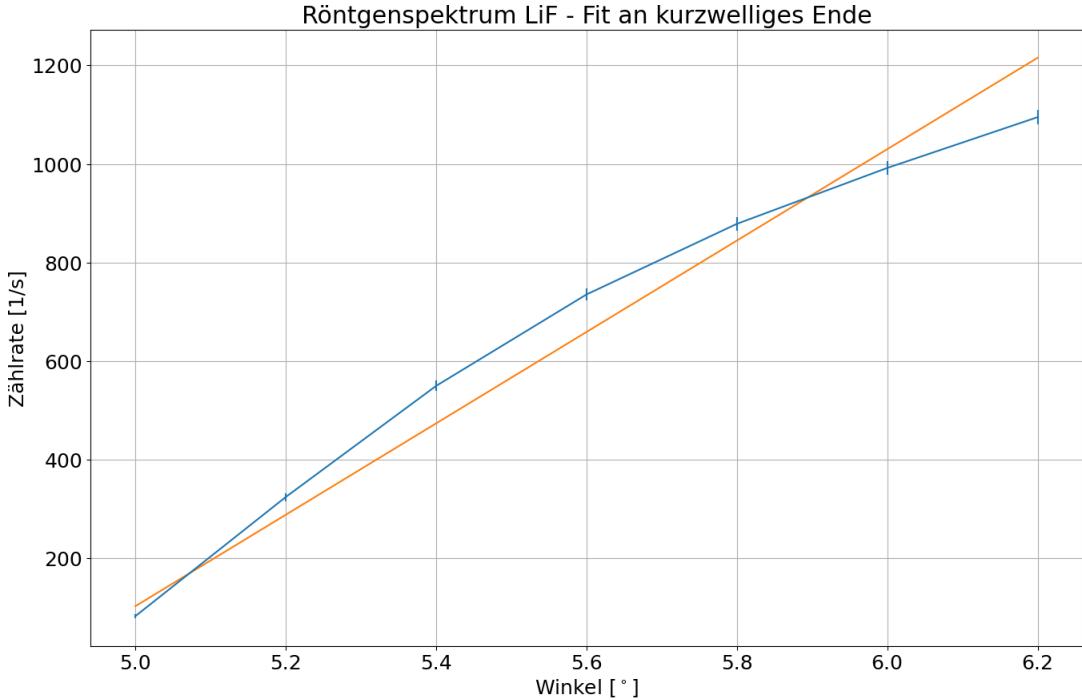


Abbildung 7: Linearer Fit an das kurzwellige Ende des zuvor gezeigten Röntgengenspektrums.

Für die optimierten Parameter erhalten wir die Werte

$$a = 926.94 \pm 56.62 \frac{1}{\text{s}^\circ}, \quad b = -4531.53 \pm 297.50 \frac{1}{\text{s}}. \quad (9)$$

Mit diesen berechnen wir den Winkel, ab dem die Bremsstrahlung einsetzt, als Nullstelle der linearen Funktion nach

$$\vartheta_{0,1,\text{Ord}} = -\frac{b}{a} = 4.9 \pm 0.5^\circ. \quad (10)$$

Mithilfe des Bragg'schen Gesetzes (4) können wir daraus nach

$$\lambda = \frac{2d \sin(\vartheta)}{n} \quad (11)$$

die entsprechende Grenzwellenlänge berechnen. Da es sich hierbei um den Einsatz des Spektrums erster Ordnung handelt, setzen wir $n = 1$. Den Netzebenenabstand d des LiF-Kristalls entnehmen wir aus der Praktikumsanleitung als $d = 201.4\text{pm}$. Damit erhalten wir für die Grenzwellenlänge einen Wert von

$$\lambda_{\text{Grenz}} = (34.33 \pm 3.08)\text{pm}. \quad (12)$$

Setzen wir nun in (1) diese Grenzwellenlänge, die Spannung $U = 35\text{kV}$ ein und stellen diese nach dem Plank'schen Wirkungsquantum h um, so erhalten wir für dieses einen Wert von

$$h = (6.4 \pm 0.6) \cdot 10^{-34} \text{J s}. \quad (13)$$

Die Grenzwellenlänge λ_{grenz} können wir weitergehend einsetzen, um den Winkel zu bestimmen, ab dem das Spektrum zweiter Ordnung einsetzt. Dazu stellen wir (4) nach ϑ um, und setzen $n = 2$. Wir erhalten damit einen Winkel von

$$\vartheta_{0,2,\text{Ord}} = (9.813 \pm 0.016)^\circ. \quad (14)$$

3.2 Bestimmung der Wellenlängen der $K_{\alpha,\beta}$ -Linien

Wir betrachten nun die Wellenlängenbereiche, in welchen die K_{α} - und K_{β} -Linien erster und zweiter Ordnung zu sehen sind, genauer. Die Ausschnitte, aufgenommen in Winkelschritten von 0.1° , sind in Abbildung (8) zu sehen.

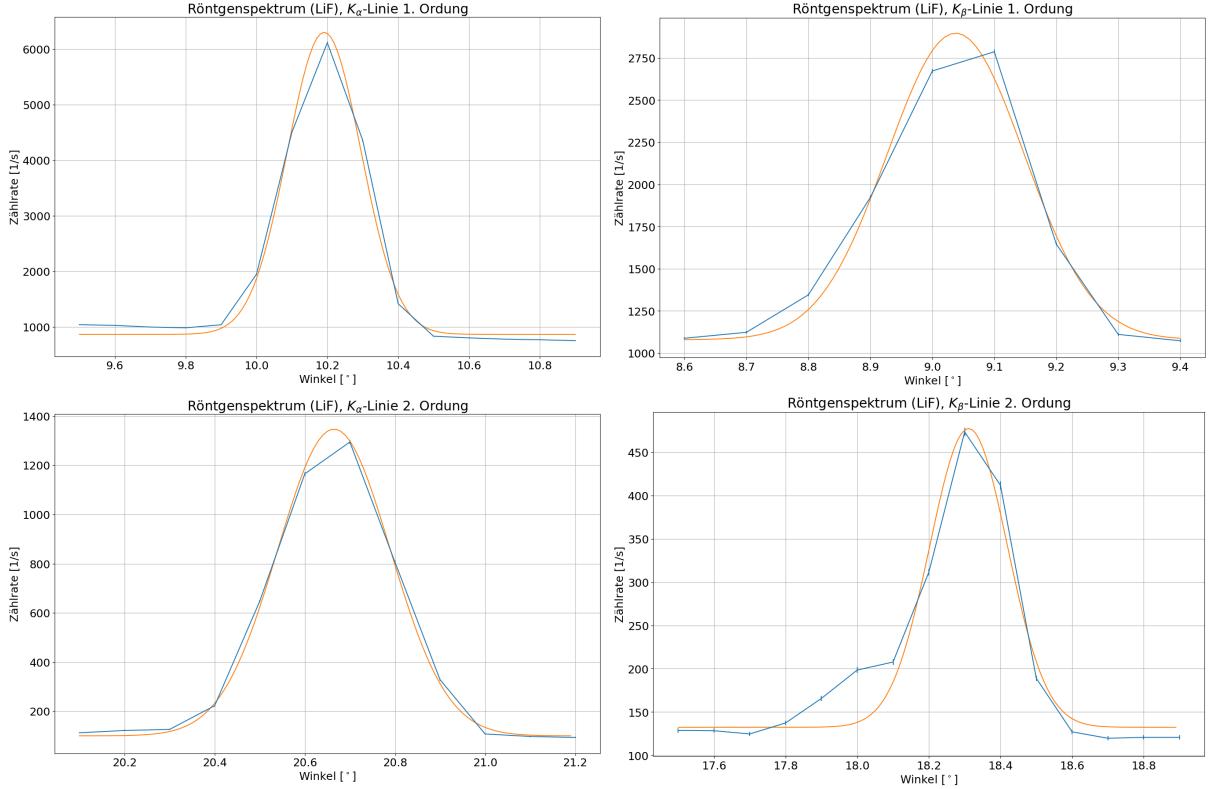


Abbildung 8: K_{α} - und K_{β} -Linien erster Ordnung (oben) und zweiter Ordnung (unten).

Um die Lage der Linien, sowie die Halbwertsbreite, zu ermitteln, fitten wir an die Daten eine Exponentialfunktion der Form

$$f(x; A, \mu, \sigma, c) = \frac{A}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(x - \mu)^2}{2\sigma^2}\right) + c. \quad (15)$$

Die Halbwertsbreite ermitteln wir dann aus der Standardabweichung σ nach der Formel

$$\text{FWHM} = 2\sqrt{2\ln(2)}\sigma. \quad (16)$$

Der optimierte Wert von μ beschreibt hier erneut den *Winkel* des Kristalls, bei welchem sich die Linie befindet. Um daraus die zugehörigen Wellenlängen zu berechnen, verwenden wir wieder das Bragg'sche Gesetz (4), mit entsprechendem $n \in \{1, 2\}$ für die Linien erster bzw. zweiter Ordnung. Bei der Berechnung der Wellenlänge verwenden wir den Wert σ/\sqrt{N} als Fehler des Winkels. Hierbei ist σ die Standardabweichung der optimierten Gaußkurve und N die Länge des Datensatzes, auf dem der Fit durchgeführt wurde. Die Ergebnisse sind in Tabelle (2) zusammengefasst.

Tabelle 2: Ergebnisse des Röntgenspektrums (LiF) für die K_α - und K_β -Linien in der 1. und 2. Ordnung.

Linie	Ordnung	Mittelw. [°]	Stabw. [°]	FWHM [°]	λ [pm]
K_α	1	10.191 ± 0.005	0.104 ± 0.005	0.245 ± 0.011	71.27 ± 0.19
K_β	1	9.038 ± 0.007	0.110 ± 0.008	0.260 ± 0.018	63.27 ± 0.26
K_α	2	20.665 ± 0.004	0.125 ± 0.004	0.294 ± 0.008	71.07 ± 0.12
K_β	2	18.311 ± 0.011	0.109 ± 0.011	0.256 ± 0.025	63.27 ± 0.10

Aus den Werten der Wellenlängen der Linien erster und zweiter Ordnung bilden wir nun noch die Mittelwerte, um mit diesen später weiterzurechnen:

$$\lambda_{K_\alpha} = (71.17 \pm 0.12)\text{pm}, \quad (17)$$

$$\lambda_{K_\beta} = (63.27 \pm 0.14)\text{pm}. \quad (18)$$

Diese beiden Werte sind charakteristisch für die Molybdänanode der Röntgenröhre.

3.3 Zählrate als Funktion der Beschleunigungsspannung

Bei einem festen Winkel des LiF-Kristalls von 7.5° haben wir die Zählraten für die Beschleunigungsspannungen im Bereich von 20 bis 35kV aufgezeichnet. Die Daten sind in Abbildung (9) zu sehen.

