Festkörperphysik Student Project

Dispersionsrelation und Zustandsdichte von Phononen im bcc-Gitter

Wintersemester 2011/12

Richard Meister, Christian Kokail

Inhaltsverzeichnis

1 Grundlegendes

In diesem Student Project im Rahmen der Vorlesung *Molecular and Solid State Physics* soll die Zustandsdichte (Density of States) und die Dispersionsrelation von Phononen in einem bcc-Kristall mit einatomiger Basis unter Berücksichtigung der nächst-nächsten-Nachbaratome berechnet werden. Frühere Berechnungen von Studenten zeigten, dass die DoS bei bloßer Berücksichtigung der nächsten Nachbarn im Bereich kleiner Frequenzen einen linearen Verlauf aufweist, obwohl im Limes langer Wellenlängen eigentlich ein quadratischer Zusammenhang zwischen k und ω zu erwarten wäre. Es soll geprüft werden, ob in den alten Berechnungen eventuell ein Fehler vorliegt, und ob sich durch Berücksichtigung der nächst-nächsten-Nachbaratome hier eine Verbesserung beobachten lässt. Weiters wird das geschriebene MATLAB-Programm im Detail beschrieben, damit Studenten späterer Jahrgänge die hier gewonnen Erkenntnisse wiederverwenden können.

Zur Berechnung der DoS werden, wie in der Vorlesung beschrieben, alle Atome mit ihren Nachbarn durch lineare Federn verbunden und daraus die Kräfte bestimmt, die bei der Auslenkung der Atome aus der Gleichgewichtslage auftreten. Aus diesen Kräften können dann die Bewegungsgleichungen für alle 3 Raumrichtungen aufgestellt werden. Danach macht man den in der Vorlesung beschriebenen Normalmoden-Ansatz (Eigenfunktionen des Translationsoperators). Die Bewegungs-Differentialgleichungen werden so zu algebraischen Gleichungen und lassen sich dann in Matrix-Form schreiben, wobei die Eigenwerte dieser Matrix die Eigenfrequenzen für die Longitudinal- und die Transversalmoden darstellen. Bei nur einer Atomsorte existieren lediglich 3 Eigenwerte (2 Transversal-, 1 Longitudinalmode). Die Eigenwerte der Matrix werden im Rahmen eines MATLAB-Programms für unterschiedliche k-Vektoren innerhalb der ersten Brillouin-Zone berechnet. Dazu ist es wichtig, die geometrische Form und die Symmetriepunkte der ersten Brillouin-Zone zu kennen.

Nachfolgend sind die Symmetriepunkte der ersten Brillouin-Zone des bcc-Gitters in einer Tabelle zusammengefasst. Die k-Vektoren sind durch $\vec{k} = u\vec{b}_1 + v\vec{b}_2 + w\vec{b}_3$ gegeben, wobei $\vec{b}_1$, $\vec{b}_2$ und $\vec{b}_3$ die reziproken Gittervektoren sind (siehe Abbildung 1).

Tabelle 1: Symmetriepunkte der ersten Brillouin-Zone des bcc-Gitters

Symmetriepunkt	(u,v,w)	$[k_x,k_y,k_z]$
Γ	(0,0,0)	[0,0,0]
H	(1/2,1/2,1/2)	$[0,0,2\pi/a]$
P	(1/4,1/4,1/4)	$[\pi/a,\pi/a,\pi/a]$
N	(0,1/2,0)	$[\pi/a,\pi/a,0]$

Der bcc-Kristall mit den nächsten Nachbarn ist in Abbildung 2 dargestellt. Wie zu erkennen ist, gibt es 8 nächste Nachbarn, welche die Ecken des Würfels mit der Gitterkonstante a darstellen. Jedes der Atome in einer Würfelecke ist auch ein Bestandteil der Elementarzelle von 7 weiteren angrenzenden Elementarzellen. Dementsprechend befinden sich im bcc-Gitter 2 Atome in einer Elementarzelle.