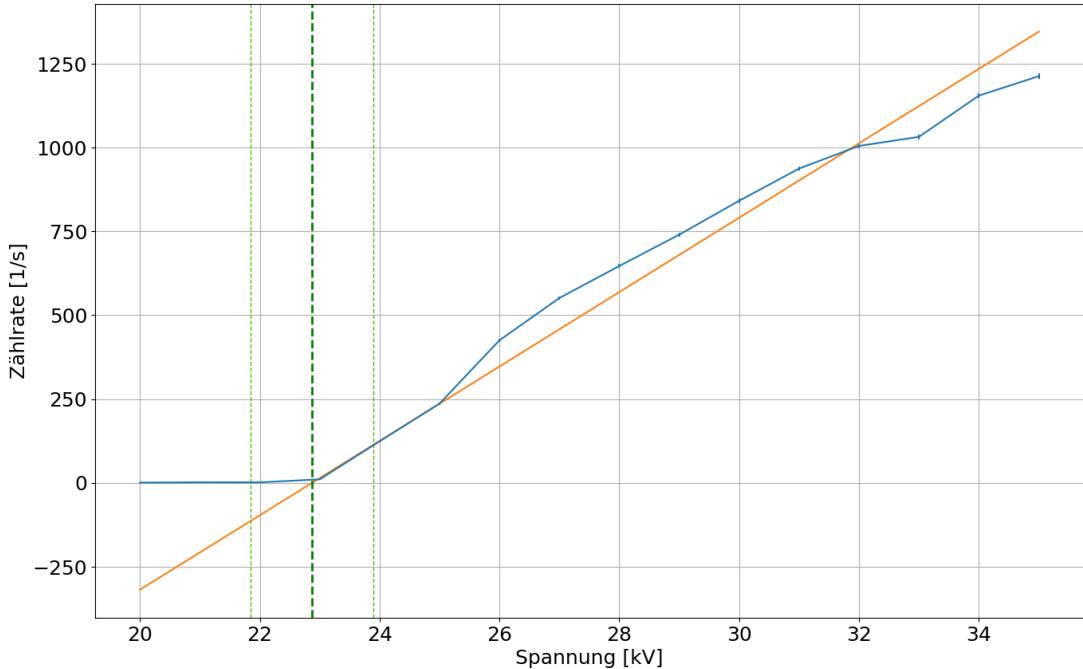


Abbildung 9: Zählraten bei steigender Beschleunigungsspannung unter einem Winkel von 7.5° .

Ebenfalls in Abbildung (9) zu sehen ist eine linearer Funktion, welchen wir an diese Daten angepasst haben, um die Nullstelle, also den Punkt, ab dem die Aussendung von Bremsstrahlung einsetzt, zu finden. Nach dem gleichen Schema wie zuvor, können wir

anhand der optimierten Parameter der linearen Funktion deren Nullstelle zu einem Wert von

$$U_0 = (22.9 \pm 1.1)\text{kV} \quad (19)$$

bestimmen. Die zu einem Winkel von 7.5° korrespondierende Wellenlänge berechnen wir auf

$$\lambda_{7.5} = (52.6 \pm 0.4)\text{pm}. \quad (20)$$

Die ermittelte Wellenlänge und die Einsatzspannung können wir nun erneut in Gleichung (1) einsetzen, um das Planck'sche Wirkungsquantum zu bestimmen. Hier erhalten wir einen Wert von

$$h = (6.43 \pm 0.30) \cdot 10^{-34} \quad (21)$$

3.4 Röntgenspektrum, ausgemessen mit NaCl-Kristall.

Wir haben die Messung aus Versuchsteil 1 wiederholt und die Drehkristallmethode mit einem NaCl-Kristall angewandt, um das Röntgenspektrum in einem Winkelbereich von 3° bis 18° aufzuzeichnen. Dieses ist in Gänze in Abbildung (10) zu sehen.

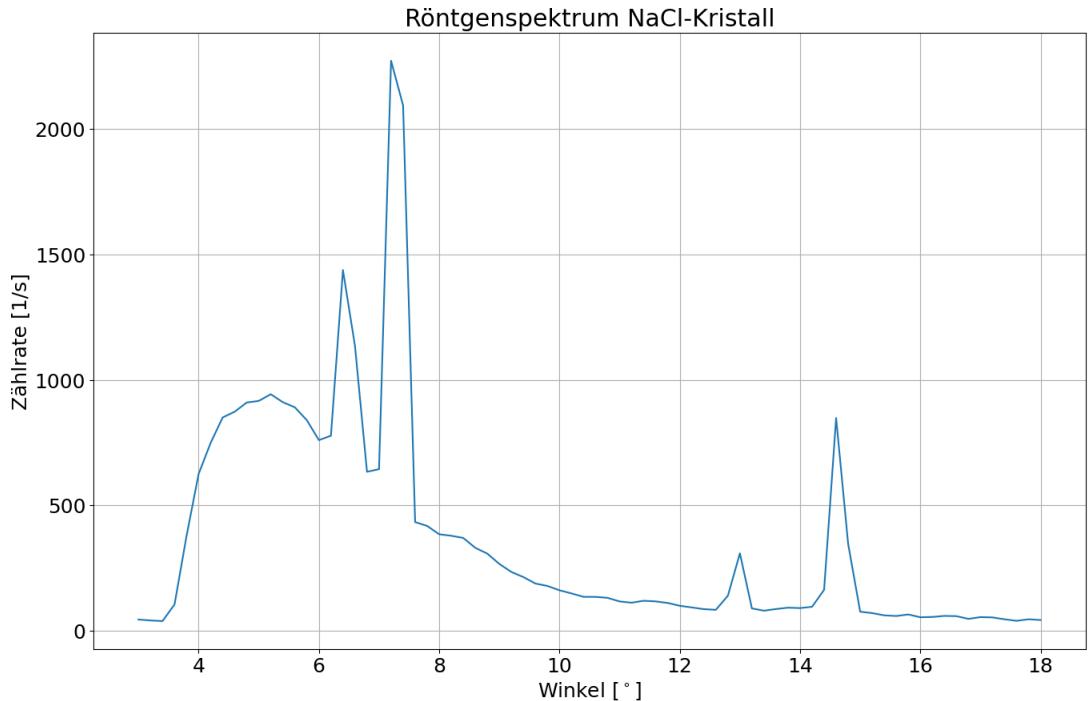


Abbildung 10: Röntgenspektrum mit NaCl-Kristall.

Ziel ist es in diesem Aufgabenteil, die Gitterkonstante von NaCl zu ermitteln. Wir wissen, dass diese gerade dem Doppelten des Netzebenenabstands d entspricht, welchen wir über das Bragg'sche Gesetz

$$d = \frac{n\lambda}{2 \sin(\vartheta)} \quad (22)$$

ermitteln können.

Hierzu ermitteln wir nun erneut die Winkelpositionen der K_α - und K_β -Linien erster Ordnung, um diese mit den Wellenlängen der Linien aus Aufgabe 2 zu identifizieren. Dies können wir tun, da diese Werte charakteristisch für die Molybdänanode der Röntgenröhre sind und damit unabhängig von der Art des Kristalls. Mit einem Fit zweier Gaußkurven an die beiden Linien erster Ordnung, zu sehen in Abbildung (11), ermitteln wir die Winkel zu

$$\beta_{K_\alpha} = 7.30 \pm 0.05^\circ, \quad (23)$$

$$\beta_{K_\beta} = 6.46 \pm 0.06^\circ. \quad (24)$$

Diese Werte setzen wir nun mit den Wellenlängen (17) und (18) aus Aufgabe 2 und $n = 1$ für die erste Ordnung in Gleichung (22) ein. Wir erhalten damit die beiden Werte für d :

$$d_{K_\alpha} = (280.4 \pm 1.8)\text{pm}, \quad (25)$$

$$d_{K_\beta} = (281.2 \pm 2.3)\text{pm}. \quad (26)$$

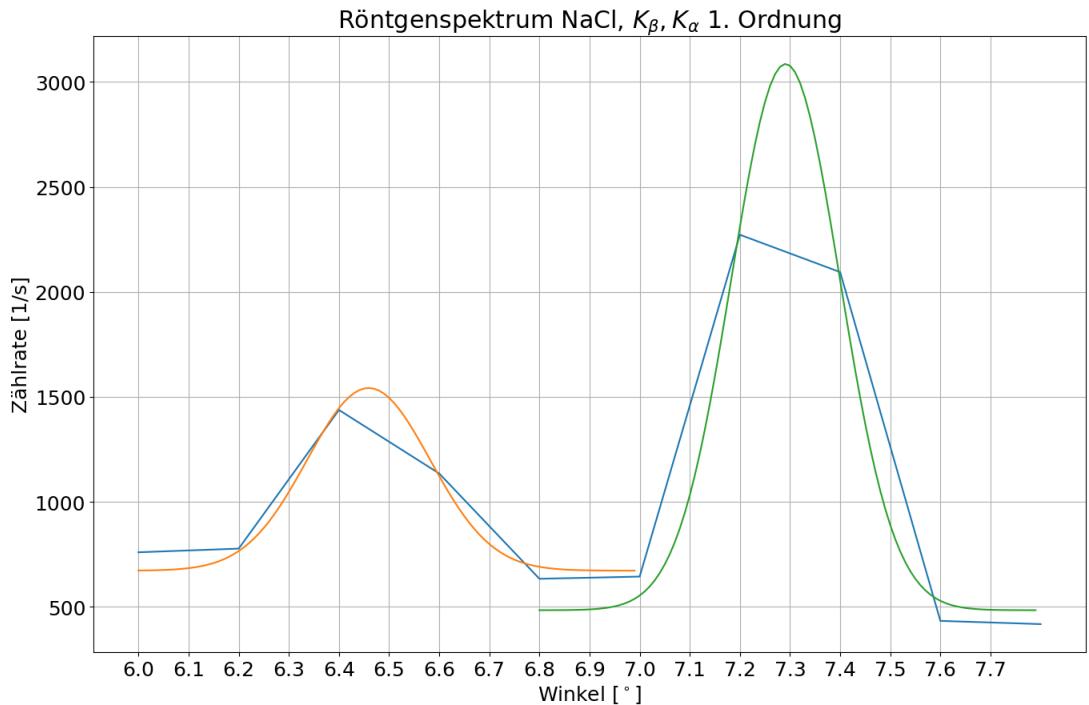


Abbildung 11: Fit an die K-Linien 1. Ordnung des Röntgengenspektrums, ausgemessen mit NaCl-Kristall.

Für den Mittelwert dieser beiden Werte gilt also

$$d = (280.8 \pm 1.5)\text{pm}. \quad (27)$$

Dies ist der Netzebenenabstand von NaCl. Um die Gitterkonstante zu berechnen, verdopeln wir diesen und erhalten

$$a = 2d = (561.6 \pm 2.9)\text{pm}. \quad (28)$$

Außerdem können wir nach Gleichung (6) anhand des Netzebenenabstands die Avogadrokonstante N_A berechnen. Hierzu ziehen wir noch die Dichte von NaCl, $\rho = 2.164 \frac{\text{g}}{\text{cm}^3}$, und dessen Molekulargewicht, $M = 58.44\text{g}$, hinzu. Wir kommen damit auf einen Wert von

$$N_A = (6.10 \pm 0.10) \cdot 10^{23} \frac{1}{\text{mol}}. \quad (29)$$

4 Zusammenfassung und Diskussion

In Versuch 255 untersuchten wir unter Einsatz der Drehkristallmethode mit einem LiF- und einem NaCl-Kristall das Spektrum der Röntgenstrahlung einer Röntgenröhre mit Molybdänanode. Das Röntgenspektrum zeichnet sich durch ein kontinuierliches Spektrum aus, welches durch Bremsstrahlung erzeugt wird, dem ein diskretes Spektrum, erzeugt durch Ionisation im Anodenmaterial der Röntgenröhre, überlagert ist.

Wir untersuchten zunächst das Röntgengenspektrum, aufgezeichnet in der Drehkristallmethode mit einem LiF-Kristall. Dieses ist noch einmal in Abbildung (12) abgebildet.

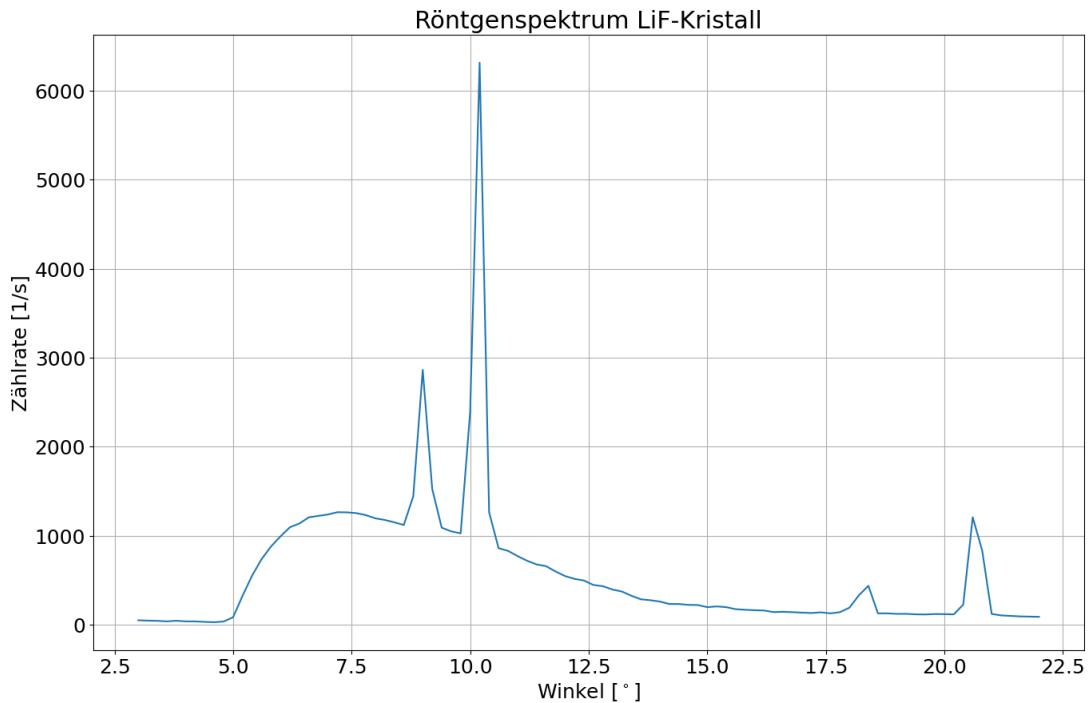


Abbildung 12: Röntgenspektrum mit LiF-Kristall

Im ersten Teil bestimmten wir die Grenzwellenlänge des Spektrums, also den Punkt am kurzweligen Ende, an dem das Spektrum erster Ordnung einsetzt. Hierzu ermittelten wir einen Wert von

$$\lambda_{Grenz} = (34.33 \pm 3.08)\text{pm}. \quad (30)$$

Aus der Grenzwellenlänge berechneten wir für das Planck'sche Wirkungsquantum einen Wert von

$$h = (6.4 \pm 0.6) \cdot 10^{-34}\text{J s}. \quad (31)$$

Dieser Wert weicht von dem in der Praktikumsanleitung gegebenen Wert von $h_{lit} = 6.6261 \cdot 10^{-34}\text{J s}$ um 0.36σ ab. Diese Abweichung ist sehr gering und kann als nicht signifikant angesehen werden.

Am Ende dieses Aufgabenteils bestimmten wir noch den Winkel des Drehkristalls, ab dem das Spektrum zweiter Ordnung einsetzt. Hierfür kamen wir auf einen Wert von

$$\vartheta_{0,2.Ord} = (9.813 \pm 0.016)^\circ. \quad (32)$$

Im zweiten Versuchsteil betrachteten wir die Lagen der K_α - und K_β -Linien erster und zweiter Ordnung, welch durch Ionisation in der Molybdänanode erzeugt werden, genauer. Für alle vier Linien bestimmten wir mittels eines Fits einer Gaußkurve an die Daten die Winkelposition des Drehkristalls, an welcher diese auftreten, die zugehörige Standardabweichung, sowie deren Halbwertsbreite. Mithilfe des Bragg'schen Gesetzes berechneten wir aus den Winkeln wieder die zugehörigen Wellenlängen und bildeten einen Mittelwert über die Linien erster und zweiter Ordnung. Die Ergebnisse sind noch einmal in Tabelle (4) zusammengefasst. Die Literaturwerte für die Positionen der Linien stammen ebenfalls aus der Praktikumsanleitung.

Linie	Ordnung	λ [pm]	$\bar{\lambda}$ [pm]	λ_{Lit} [pm]	Abweichung
K_α	1	71.27 ± 0.19	71.17 ± 0.12	71.1	0.65σ
	2	71.07 ± 0.12			
K_β	1	63.27 ± 0.05	63.27 ± 0.14	63.1	1.29σ
	2	63.27 ± 0.10			

Wir können sehen, dass auch hier wieder die Abweichung sehr gering ausfällt.

Im darauf folgenden Versuchsteil betrachteten wir die Auswirkungen der an der Röntgenröhre anliegenden Beschleunigungsspannung genauer. Dazu stellten wir den Drehkristall auf einen festen Winkel von 7.5° und nahmen die Zählraten für Beschleunigungsspannungen im Bereich von 20 bis 35kV auf. Die Zählraten zeigten ab einem bestimmten Punkt einen mit der Spannung linearen Anstieg auf. Mittels einer Extrapolation ermittelten wir die Spannung, ab welcher die ersten Ereignisse gezählt wurde auf

$$U_0 = 22.872 \pm 1.025 \text{kV}. \quad (33)$$

Nach dem Bragg'schen Gesetz entsprach der eingestellte Winkel von 7.5° einer Wellenlänge von $(52.6 \pm 0.4)\text{pm}$. Mit diesen Werten konnten wir erneut einen Wert von

$$h = (6.43 \pm 0.30) \cdot 10^{-34} \text{ Js} \quad (34)$$

ermitteln. Dieser Wert weicht um 0.69σ vom Literaturwert aus der Praktikumsanleitung ab. Der berechnete Wert liegt zwar etwas näher am Literaturwert, allerdings haben wir nun einen kleineren Fehler, wodurch die größere σ -Abweichung zustande kommt.

Für den letzten Versuchsteil führten wir die Aufzeichnung des Röntgengenspektrums noch einmal mit einem NaCl-Kristall in einem Winkelbereich von 3° bis 18° durch. Dieser Bereich des Spektrums ist noch einmal in Abbildung (13) zu sehen. Aus den Daten ermittelten wir erneut die Winkelpositionen der K_α - und K_β -Linien erster Ordnung. Diese identifizierten wir mit den Wellenlängen der K_α - und K_β -Linien aus Versuchsteil 2. Damit konnten wir, erneut durch eine umgestellte Form des Bragg'schen Gesetzes, den Netzebenenabstand d berechnen.

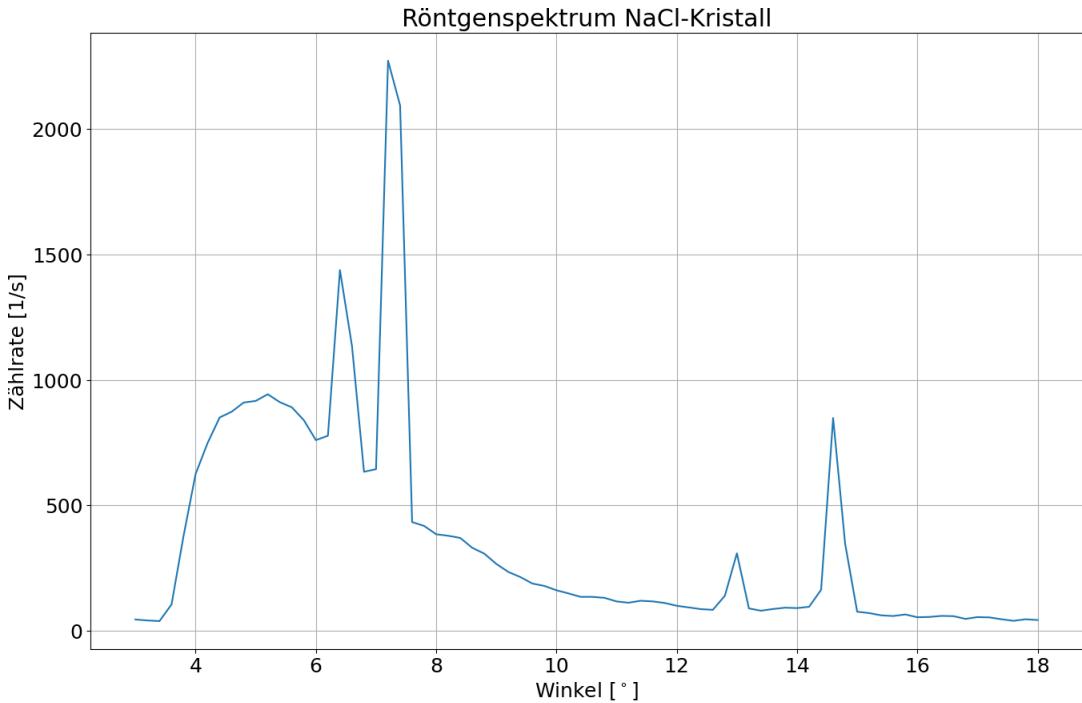


Abbildung 13: spektrumnaclkomplett

Die Ergebnisse sind noch einmal in der folgenden Tabelle (4) zusammengefasst.

Linie	Winkel °	λ [pm] (Aufg. 2)	d [pm]	\bar{d} [pm]
K_α	7.29 ± 0.05	71.17 ± 0.12	280.4 ± 1.8	280.8 ± 1.5
K_β	6.46 ± 0.05	63.27 ± 0.14	281.2 ± 2.3	

Der Netzebenenabstand d entspricht bei NaCl gerade der halben Gitterkonstante, somit konnten wir diese durch einfaches Verdoppeln des Wertes zu

$$a = (561.6 \pm 2.9)\text{pm} \quad (35)$$

berechnen. In der Literatur ist die Gitterkonstante von NaCl mit $a_{Lit} = 564\text{pm}^1$ angegeben. Der von uns berechnete Wert weicht von diesem um etwa 0.84σ ab.