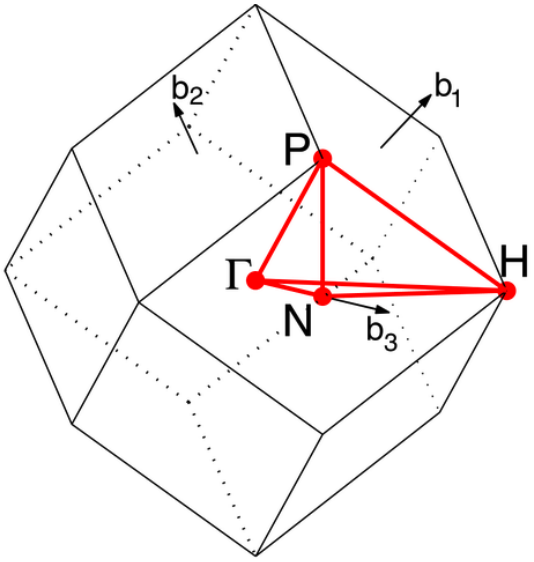

Abbildung 1: Erste Brillouin-Zone des bcc-Gitters

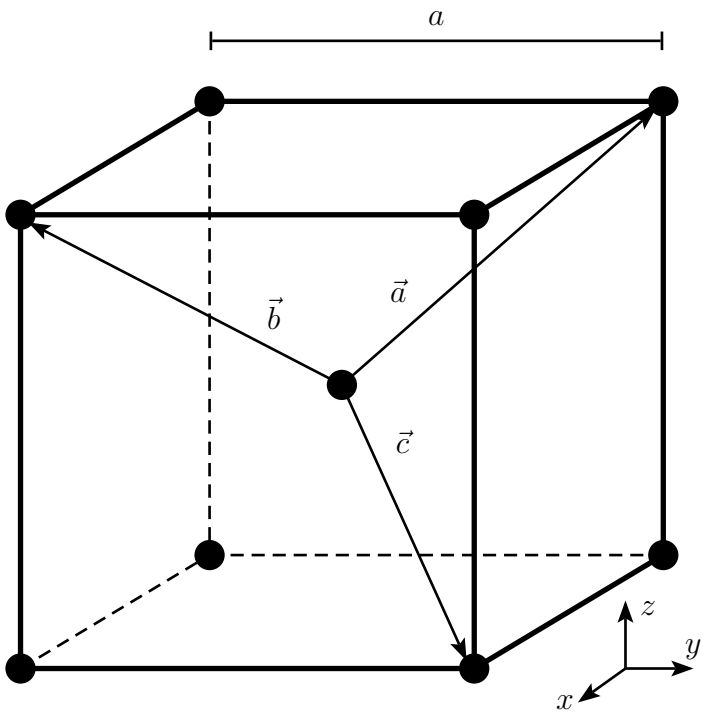

Abbildung 2: Skizze der konventionellen Einheitszelle des bcc-Gitters. Der Koordinatenursprung ist das mittlere Atom.

Die Anzahl der Atome in der Elementarzelle hat direkten Einfluss auf die Normierung der DoS. Ein Kristall aus N Atomen hat $3N$ Freiheitsgrade (x-, y-, und z-Position jedes Atoms). Um die Positionen aller Atome eindeutig festzulegen, braucht man also $3N$ Bedingungen. Diese kann man nicht nur in Form von x-, y-, und z-Koordinaten, sondern auch durch Angabe von $3N$ Schwingungsmoden festlegen. Weil die DoS

immer pro Volumeneinheit angegeben ist, muss die Fläche unter der Zustandsdichte also $3n$ sein, wobei $n = N/l^3$ die Atomdichte, also die Anzahl der Atome pro Volumenheinheit, ist.

$$\int_0^\infty D(\omega)d\omega = 3n$$

Nun wissen wir, dass in einer Einheitszelle 2 Atome enthalten sind, und die Anzahl der Einheitszellen pro Volumeneinheit a^{-3} ist. Daraus folgt sofort

$$n = \frac{2}{a^3}$$

und damit

$$\int_0^\infty D(\omega)d\omega = \frac{6}{a^3} \tag{1.1}$$

Später wird sich zeigen, dass es sinnvoll ist, reduzierte Einheiten zu verwenden. Anstelle von ω wird auf der x-Achse die dimensionslose Einheit $\omega\sqrt{\frac{m}{C}}$ aufgetragen. Damit ergibt sich für die Normierung der Zustandsdichte

$$\int_0^\infty D(\omega)d\left(\omega\sqrt{\frac{m}{C}}\right) = \sqrt{\frac{m}{C}}\int_0^\infty D(\omega)d\omega = \sqrt{\frac{m}{C}}\frac{6}{a^3} \tag{1.2}$$

Aus diesem Grund wird im Programm anstelle der gewöhnlichen DoS die ebenfalls dimensionslose Größe

$$D(\omega)\sqrt{\frac{C}{m}}a^3$$

aufgetragen. Diese Größe ist nun immer auf 6 normiert, wie man an Gleichung 1.2 sieht.