Zu guter Letzt verwendeten wir den soeben berechneten Netzebenenabstand zur Berechnung der Avogadrokonstante. Hierbei kamen wir auf einen Wert von

$$N_A = (6.10 \pm 0.10) \cdot 10^{23} \frac{1}{\text{mol}}. \quad (36)$$

In der Praktikumsanleitung ist die Avogadrokonstante mit einem Wert von $N_{A,Lit} = 6.0221 \cdot 10^{23} \frac{1}{\text{mol}}$ angegeben. Unser Wert weicht von diesem um 0.82σ ab.

Allgemein können wir sehen, dass die Abweichungen der von uns berechneten Werte von den Literaturwerten in einem Bereich um 1σ bewegen, also sehr gering sind. Ein Grund für die dennoch auftretenden Abweichungen könnte zum einen dadurch zustande kommen, dass wir die Winkelbereiche immer noch in vergleichsweise großen Schritten aufgenommen haben. Somit fällt es schwer, die genaue Position der diskreten K -Linien zu ermitteln

¹<https://de.wikipedia.org/wiki/Natriumchlorid-Struktur>.

und auch die Fehlerabschätzung ist nur sehr grob möglich. Weiter ist zu bemerken, dass sich die K -Linien eigentlich, wie in der Einleitung erwähnt, noch durch eine Feinstrukturaufspaltung auszeichnen, welche eine Entartung der Drehimpuls- und Spinquantenzahl miteinbezieht. Diese haben wir in unseren Berechnungen nicht beachtet und auch einige der verglichenen Literaturwerte sind lediglich Mittelwerte über die Aufspaltung.

Python Code, Hauptprogramm

255auswertung

April 9, 2025

```
[37]: import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import numpy as np
from scipy.signal import argrelextrema
from scipy.optimize import curve_fit
from scipy.stats import chi2

plt.rcParams.update({'font.size': 18})

%run ../lib.ipynb
LibFormatter.OutputType = 'latex'

class Consts:
    hc = 1.2398 * 10**3 #nm eV
    E_Ry = -13.605 # eV
    E_inf = 3.2898 * 10**(15) # Hz
    c = 2.9979 * 10**(8) # m/s
    e = 1.6022 * 10**(-19) # C
    h = 6.6261 * 10**(-34) # Js
    N_A = 6.0221 * 10**(23)
```

```
[38]: def comma_to_float(valstr):
        return float(valstr.replace(',', '.'))

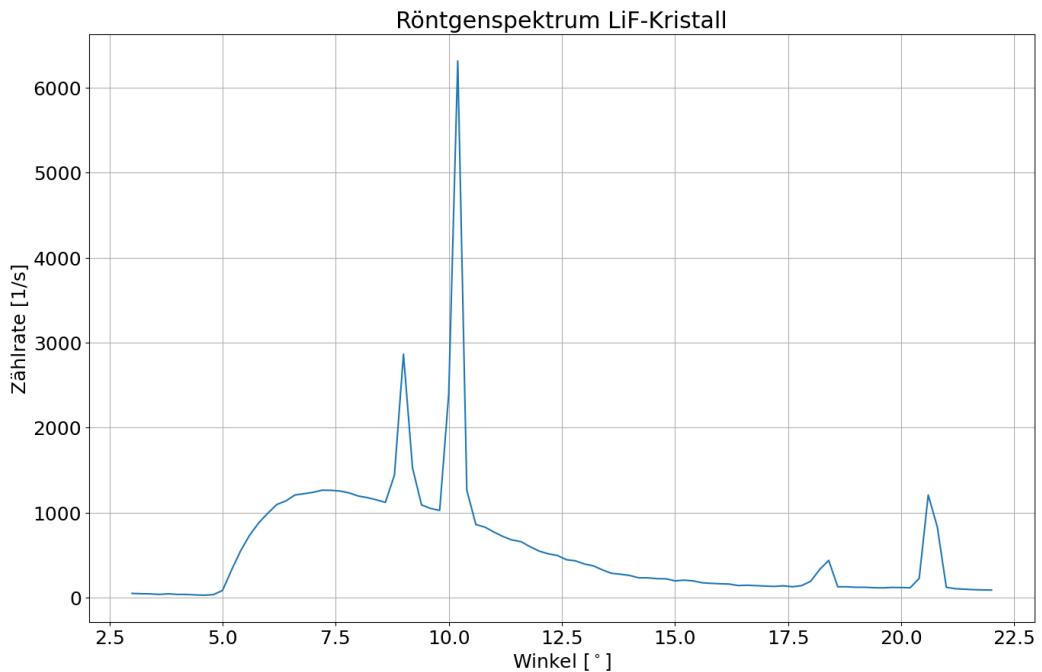
def countrate_error(ctrt, t):
    return [np.sqrt(x*t) / t for x in ctrt]
```

```
[39]: # fit functions
def gauss_func(x, A, mu, sig, c):
    return A/(np.sqrt(2 * np.pi)*sig)*np.exp(-(x-mu)**2 / (2 * sig**2)) + c

def linear_func(x, a, b):
    return a * x + b
```

```
[40]: lif_ang, lif_countrate = np.loadtxt('aufgabe1.txt', skiprows=0,
                                         converters= {0:comma_to_float, 1:comma_to_float},
                                         comments='>', unpack=True)
```

```
[41]: plt.figure(figsize=(16,10))
plt.plot(lif_ang, lif_countrate)
plt.title('Röntgenspektrum LiF-Kristall')
plt.xlabel(r'Winkel [${}^{\circ}$]')
plt.ylabel(r'Zählrate [1/s]')
plt.grid()
plt.savefig('out/spektrum_lif_komplett.png', format='png', bbox_inches='tight')
```



0.0.1 Aufgabe 1

```
[42]: plt.figure(figsize=(16,10))

lowlim = 10
uplim = 17

lif_countrate_errs = countrate_error(lif_countrate[lowlim:uplim], 5)

plt.errorbar(lif_ang[lowlim:uplim], lif_countrate[lowlim:uplim], yerr=lif_countrate_errs)
plt.title('Röntgenspektrum LiF - Fit an kurzwelliges Ende')
plt.xlabel(r'Winkel [${}^{\circ}$]')
plt.ylabel(r'Zählrate [1/s]')
#plt.ylim((0,3000))
```

```

plt.grid()

start_params = [0, 0]
popt_lif_lgrenz_lin, pcov_lif_lgrenz_lin = curve_fit(linear_func,
    lif_ang[lowlim:uplim], lif_countrate[lowlim:uplim], sigma =
    lif_countrate_errs, p0 = start_params)
lif_lin_a = ValErr.fromFit(popt_lif_lgrenz_lin, pcov_lif_lgrenz_lin, 0)
lif_lin_b = ValErr.fromFit(popt_lif_lgrenz_lin, pcov_lif_lgrenz_lin, 1)

# 0 = a x + b <=> x = -b/a
lif_lin_nullst = (-1) * (lif_lin_b / lif_lin_a)

plt.plot(lif_ang[lowlim:uplim], linear_func(lif_ang[lowlim:uplim],
    *popt_lif_lgrenz_lin))
plt.savefig('out/spektrum_lif_linear_fit.png', format='png',
    bbox_inches='tight')

plt.figure(figsize=(16,10))
plt.plot(lif_ang, lif_countrate)
plt.plot(lif_ang[lowlim-4:uplim+4], linear_func(lif_ang[lowlim-4:uplim+4],
    *popt_lif_lgrenz_lin))
plt.axvline(x=lif_lin_nullst.val, linewidth=2, linestyle='--', color='green')
plt.axvline(x=lif_lin_nullst.val+lif_lin_nullst.err, linewidth=1,
    linestyle='--', color="#56b300")
plt.axvline(x=lif_lin_nullst.val-lif_lin_nullst.err, linewidth=1,
    linestyle='--', color="#56b300")
plt.ylim((-100,3000))
plt.xlim((2.5,10))
plt.xlabel(r'Winkel [$^\circ$]')
plt.ylabel(r'Zählrate [1/s]')
plt.grid()

plt.savefig('out/spektrum_lif_linear_fit_wide.png', format='png',
    bbox_inches='tight')

# 2d sin(theta) = n lambda <=> lambda = 2d sin(theta) / n (n = 1)
# theta = arcsin (n lambda / 2 d)

lif_d = 201.4 * 10**(-12)

lif_lam_grenz_val = 2 * lif_d * np.sin(np.deg2rad(lif_lin_nullst.val))
lif_lam_grenz_err = 2 * lif_d * np.cos(np.deg2rad(lif_lin_nullst.val)) * np.
    deg2rad(lif_lin_nullst.err)
lif_lam_grenz = ValErr(lif_lam_grenz_val, lif_lam_grenz_err)