Zusätzlich sind in Abbildung 2 die primitiven Gittervektoren abgebildet. Hierfür gilt

$$\vec{a} = \frac{a}{2}\begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix}, \quad \vec{b} = \frac{a}{2}\begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}, \quad \vec{c} = \frac{a}{2}\begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} \tag{1.3}$$

Gemäß diesen Gittervektoren erfolgt die Indizierung der Atompositionen durch drei Indizes der Form (l,m,n). Die Gleichgewichtsposition eines Atoms ist damit durch $\vec{x} = l\vec{a} + m\vec{b} + n\vec{c}$ gegeben. Ausgangspunkt unserer Rechnung ist ein beliebiges Atom mit den Indizes (l,m,n). Das Atom, dessen Position um den Vektor $\vec{a}$ gegenüber dem Betrachteten verschoben ist, das wäre in Abbildung 2 das Atom rechts oben hinten, hat die Indizes $(l+1,m,n)$, usw...

2 Analytische Rechnung

2.1 Kräfte der Nachbarn

Um die Bewegungen der Atome zu beschreiben, verwenden wir die Funktionen $u^{\alpha}_{l,m,n}(t)$, die die Auslenkung des Atoms mit den Indizes (l,m,n) aus der Gleichgewichtslage in die α-Richtung ($\alpha = x, y, z$) angeben. Für die Bewegungsgleichungen müssen wir nun berechnen, welche Kräfte abhängig von der Auslenkung aus der Ruhelage auf die Atome wirken. Zunächst betrachten wir nur die nächsten Nachbarn. Wir verbinden jedes Atom mit seinen nächsten Nachbarn über Federn der Federkonstante C und berechnen die Kräfte in Abhängigkeit von der Auslenkung.

Dabei sind zwei wichtige Dinge zu beachten. Erstens wird die Feder zwischen zwei Atomen bei z.B. einer Bewegung in x-Richtung nicht um den Betrag der Auslenkung gestreckt oder gestaucht, weil die Feder schräg im Raum liegt. Zweitens wirkt die aus der Streckung oder Verkürzung resultierende Kraft nicht direkt in die x-, y-, oder z-Richtung, sondern entlang der Verbindungslinie der Atome. Um die Bewegungsgleichungen in die drei Raumrichtungen getrennt aufstellen zu können, muss diese Kraft jeweils auf die betrachtete Raumachse projiziert werden.

Zunächst berechnen wir, um wie viel eine Feder bei einer Auslenkung der Atome in eine Raumrichtung gestreckt bzw. gestaucht wird. Dazu betrachten wir zunächst nur die beiden Atome mit den Indizes (l,m,n) und ($l+1$,m,n) und lenken sie in die x-Richtung aus. Gemäß Abbildung 2 ist die Länge der Feder im Gleichgewichtszustand durch $|\vec{a}|$ gegeben. Bewegen wir beide Atome um den gleichen Betrag in die selbe Richtung, ändert das den Abstand der Atome nicht. Bewegen wir sie in entgegengesetzte Richtungen, wird die Feder zwischen ihnen entweder gestreckt oder gestaucht. Den neuen Verbindungsvektor zwischen den Atomen bezeichnen wir mit $\vec{\xi}$. Um $\vec{\xi}$ zu erhalten, müssen wir zum Verbindungsvektor die Auslenkung des Atoms ($l+1$,m,n) addieren und die des Atoms (l,m,n) subtrahieren, weil nur die Differenz der Auslenkungen eine Abstandsänderung bewirkt. Deshalb gilt

$$\vec{\xi} = u^{x}_{l+1,m,n}\vec{e}_x + \vec{a} - u^{x}_{l,m,n}\vec{e}_x$$

Der Betrag dieses Vektors ist

$$|\vec{\xi}| = \left(\left(u^{x}_{l+1,m,n} - u^{x}_{l,m,n} - \frac{a}{2}\right)^2 + \frac{2a^2}{4}\right)^{1/2}$$

Da dieser Ausdruck für die weitere Auswertung zu kompliziert ist, machen wir die Näherung, dass die Auslenkungen der Atome klein sind und entwickeln $|\vec{\xi}|$ bis zur ersten Ordnung in $u^{x}_{l+1,m,n}$ und $u^{x}_{l,m,n}$.

$$
\begin{aligned}
|\vec{\xi}|(u^x_{l,m,n},u^x_{l+1,m,n}) &= |\vec{\xi}|(0,0) + \left.\frac{\partial|\vec{\xi}|}{\partial u^x_{l,m,n}}\right|_{(0,0)} u^x_{l,m,n} \\
&\quad + \left.\frac{\partial|\vec{\xi}|}{\partial u^x_{l+1,m,n}}\right|_{(0,0)} u^x_{l+1,m,n} + \mathcal{O}({u^x_{l,m,n}}^2,{u^x_{l+1,m,n}}^2) \\
&= \frac{\sqrt{3}a}{2} - \left.\frac{\left(u^x_{l+1,m,n} - u^x_{l,m,n} - \frac{a}{2}\right)}{\left(\left(u^x_{l+1,m,n} - u^x_{l,m,n} - \frac{a}{2}\right)^2 + \frac{2a^2}{4}\right)^{1/2}}\right|_{(0,0)} u^x_{l,m,n} \\
&\quad + \left.\frac{\left(u^x_{l+1,m,n} - u^x_{l,m,n} - \frac{a}{2}\right)}{\left(\left(u^x_{l+1,m,n} - u^x_{l,m,n} - \frac{a}{2}\right)^2 + \frac{2a^2}{4}\right)^{1/2}}\right|_{(0,0)} u^x_{l+1,m,n} + \mathcal{O}({u^x_{l,m,n}}^2,{u^x_{l+1,m,n}}^2) \\
&= \frac{\sqrt{3}a}{2} - \frac{1}{\sqrt{3}}(u^x_{l+1,m,n} - u^x_{l,m,n}) + \mathcal{O}({u^x_{l,m,n}}^2,{u^x_{l+1,m,n}}^2)
\end{aligned}
$$

Die Längenänderung der Feder beträgt also in erster Ordnung $\frac{1}{\sqrt{3}}(u^x_{l+1,m,n} - u^x_{l,m,n})$. Für die Kraft entlang der Verbindungslinie der Atome folgt darus

$$|F| = \left|\frac{C}{\sqrt{3}}(u^x_{l+1,m,n} - u^x_{l,m,n})\right| \tag{2.1}$$

Diese Rechnung gilt aufgrund der Symmetrie des Problems analog auch für alle anderen Auslenkungsrichtungen aller nächst-Nachbaratome. Das Vorzeichen muss für jeden Nachbarn und jede Richtung eigens ermittelt werden (siehe unten).

Um die vollständige Bewegungsgleichung aufstellen zu können, müssen wir noch die Federkraft in die x-Richtung projizieren. Weil wir wissen, dass die Kraft entlang der Verbindungslinie der Atome gerichtet ist, können wir die Projektion einfach durch Bildung des Skalarproduktes des Verbindungsvektors der Atome mit dem Einheitsvektor in x-Richtung berechnen. Ist F der Betrag der Kraft entlang der Verbindungslinie, so gilt für die projizierte Kraft

$$|F_{proj}| = \left|F\frac{\vec{a}}{|\vec{a}|}\vec{e}_x\right| = \left|F\frac{1}{\sqrt{3}}\cdot\begin{pmatrix}-1\\1\\1\end{pmatrix}\cdot\begin{pmatrix}1\\0\\0\end{pmatrix}\right| = \left|\frac{F}{\sqrt{3}}\right|$$

Auch diese Berechnung gilt aufgrund der Symmetrie für alle Richtungen und nächsten Nachbarn.