```

```

# l_grenz = hc / Ue <=> h = l_grenz U e / c
U_a1 = 35 * 10**3
h_a1 = lif_lam_grenz * (U_a1 * Consts.e / Consts.c)

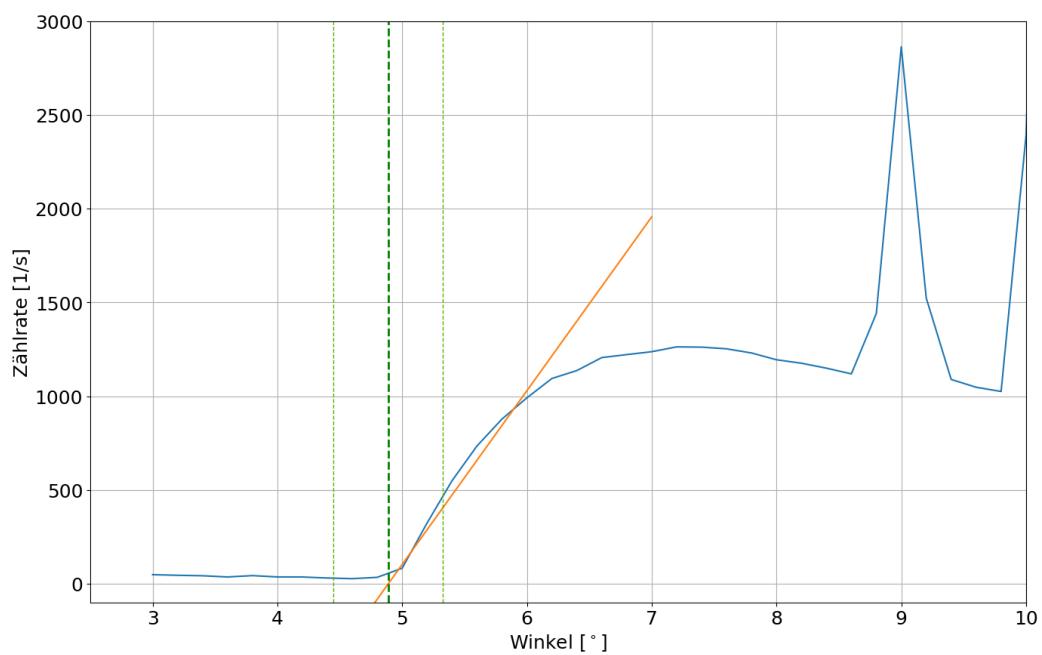
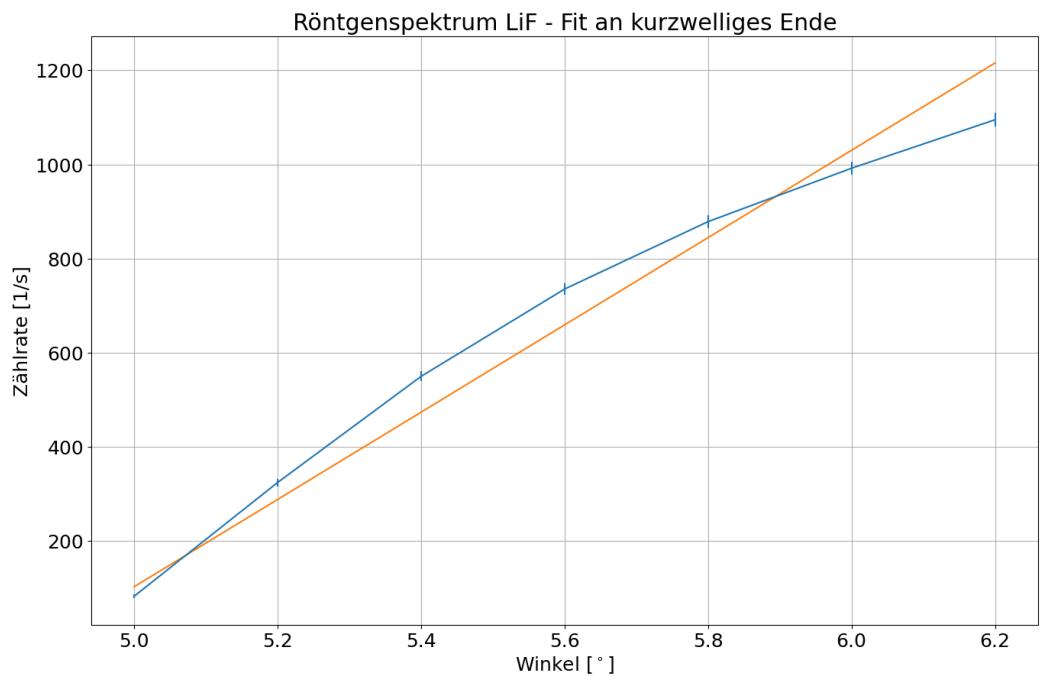
# theta(n=2) = arcsin(lambda/d)

lif_ang_2ord_val = np.rad2deg(np.arcsin(lif_lam_grenz.val / lif_d))
lif_ang_2ord_err = (lif_lam_grenz.err / lif_d) * (1 / np.sqrt(1 -_
    ↪(lif_lam_grenz.val**2/lif_d**2)))
lif_ang_2ord = ValErr(lif_ang_2ord_val, lif_ang_2ord_err)

print_all(
    lif_lin_a,
    lif_lin_b,
    lif_lin_nullst.strfmf2(6, 0, 'beta'),
    lif_lam_grenz.strfmf2(6, -12, '_grenz'),
    h_a1.strfmf2(6, -34, 'h'),
    h_a1.sigmadiff_fmt(ValErr(Consts.h, 0)),
    lif_ang_2ord.strfmf2(6, 0, 'beta, 2. ord'))

```

ValErr(926.9388981212614, 56.62078398042604)
 ValErr(-4531.531907160432, 297.4967190001228)
 beta = 4.888706 \pm 0.438383
 _grenz = (34.326828 \pm 3.070703) \cdot 10^{-12}
 h = (6.420980 \pm 0.574388) \cdot 10^{-34}
 0.36
 beta, 2. ord = 9.813464 \pm 0.015473



0.0.2 Aufgabe 2

```
[43]: lif_kalph1_ang, lif_kalph1_countrate = np.loadtxt('aufgabe2/kalpha1.txt',  

    ↪skiprows=0,  

    converters= {0:comma_to_float, 1:comma_to_float},  

    comments='>', unpack=True)  
  

lif_kbeta1_ang, lif_kbeta1_countrate = np.loadtxt('aufgabe2/kbeta1.txt',  

    ↪skiprows=0,  

    converters= {0:comma_to_float, 1:comma_to_float},  

    comments='>', unpack=True)  
  

lif_kalph2_ang, lif_kalph2_countrate = np.loadtxt('aufgabe2/kalpha2.txt',  

    ↪skiprows=0,  

    converters= {0:comma_to_float, 1:comma_to_float},  

    comments='>', unpack=True)  
  

lif_kbeta2_ang, lif_kbeta2_countrate = np.loadtxt('aufgabe2/kbeta2.txt',  

    ↪skiprows=0,  

    converters= {0:comma_to_float, 1:comma_to_float},  

    comments='>', unpack=True)  
  

def find_peak(peak_ang, peak_countrate, peak_order, offset, title, fname):  

    pf_lowlim = 0 + offset  

    pf_uplim = len(peak_ang) - offset  
  

    peak_countrate_errs = countrate_error(peak_countrate[pf_lowlim:pf_uplim],  

    ↪20)  
  

    plt.figure(figsize=(16,10))  

    plt.errorbar(peak_ang[pf_lowlim:pf_uplim], peak_countrate[pf_lowlim:  

    ↪pf_uplim], yerr=peak_countrate_errs)  

    plt.title(title)  

    plt.xlabel(r'Winkel [$^\circ$]')  

    plt.ylabel(r'Zählrate [1/s]')  

    plt.grid()  
  

    start_params = [500, np.mean(peak_ang[pf_lowlim:pf_uplim]), np.  

    ↪std(peak_ang[pf_lowlim:pf_uplim]), np.min(peak_countrate[pf_lowlim:  

    ↪pf_uplim])]  

    #print(start_params)  
  

    popt_peak, pcov_peak = curve_fit(  

        gauss_func,  

        peak_ang[pf_lowlim:pf_uplim],  

        peak_countrate[pf_lowlim:pf_uplim],  

        sigma = peak_countrate_errs,
```

```

    p0 = start_params)

peak_A = ValErr.fromFit(popt_peak, pcov_peak, 0)
peak_mu = ValErr.fromFit(popt_peak, pcov_peak, 1)
peak_sig = ValErr.fromFit(popt_peak, pcov_peak, 2)
peak_c = ValErr.fromFit(popt_peak, pcov_peak, 3)

peak_fwhm = 2 * np.sqrt(2 * np.log(2)) * peak_sig

print(peak_fwhm.val, peak_fwhm.val / 2, peak_sig.val, peak_sig.val / np.
      sqrt(len(peak_ang[pf_lowlim:pf_uplim])))

angle_err = peak_sig.val / np.sqrt(len(peak_ang[pf_lowlim:pf_uplim]))

contin_angle = np.arange(peak_ang[pf_lowlim], peak_ang[pf_uplim-1], 0.01)
plt.plot(contin_angle, gauss_func(contin_angle, *popt_peak))
# plt.axvline(x=peak_mu.val, linewidth=2, linestyle='--', color='green')
# plt.axvline(x=peak_mu.val + angle_err, linewidth=1, linestyle='--', color='red')
# plt.axvline(x=peak_mu.val - angle_err, linewidth=1, linestyle='--', color='red')

plt.savefig(f'out/{fname}.png', format='png', bbox_inches='tight')

peak_lam_val = (2 * 201.4 * 10**(-12) / peak_order) * np.sin(np.
      deg2rad(peak_mu.val))
peak_lam_err = (2 * 201.4 * 10**(-12) / peak_order) * np.cos(np.
      deg2rad(peak_mu.val)) * np.deg2rad(angle_err)
peak_lam = ValErr(peak_lam_val, peak_lam_err)

print_all(title,
          peak_A.strfmtf(4, 0, 'A'),
          peak_mu.strfmtf(4, 0, ''),
          peak_sig.strfmtf(4, 0, ''),
          peak_fwhm.strfmtf(4, 0, 'fwhm'),
          peak_lam.strfmtf2(6, -12, ''),
          '')

return (peak_A, peak_mu, peak_sig, peak_c, peak_lam)

peak_dat_kalpha1 = find_peak(lif_kalpha1_ang, lif_kalpha1_countrate, 1, 1,
      r'Röntgenspektrum (LiF), $K_{\alpha}$-Linie 1. Ordung', 'lif_kalpha_1ord')
peak_dat_kbeta1 = find_peak(lif_kbeta1_ang, lif_kbeta1_countrate, 1, 1,
      r'Röntgenspektrum (LiF), $K_{\beta}$-Linie 1. Ordung', 'lif_kbeta_1ord')

```

```

peak_dat_kalpha2 = find_peak(lif_kalph2_ang, lif_kalph2_countrate, 2, 0, □
    ↪r'Röntgenspektrum (LiF), $K_{\alpha}$-Linie 2. Ordung', 'lif_kalpha_2ord')
peak_dat_kbta2 = find_peak(lif_kbta2_ang, lif_kbta2_countrate, 2, 0, □
    ↪r'Röntgenspektrum (LiF), $K_{\beta}$-Linie 2. Ordung', 'lif_kbta_2ord')

kalpha_mean_lit = ValErr(71.1 * 10**(-12))
kbta_mean_lit = ValErr(63.1 * 10**(-12))

kalpha_mean = (peak_dat_kalpha1[4] + peak_dat_kalpha2[4]) / 2
kbta_mean = (peak_dat_kbta1[4] + peak_dat_kbta2[4]) / 2

print_all('Mittelwerte', kalpha_mean.strfmtf2(6, -12, ' K_'), kalpha_mean.
    ↪sigmadiff_fmt(kalpha_mean_lit), kbta_mean.strfmtf2(6, -12, ' K_'), □
    ↪kbta_mean.sigmadiff_fmt(kbta_mean_lit))

```

0.2449426466010601 0.12247132330053005 0.1040175647892622 0.0268572197427909

Röntgenspektrum (LiF), \$K_{\alpha}\$-Linie 1. Ordung
A = 1417.9660 \pm 64.8681
= 10.1912 \pm 0.0049
= 0.1040 \pm 0.0045
fwhm = 0.2449 \pm 0.0105
= (71.268530 \pm 0.185832) \cdot 10^{-12}

0.2598405224074471 0.12992026120372355 0.11034411013943615 0.03678137004647872

Röntgenspektrum (LiF), \$K_{\beta}\$-Linie 1. Ordung
A = 503.0489 \pm 37.1957
= 9.0377 \pm 0.0064
= 0.1103 \pm 0.0075
fwhm = 0.2598 \pm 0.0176
= (63.273447 \pm 0.255370) \cdot 10^{-12}

0.29438118966863097 0.14719059483431549 0.12501218099014522 0.03608790817332128

Röntgenspektrum (LiF), \$K_{\alpha}\$-Linie 2. Ordung
A = 390.5916 \pm 10.7041
= 20.6648 \pm 0.0033
= 0.1250 \pm 0.0031
fwhm = 0.2944 \pm 0.0074
= (71.074234 \pm 0.118691) \cdot 10^{-12}

0.2556687909958171 0.12783439549790854 0.10857253892301431 0.028033309006952733

Röntgenspektrum (LiF), \$K_{\beta}\$-Linie 2. Ordung
A = 93.8456 \pm 9.1049
= 18.3110 \pm 0.0106
= 0.1086 \pm 0.0104
fwhm = 0.2557 \pm 0.0246
= (63.274851 \pm 0.093550) \cdot 10^{-12}