Somit können wir jetzt die Kräfte der nächsten Nachbarn bestimmen.

$$|F_{proj}| = \left|\frac{F}{\sqrt{3}}\right| = \left|\frac{C}{3}(u^x_{l+1,m,n} - u^x_{l,m,n})\right| \tag{2.2}$$

Als nächstes betrachten wir die Kopplung zu den nächst-nächsten Nachbarn im bcc-Gitter. Die nächst-nächsten Nachbarn sind die raumzentrierten Atome der an den Flächen angrenzenden Einheitszellen. Es gibt demnach 6 von diesen Nachbarn, jeweils 2 in jeder Raumrichtung. Wir betrachten zunächst eine Auslenkung aller Atome in die x-Richtung. Die Federn zu den Nachbarn in x-Richtung mit den Indizes $(l,m+1,n+1)$ und $(l,m-1,n-1)$ werden dabei genau um die Differenz der Auslenkungen ($u^x_{l,m+1,n+1} - u^x_{l,m,n}$ bzw. $u^x_{l,m-1,n-1} - u^x_{l,m,n}$) gestreckt oder gestaucht. Die Beiträge der anderen Nachbarn (in y- und z-Richtung) sind bei kleinen Auslenkungen vernachlässigbar. Das lässt sich wieder mit einer Taylor-Entwicklung zeigen.

Wir betrachten die Kraft, die das Atom in der positiven y-Richtung bei einer Auslenkung in die x-Richtung ausübt. Die Indizes dieses Atoms sind $(l+1,m,n+1)$. Den neuen Verbindungsvektor bezeichnen wir wieder mit $\vec{\xi}$.

$$\vec{\xi} = \vec{a} + \vec{c} + (u^x_{l+1,m,n+1} - u^x_{l,m,n})\vec{e}_x$$

Der Betrag dieses Vektors ist

$$|\vec{\xi}| = \sqrt{a^2 + (u^x_{l+1,m,n+1} - u^x_{l,m,n})^2}$$

Diesen Ausdruck entwickeln wir jetzt wieder bis zur ersten Ordnung

$$\begin{aligned}
|\vec{\xi}|(u^x_{l,m,n}, u^x_{l+1,m,n+1}) &= |\vec{\xi}|(0,0) + \left.\frac{\partial|\vec{\xi}|}{\partial u^x_{l,m,n}}\right|_{(0,0)} u^x_{l,m,n} \\
&\quad + \left.\frac{\partial|\vec{\xi}|}{\partial u^x_{l+1,m,n+1}}\right|_{(0,0)} u^x_{l+1,m,n+1} + \mathcal{O}({u^x_{l,m,n}}^2, {u^x_{l+1,m,n+1}}^2) \\
&= a - \left.\frac{u^x_{l+1,m,n+1} - u^x_{l,m,n}}{\sqrt{a^2 + (u^x_{l+1,m,n+1} - u^x_{l,m,n})^2}}\right|_{(0,0)} u^x_{l,m,n} \\
&\quad + \left.\frac{u^x_{l+1,m,n+1} - u^x_{l,m,n}}{\sqrt{a^2 + (u^x_{l+1,m,n+1} - u^x_{l,m,n})^2}}\right|_{(0,0)} u^x_{l+1,m,n+1} + \mathcal{O}({u^x_{l,m,n}}^2, {u^x_{l+1,m,n+1}}^2) \\
&= a + \mathcal{O}({u^x_{l,m,n}}^2, {u^x_{l+1,m,n+1}}^2)
\end{aligned}$$

Wegen der Symmetrie gilt diese Rechnung auch für alle anderen Atome, die sich senkrecht zur Auslenkungsrichtung befinden.

Für die nächst-nächsten Nachbarn nehmen die Kräfte deshalb eine besonders einfache Form an. Für die x-Richtung gilt beispielsweise insgesamt

$$\begin{aligned} |F_{nnN}| &= \left|C_{\text{eff}}\left[(u^x_{l,m+1,n+1} - u^x_{l,m,n}) + (u^x_{l,m-1,n-1} - u^x_{l,m,n})\right]\right| \\ &= \left|C_{\text{eff}}(u^x_{l,m+1,n+1} - 2u^x_{l,m,n} + u^x_{l,m-1,n-1})\right| \end{aligned}$$

wobei berücksichtigt werden muss, dass die Federkonstante zu den nächst-nächsten Nachbarn eine Andere ist. Diese Federkonstante wird hier als effektive Federkonstante C_{eff} bezeichnet.

2.2 Bewegungsgleichungen

Wir gehen jetzt von der Newtonschen Bewegungsgleichung

$$m\frac{d^2u^\alpha_{l,m,n}}{dt^2} = F$$

aus, wobei der Betrag von F im vorigen Abschnitt hergeleitet wurde. Um die Vorzeichen zu erhalten, betrachten wir nochmals Abbildung 2. Als Beispiel nehmen wir das Atom vorne links oben und Auslenkungen in die y-Richtung, wollen aber die Kraft in die x-Richtung berechnen. Das entspricht dem Term der Gleichung

$$\left|m\frac{d^2u^x_{l,m,n}}{dt^2}\right| = \left|\frac{C}{3}(u^y_{l,m+1,n} - u^y_{l,m,n})\right|$$

Um das Vorzeichen der Kraft auf das mittlere Atom zu erhalten, lenken wir das Atom $(l,m+1,n)$ in die positive y-Richtung aus. Dadurch verkürzt sich der Abstand zum mittleren Atom und die Kraft ist nach rechts unten hinten gerichtet. Für die Kraft in x-Richtung bedeutet das, dass sie ein negatives Vorzeichen bekommt, weil „hinten“ die negative x-Richtung ist. Die Kraft in die y-Richtung wäre in diesem Fall positiv, die Kraft in die z-Richtung negativ.

Geht man nach diesem Schema alle nächsten Nachbaratome durch, lenkt sie jeweils in alle 3 Raumrichtungen aus und betrachtet, in welche Richtung die Kraft wirkt, erhält man alle Vorzeichen der Kräfte.

Für die nächst-nächsten Nachbarn ist dieser Zusammenhang wesentlich einfacher, weil die Nachbarn auf der Raumachse liegen. Die Kraft hat immer die Richtung der Auslenkung der Nachbarn und damit ein positives Vorzeichen. Summieren wir jetzt alle Kräfte, erhalten wir insgesamt Gleichung 2.4.

In diese Bewegungsgleichung macht man jetzt, wie in der Vorlesung *Molekül- und Festkörperphysik* beschrieben, den Normalmoden-Ansatz, bei dem alle Atome mit der gleichen Frequenz harmonische Schwingungen ausführen, da man aus einer Linearkombination dieser Lösungen jede spezielle Lösung konstruieren kann. Außerdem sind diese Funktionen Eigenfunktionen des Translationsoperators. Der Ansatz ist von der Form

$$u^{x,y,z}_{l,m,n} = u^{x,y,z}_{\vec{k}} \exp\left(i(l\vec{k}\vec{a} + m\vec{k}\vec{b} + n\vec{k}\vec{c} - \omega t)\right) \tag{2.3}$$

Setzt man diesen Ansatz in die Bewegungsgleichungen ein, kann man die Ableitungen nach t durchführen und die Differentialgleichungen werden zu algebraischen Gleichungen.

$$
\begin{aligned}
m\frac{d^2u^{x,y,z}_{l,m,n}}{dt^2} = \frac{C}{3}\Bigg[& \begin{matrix}-\\-\\+\end{matrix}(u^x_{l+1,m,n}-u^x_{l,m,n}) \begin{matrix}+\\+\\-\end{matrix}(u^y_{l+1,m,n}-u^y_{l,m,n}) \begin{matrix}+\\+\\-\end{matrix}(u^z_{l+1,m,n}-u^z_{l,m,n}) \\
& \begin{matrix}-\\+\\+\end{matrix}(u^x_{l,m,n-1}-u^x_{l,m,n}) \begin{matrix}-\\+\\+\end{matrix}(u^y_{l,m,n-1}-u^y_{l,m,n}) \begin{matrix}+\\-\\-\end{matrix}(u^z_{l,m,n-1}-u^z_{l,m,n}) \\
& \begin{matrix}+\\-\\+\end{matrix}(u^x_{l,m+1,n}-u^x_{l,m,n}) \begin{matrix}-\\+\\-\end{matrix}(u^y_{l,m+1,n}-u^y_{l,m,n}) \begin{matrix}+\\-\\+\end{matrix}(u^z_{l,m+1,n}-u^z_{l,m,n}) \\
& \begin{matrix}+\\-\\+\end{matrix}(u^x_{l,m-1,n}-u^x_{l,m,n}) \begin{matrix}-\\+\\-\end{matrix}(u^y_{l,m-1,n}-u^y_{l,m,n}) \begin{matrix}+\\-\\+\end{matrix}(u^z_{l,m-1,n}-u^z_{l,m,n}) \\
& \begin{matrix}-\\-\\+\end{matrix}(u^x_{l-1,m,n}-u^x_{l,m,n}) \begin{matrix}+\\+\\-\end{matrix}(u^y_{l-1,m,n}-u^y_{l,m,n}) \begin{matrix}+\\+\\-\end{matrix}(u^z_{l-1,m,n}-u^z_{l,m,n}) \\
& \begin{matrix}-\\+\\+\end{matrix}(u^x_{l,m,n+1}-u^x_{l,m,n}) \begin{matrix}-\\+\\+\end{matrix}(u^y_{l,m,n+1}-u^y_{l,m,n}) \begin{matrix}+\\-\\-\end{matrix}(u^z_{l,m,n+1}-u^z_{l,m,n}) \\
& \begin{matrix}+\\+\\+\end{matrix}(u^x_{l-1,m-1,n-1}-u^x_{l,m,n}) \begin{matrix}+\\+\\+\end{matrix}(u^y_{l-1,m-1,n-1}-u^y_{l,m,n}) \\
& \begin{matrix}+\\+\\+\end{matrix}(u^z_{l-1,m-1,n-1}-u^z_{l,m,n}) \begin{matrix}+\\+\\+\end{matrix}(u^x_{l+1,m+1,n+1}-u^x_{l,m,n}) \\
& \begin{matrix}+\\+\\+\end{matrix}(u^y_{l+1,m+1,n+1}-u^y_{l,m,n}) \begin{matrix}+\\+\\+\end{matrix}(u^z_{l+1,m+1,n+1}-u^z_{l,m,n})\Bigg] \\
& + \underbrace{\begin{cases} C_{\text{eff}}(u^z_{l+1,m+1,n}-2u^z_{l,m,n}+u^z_{l-1,m-1,n}) \\ C_{\text{eff}}(u^y_{l+1,m,n+1}-2u^y_{l,m,n}+u^y_{l-1,m,n-1}) \\ C_{\text{eff}}(u^x_{l,m+1,n+1}-2u^x_{l,m,n}+u^x_{l,m-1,n-1}) \end{cases}}_{\text{nächst-nächste Nachbarn}}
\end{aligned}
\tag{2.4}
$$