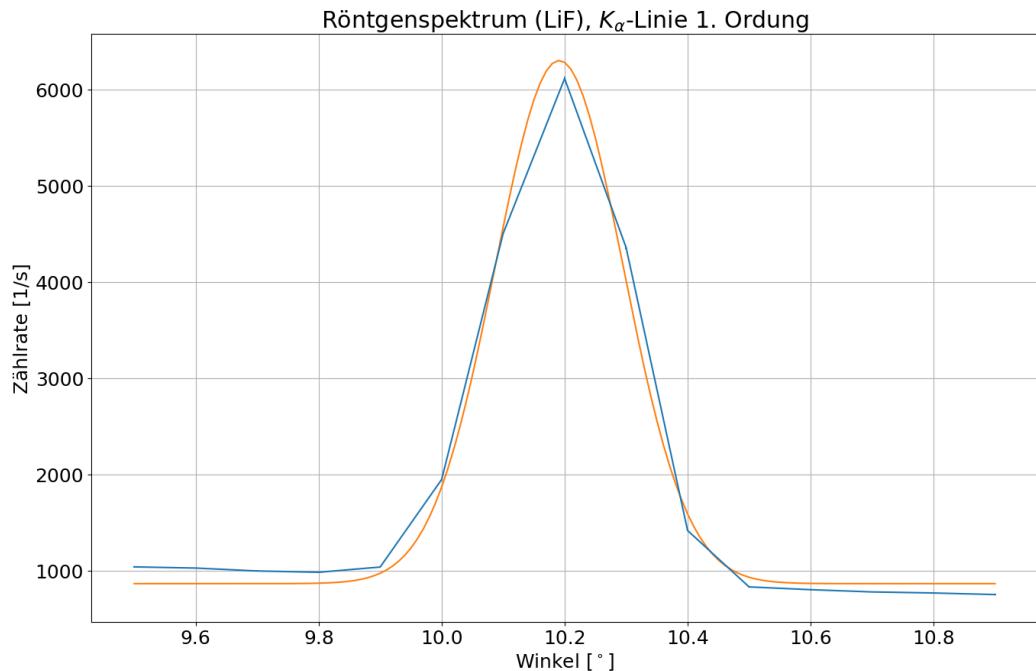
Mittelwerte

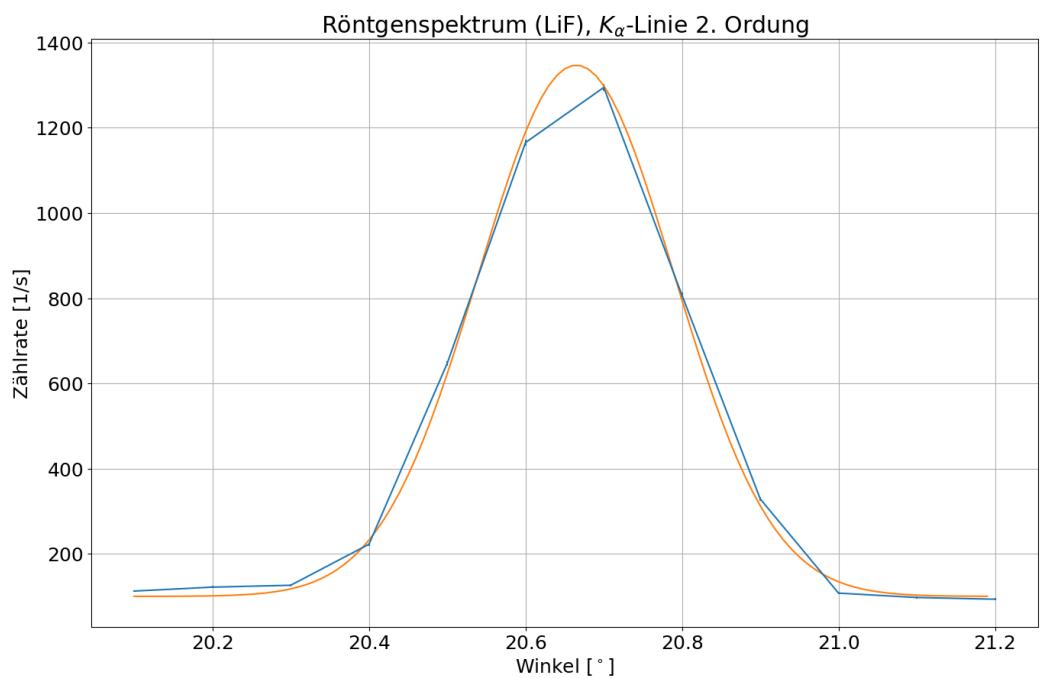
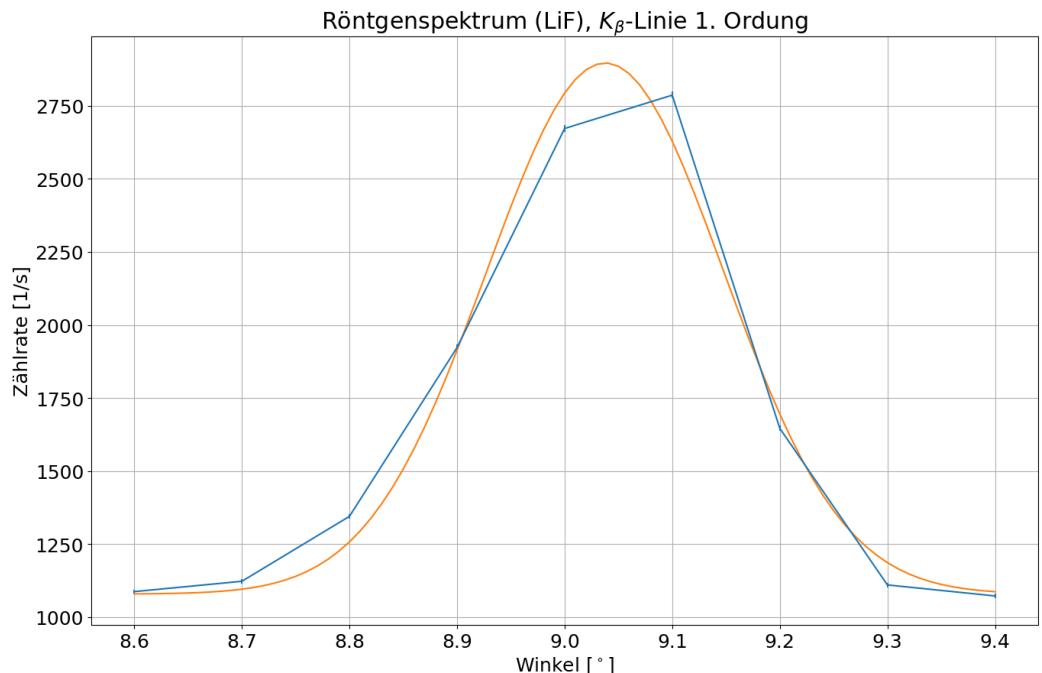
$$K_{\alpha} = (71.171382 \pm 0.110251) \cdot 10^{-12}$$