Außerdem kann man wegen

$$
\begin{aligned}
\exp\Big(i((l+1)\vec{k}\vec{a}+(m+1)\vec{k}\vec{b}+(n+1)\vec{k}\vec{c}-\omega t)\Big) = \\
= \exp\Big(i(l\vec{k}\vec{a}+m\vec{k}\vec{b}+n\vec{k}\vec{c}-\omega t)\Big)\cdot\exp\Big(i(\vec{k}\vec{a}+\vec{k}\vec{b}+\vec{k}\vec{c})\Big)
\end{aligned}
$$

jeweils die Funktion $u^\alpha_{l,m,n}$ aus der Klammer herausheben. Weiters stellt sich heraus, dass es zu jedem $e^{i\cdots}$ ein $e^{-i\cdots}$ gibt, die man wegen

$$2\cos x = e^{ix}+e^{-ix}$$

zu einem cos-Term zusammenfassen kann. Mit all diesen Vereinfachungen und nach dem Zusammenfassen der Terme nach $u^\alpha_{l,m,n}$ erhalten wir schließlich die folgenden Gleichungen.

$$
\begin{aligned}
&u^x_{l,m,n}\left(\frac{3m\omega^2}{C}-8+2\cos\left(\vec{k}\vec{a}\right)+2\cos\left(\vec{k}\vec{c}\right)+2\cos\left(\vec{k}\vec{b}\right)+2\cos\left(\vec{k}\left(\vec{a}+\vec{b}+\vec{c}\right)\right)\right)+\\
&u^y_{l,m,n}\left(-2\cos\left(\vec{k}\vec{a}\right)+2\cos\left(\vec{k}\vec{c}\right)-2\cos\left(\vec{k}\vec{b}\right)+2\cos\left(\vec{k}\left(\vec{a}+\vec{b}+\vec{c}\right)\right)\right)+\\
&u^z_{l,m,n}\left(-2\cos\left(\vec{k}\vec{a}\right)-2\cos\left(\vec{k}\vec{c}\right)+2\cos\left(\vec{k}\vec{b}\right)+2\cos\left(\vec{k}\left(\vec{a}+\vec{b}+\vec{c}\right)\right)\right)+\\
&u^x_{l,m,n}\left(6\frac{C_{\text{eff}}}{C}\left(\cos\left(\vec{k}\vec{b}+\vec{k}\vec{c}\right)-1\right)\right)=0
\end{aligned}
$$