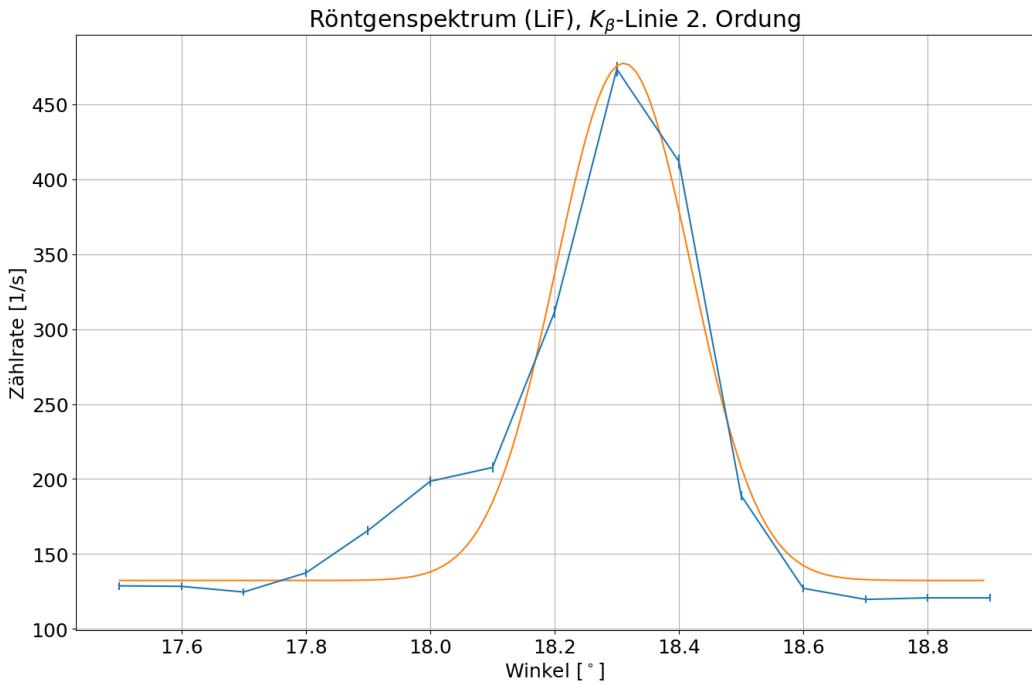
0.65

$$K_{\beta} = (63.274149 \pm 0.135983) \cdot 10^{-12}$$

1.29







0.0.3 Aufgabe 3

```
[44]: lif_ang75_volt, lif_ang75_countrate = np.loadtxt('aufgabe3.txt', skiprows=0,
                                                     converters= {0:comma_to_float, 1:comma_to_float},
                                                     comments='>', unpack=True)

[45]: lowlim_einsp = 3
uplin_einsp = len(lif_ang75_volt)

lif_ang75_countrate_errs = countrate_error(lif_ang75_countrate, 20)

plt.figure(figsize=(16,10))
plt.errorbar(lif_ang75_volt, lif_ang75_countrate, yerr=lif_ang75_countrate_errs)
#plt.title(title)
plt.xlabel(r'Spannung [kV]')
plt.ylabel(r'Zählrate [1/s]')
plt.grid()

start_params = [0, 0]
popt_lif_einsp, pcov_lif_einsp = curve_fit(
    linear_func,
    lif_ang75_volt[lowlim_einsp:uplin_einsp],
    lif_ang75_countrate[lowlim_einsp:uplin_einsp],
```

```

sigma = lif_ang75_countrate_errs[lowlim_einsp:uplin_einsp],
p0 = start_params)

lif_einsp_a = ValErr.fromFit(popt_lif_einsp, pcov_lif_einsp, 0)
lif_einsp_b = ValErr.fromFit(popt_lif_einsp, pcov_lif_einsp, 1)

plt.plot(lif_ang75_volt, linear_func(lif_ang75_volt, *popt_lif_einsp))

#  $0 = a \cdot x + b \Leftrightarrow x = -b/a$ 
lif_einsp_nullst = (-1) * (lif_einsp_b / lif_einsp_a)

plt.axvline(x=lif_einsp_nullst.val, linewidth=2, linestyle='--', color='green')
plt.axvline(x=lif_einsp_nullst.val+lif_einsp_nullst.err, linewidth=1, color='red')
plt.axvline(x=lif_einsp_nullst.val-lif_einsp_nullst.err, linewidth=1, color='red')

plt.savefig(f'out/lif_ang75_volt_fit.png', format='png', bbox_inches='tight')

#  $2d \sin(\theta) = n \lambda \Leftrightarrow \lambda = 2d \sin(\theta) / n \ (n = 1)$ 

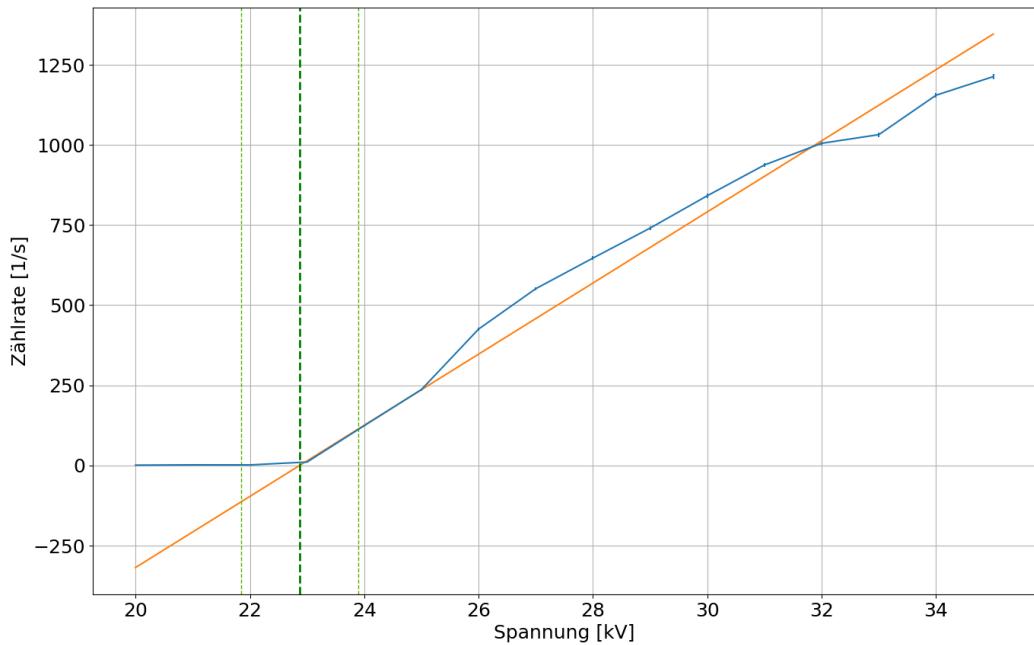
lif_einsp_lam_val = 2 * 201.4 * 10**(-12) * np.sin(np.deg2rad(7.5))
lif_einsp_lam_err = 2 * 201.4 * 10**(-12) * np.cos(np.deg2rad(7.5)) * np.deg2rad(0.05)
lif_einsp_lam = ValErr(lif_einsp_lam_val, lif_einsp_lam_err)

#  $h = hc / Ue \Leftrightarrow h = l_{grenz} U e / c$ 
U_a3 = lif_einsp_nullst * 10**3
h_a3 = lif_einsp_lam * (U_a3 * Consts.e / Consts.c)

print_all(lif_einsp_a,
          lif_einsp_b,
          lif_einsp_nullst.strfmtf2(6, 0, 'U'),
          lif_einsp_lam.strfmtf2(6, -12, ' '),
          h_a3.strfmtf2(6, -34, 'h'),
          h_a3.sigmadiff_fmt(ValErr(Consts.h, 0)))

ValErr(110.96963341748018, 3.434413817817252)
ValErr(-2538.133262327901, 82.21087923807104)
U = 22.872323 \pm 1.024665
= (52.575950 \pm 0.348502) \cdot 10^{-12}
h = (6.426833 \pm 0.291052) \cdot 10^{-34}
0.69

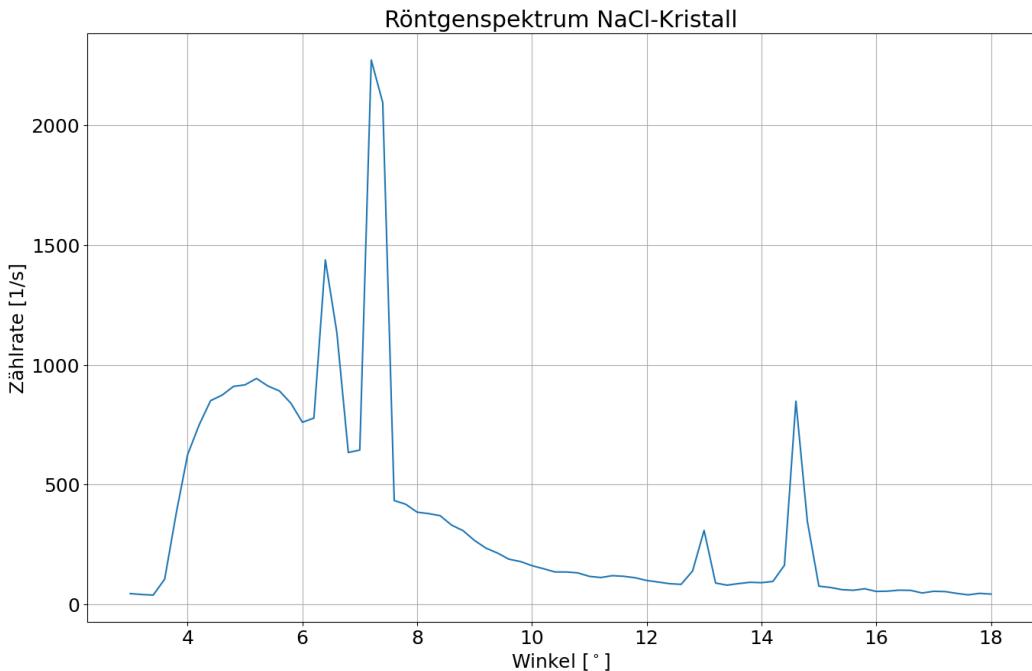
```



0.0.4 Aufgabe 4

```
[46]: nacl_ang, nacl_countrate = np.loadtxt('aufgabe4.txt', skiprows=0,
                                             converters= {0:comma_to_float, 1:comma_to_float},
                                             comments='>', unpack=True)

plt.figure(figsize=(16,10))
plt.plot(nacl_ang, nacl_countrate)
plt.title(r'Röntgenspektrum NaCl-Kristall')
plt.xlabel(r'Winkel [ $\circ$ ])
plt.ylabel(r'Zählrate [1/s]')
plt.grid()
plt.savefig('out/spektrum_nacl_komplett.png', format='png', bbox_inches='tight')
```



```
[47]: nacl_lowlim = 15
nacl_uplim = 25

plt.figure(figsize=(16,10))
plt.plot(nacl_ang[nacl_lowlim:nacl_uplim], nacl_countrate[nacl_lowlim:
    ↪nacl_uplim])
plt.title(r'Röntgenspektrum NaCl, $K_{\beta}, K_{\alpha}$ 1. Ordnung')
plt.xlabel(r'Winkel [$^\circ$]')
plt.ylabel(r'Zählrate [1/s]')
plt.xticks(np.arange(nacl_ang[nacl_lowlim], nacl_ang[nacl_uplim-1], 0.1))
plt.grid()
# plt.savefig('out/himmel_m_o_g.png', format='png', bbox_inches='tight')

def find_nacl_peak(nacl_peak_low, nacl_peak_up):
    peak_countrate_errs = countrate_error(nacl_countrate[nacl_peak_low:
        ↪nacl_peak_up], 5)

    start_params = [500, np.mean(nacl_ang[nacl_peak_low:nacl_peak_up]), np.
        ↪std(nacl_ang[nacl_peak_low:nacl_peak_up]), 500]

    popt_nacl_k_peak, pcov_nacl_k_peak = curve_fit(
        gauss_func,
```

```

nacl_ang[nacl_peak_low:nacl_peak_up],
nacl_countrate[nacl_peak_low:nacl_peak_up],
sigma=peak_countrate_errs,
p0 = start_params)

contin_angle = np.arange(nacl_ang[nacl_peak_low], nacl_ang[nacl_peak_up-1], 0.01)
plt.plot(contin_angle, gauss_func(contin_angle, *popt_nacl_k_peak))

peak_mu = ValErr.fromFit(popt_nacl_k_peak, pcov_nacl_k_peak, 1)
peak_sig = ValErr.fromFit(popt_nacl_k_peak, pcov_nacl_k_peak, 2)

angle_err = peak_sig.val / np.sqrt(len(nacl_ang[nacl_peak_low:nacl_peak_up]))
# plt.axvline(x=peak_mu.val, linewidth=2, linestyle='--', color='green')
# plt.axvline(x=peak_mu.val + angle_err, linewidth=1, linestyle='--', color="#56b300")
# plt.axvline(x=peak_mu.val - angle_err, linewidth=1, linestyle='--', color="#56b300")

return ValErr(peak_mu.val, angle_err)

nacl_kbetal_angle = find_nacl_peak(15, 21)
nacl_kalpatal_angle = find_nacl_peak(19, 25)

plt.savefig(f'out/nacl_k_1ord_fit.png', format='png', bbox_inches='tight')

# 2d sin() = n    <=> d = n / 2 sin()

def calc_d(lam, ang):
    lam_val = lam.val
    lam_err = lam.err

    ang_val = np.deg2rad(ang.val)
    ang_err = np.deg2rad(ang.err)

    sin_ang_val = np.sin(ang_val)
    cos_ang_val = np.cos(ang_val)

    d_val = (1/2) * (lam_val / sin_ang_val)
    d_err = (1/2) * np.sqrt((lam_err / sin_ang_val) ** 2 + ((lam_val * cos_ang_val * ang_err) / sin_ang_val ** 2) ** 2)
    return ValErr(d_val, d_err)

nacl_d_kbetal = calc_d(kbeta_mean, nacl_kbetal_angle)
nacl_d_kalpatal = calc_d(kalpatal_mean, nacl_kalpatal_angle)