$$
\begin{aligned}
&u^x_{l,m,n}\left(-2\cos\left(\vec{k}\vec{a}\right)+2\cos\left(\vec{k}\vec{c}\right)-2\cos\left(\vec{k}\vec{b}\right)+2\cos\left(\vec{k}\left(\vec{a}+\vec{b}+\vec{c}\right)\right)\right)+\\
&u^y_{l,m,n}\left(\frac{3m\omega^2}{C}-8+2\cos\left(\vec{k}\vec{a}\right)+2\cos\left(\vec{k}\vec{c}\right)+2\cos\left(\vec{k}\vec{b}\right)+2\cos\left(\vec{k}\left(\vec{a}+\vec{b}+\vec{c}\right)\right)\right)+\\
&u^z_{l,m,n}\left(2\cos\left(\vec{k}\vec{a}\right)-2\cos\left(\vec{k}\vec{c}\right)-2\cos\left(\vec{k}\vec{b}\right)+2\cos\left(\vec{k}\left(\vec{a}+\vec{b}+\vec{c}\right)\right)\right)+\\
&u^y_{l,m,n}\left(6\frac{C_{\text{eff}}}{C}\left(\cos\left(\vec{k}\vec{a}+\vec{k}\vec{c}\right)-1\right)\right)=0
\end{aligned}
$$

$$
\begin{aligned}
&u^x_{l,m,n}\left(-2\cos\left(\vec{k}\vec{a}\right)-2\cos\left(\vec{k}\vec{c}\right)+2\cos\left(\vec{k}\vec{b}\right)+2\cos\left(\vec{k}\left(\vec{a}+\vec{b}+\vec{c}\right)\right)\right)+\\
&u^y_{l,m,n}\left(2\cos\left(\vec{k}\vec{a}\right)-2\cos\left(\vec{k}\vec{c}\right)-2\cos\left(\vec{k}\vec{b}\right)+2\cos\left(\vec{k}\left(\vec{a}+\vec{b}+\vec{c}\right)\right)\right)+\\
&u^z_{l,m,n}\left(\frac{3m\omega^2}{C}-8+2\cos\left(\vec{k}\vec{a}\right)+2\cos\left(\vec{k}\vec{c}\right)+2\cos\left(\vec{k}\vec{b}\right)+2\cos\left(\vec{k}\left(\vec{a}+\vec{b}+\vec{c}\right)\right)\right)+\\
&u^z_{l,m,n}\left(6\frac{C_{\text{eff}}}{C}\left(\cos\left(\vec{k}\vec{a}+\vec{k}\vec{b}\right)-1\right)\right)=0
\end{aligned}
$$

Dieses homogene Gleichungssystem in $u^{\alpha}_{l,m,n}$ lässt sich als Matrix schreiben. Damit eine Lösung außer der Nullösung existiert, muss die Determinante der Koeffizientenmatrix (2.6) verschwinden.

Das entspricht genau einem Eigenwertproblem

$$\det\left(\hat{A}-\lambda\mathbb{1}\right)=0$$

mit

$$\lambda=\frac{3m\omega^2}{2C}$$

Hier ist es nun praktisch, reduzierte Einheiten zu verwenden. Für die Frequenzen erhalten wir

$$\omega\sqrt{\frac{m}{C}}=\sqrt{\frac{2\lambda}{3}} \tag{2.5}$$

mit λ den Eigenwerten der unten angeführten Matrix. Mit diesen Gleichungen können nun zu jedem $\vec{k}$ die 3 Frequenzen und somit die Dispersionsrelation und die Zustandsdichte berechnet werden.

$$
\begin{pmatrix}
2\cos\left(\vec{k}\vec{a}\right)+2\cos\left(\vec{k}\vec{c}\right)+2\cos\left(\vec{k}\vec{b}\right) & -2\cos\left(\vec{k}\vec{a}\right)+2\cos\left(\vec{k}\vec{c}\right) & -2\cos\left(\vec{k}\vec{a}\right)-2\cos\left(\vec{k}\vec{c}\right) \\
+2\cos\left(\vec{k}\left(\vec{a}+\vec{b}+\vec{c}\right)\right)+\frac{3m\omega^2}{C}-8 & -2\cos\left(\vec{k}\vec{b}\right)+2\cos\left(\vec{k}\left(\vec{a}+\vec{b}+\vec{c}\right)\right) & +2\cos\left(\vec{k}