```

```

nacl_d_mean = (nacl_d_kbta + nacl_d_kalpha) / 2

nacl_M_Mol = 58.44 # g
nacl_rho = 2.164 * 100**3 # g / m^3

# N_A = 1/2 * M_{Mol} / d^3

N_A_val = (1/2) * (nacl_M_Mol / nacl_rho) * (1 / nacl_d_mean.val**3)
N_A_err = (1/2) * (nacl_M_Mol / nacl_rho) * (3 * nacl_d_mean.err / nacl_d_mean.
    ↪val**4)
N_A = ValErr(N_A_val, N_A_err)

# NaCL a, https://de.wikipedia.org/wiki/Natriumchlorid-Struktur
nacl_a_lit = 0.564 * 10**(-9)

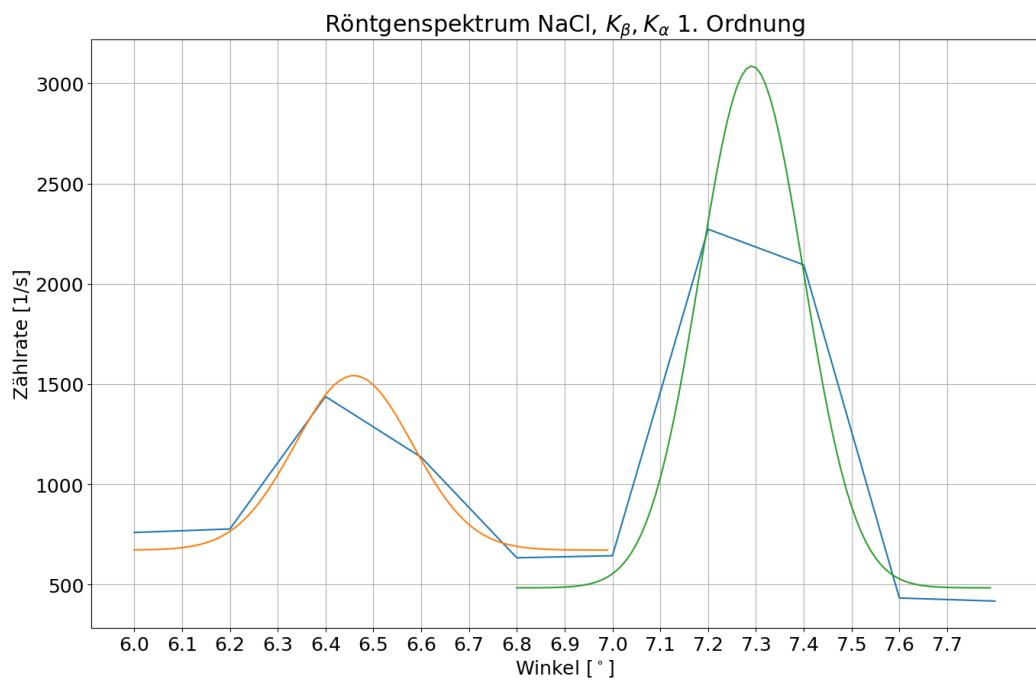
print_all(
    kalpha_mean.strfmf2(6, -12, ' ,K_ '),
    kbta_mean.strfmf2(6, -12, ' ,K_ '), '',
    nacl_kalpha1_angle.strfmf2(6, 0, ' ,K_ '),
    nacl_kbta1_angle.strfmf2(6, 0, ' ,K_ '), '',
    nacl_d_kalpha.strfmf2(6, -12, 'd,K_ '),
    nacl_d_kbta.strfmf2(6, -12, 'd,K_ '),
    nacl_d_mean.strfmf2(6, -12, 'd'),
    (nacl_d_mean * 2).strfmf2(6, -12, 'a'),
    (nacl_d_mean * 2).sigmadiff_fmt(ValErr(nacl_a_lit, 0)),
    N_A.strfmf2(6, 23, 'N_A'),
    N_A.sigmadiff_fmt(ValErr(Consts.N_A, 0)))

,K_ = (71.171382 \pm 0.110251) \cdot 10^{-12}
,K_ = (63.274149 \pm 0.135983) \cdot 10^{-12}

,K_ = 7.291215 \pm 0.044213
,K_ = 6.459243 \pm 0.050100

d,K_ = (280.395432 \pm 1.746003) \cdot 10^{-12}
d,K_ = (281.227410 \pm 2.254563) \cdot 10^{-12}
d = (280.811421 \pm 1.425796) \cdot 10^{-12}
a = (561.622843 \pm 2.851593) \cdot 10^{-12}
0.84
N_A = (6.097877 \pm 0.092884) \cdot 10^{23}
0.82

```



[]:

Python Code, Bibliothek

lib

April 9, 2025

```
[51]: def floatfmt(v, prec, exp):
    return f"{v/10**exp}:0={prec}f}{LibFormatter.exp10(exp) if exp != 0 else"
           ↵' '}""

def prec_ceil(v, prec=0):
    return np.true_divide(np.ceil(v * 10**prec), 10**prec)

def prec_floor(v, prec=0):
    return np.true_divide(np.floor(v * 10**prec), 10**prec)

[57]: class LibFormatter:
    OutputType = 'text'

    @classmethod
    def exp10(self, exp):
        if LibFormatter.OutputType == 'latex':
            return f' \cdot 10^{{exp}}'
        elif LibFormatter.OutputType == 'text':
            return f'e{exp}'
        else:
            raise ValueError(f"Unsupported OutputType: '{LibFormatter."
                           ↵OutputType}'")

    @classmethod
    def pm(self):
        if LibFormatter.OutputType == 'latex':
            return f'\pm'
        elif LibFormatter.OutputType == 'text':
            return f'±'
        else:
            raise ValueError(f"Unsupported OutputType: '{LibFormatter."
                           ↵OutputType}'")

    @classmethod
    def sigma(self):
        if LibFormatter.OutputType == 'latex':
            return f'\sigma'
```

```

    elif LibFormatter.OutputType == 'text':
        return f' '
    else:
        raise ValueError(f"Unsupported OutputType: '{LibFormatter.
        ↪OutputType}'")

```

```

[2]: import math
import numpy as np

class ValErr:
    val: float = 0
    err: float = 0
    err_set = False

    def __init__(self, val, err=0):
        self.val = val
        if err != 0:
            self.err_set = True
            self.err = err

    def getTuple(self):
        return (self.val, self.err)

    def setErr(self, err_value):
        self.err_set = True
        self.err = err_value

    @classmethod
    def fromMeasurements(self, measurements):
        return ValErr(np.mean(measurements), (1 / math.sqrt(len(measurements))) ↪
        ↪* np.std(measurements, ddof=1))

    @classmethod
    def fromTuple(self, tup):
        return ValErr(tup[0], tup[1])

    @classmethod
    def fromFit(self, popt, pcov, i):
        return ValErr(popt[i], np.sqrt(pcov[i][i]))

    @classmethod
    def fromFitAll(self, popt, pcov):
        for i in range(0, len(popt)):
            yield ValErr(popt[i], np.sqrt(pcov[i][i]))

    @classmethod
    def fromValPerc(self, v, perc):

```

```

    return ValErr(v, v * perc/100)

def strfmt(self, prec=2):
    if self.err != 0:
        return fr"{{self.val:.{prec}e} {LibFormatter.pm()} {self.err:.{prec}e}}"
    else:
        return f"{{self.val:.{prec}e}}"

def strfmtf(self, prec, exp, name = ""):
    prefix = ""
    if name != "":
        prefix = f"{{name}} = "

    if self.err != 0:
        return prefix + fr"{{floatfmt(self.val, prec, exp)} {LibFormatter.
pm()} {{floatfmt(self.err, prec, exp)}}}"
    else:
        return prefix + f"{{floatfmt(self.val, prec, exp)}}"

def strfmtf2(self, prec, exp, name = ""):
    prefix = ""
    if name != "":
        prefix = f"{{name}} = "

    if self.err != 0:
        return prefix + fr"{{f'({if exp != 0 else ''}{self.val/10**exp}:
{0=1.{prec}f} {LibFormatter.pm()} {self.err/10**exp}:0=1.
{prec}f}{f')}{LibFormatter.exp10(exp)}' if exp != 0 else ''}}"
    else:
        return prefix + f"{{floatfmt(self.val, prec, exp)}}"

def strltx(self, prec=2):
    if self.err != 0:
        return fr"{{self.val:.{prec}e} \pm {self.err:.{prec}e}}"
    else:
        return f"{{self.val}}"

def relerr(self):
    return self.err / self.val

def sigmadiff(self, other):
    return np.abs(self.val - other.val) / np.sqrt(self.err**2 + other.
err**2)

def sigmadiff_fmt(self, other, prec=2):
    return f"{{prec_ceil(sigmadiff(other), prec)}{LibFormatter.sigma()}}"

```

```

def pow(self, p):
    return ValErr(self.val**2, 2 * self.val * self.err)

def __repr__(self):
    return f"ValErr({self.val}, {self.err})"

def __radd__(self, other):
    return self.__add__(other)

def __add__(self, other):
    if isinstance(other, self.__class__):
        return ValErr(self.val + other.val, math.sqrt(self.err**2 + other.
        ↪err**2))
    elif isinstance(other, float) or isinstance(other, int):
        return ValErr(self.val + other, self.err)
    else:
        raise TypeError(f"unsupported operand type(s) for +: '{self.
        ↪__class__}' and '{type(other)}'")

def __rsub__(self, other):
    if isinstance(other, self.__class__):
        return ValErr(other.val - self.val, math.sqrt(other.err**2 + self.
        ↪err**2))
    elif isinstance(other, float) or isinstance(other, int):
        return ValErr(other - self.val, self.err)
    else:
        raise TypeError(f"unsupported operand type(s) for +: '{self.
        ↪__class__}' and '{type(other)}'")

def __sub__(self, other):
    if isinstance(other, self.__class__):
        return ValErr(self.val - other.val, math.sqrt(self.err**2 + other.
        ↪err**2))
    elif isinstance(other, float) or isinstance(other, int):
        return ValErr(self.val - other, self.err)
    else:
        raise TypeError(f"unsupported operand type(s) for +: '{self.
        ↪__class__}' and '{type(other)}'")

def __rmul__(self, other):
    return self.__mul__(other)

def __mul__(self, other):
    if isinstance(other, self.__class__):

```

```

        return ValErr(self.val * other.val, math.sqrt((other.val * self.
↳err)**2 + (self.val * other.err)**2))
    elif isinstance(other, float) or isinstance(other, int):
        return ValErr(self.val * other, self.err * np.abs(other))
    else:
        raise TypeError(f"unsupported operand type(s) for +: '{self.
↳__class__}' and '{type(other)}'")

def __rtruediv__(self, other):
    if isinstance(other, self.__class__):
        return ValErr(other.val / self.val, math.sqrt((other.err / self.
↳val)**2 + (other.val * self.err / self.val**2)**2))
    elif isinstance(other, float) or isinstance(other, int):
        return ValErr(other / self.val, np.abs(other / self.val**2) * self.
↳err)
    else:
        raise TypeError(f"unsupported operand type(s) for +: '{self.
↳__class__}' and '{type(other)}'")

def __truediv__(self, other):
    if isinstance(other, self.__class__):
        return ValErr(self.val / other.val, math.sqrt((self.err / other.
↳val)**2 + (self.val * other.err / other.val**2)**2))
    elif isinstance(other, float) or isinstance(other, int):
        return ValErr(self.val / other, self.err / other)
    else:
        raise TypeError(f"unsupported operand type(s) for +: '{self.
↳__class__}' and '{type(other)}'")

```

[54]: def spacearound(dat, add):
 return np.linspace(dat[0] - add, dat[len(dat)-1] + add)

[55]: def div_with_err(a, a_err, b, b_err):
 err = (1 / b) * np.sqrt(a_err**2 + (a * b_err / b)**2)
 return (a / b, err)

[56]: def print_all(*args):
 for e in args:
 print(e)

[]: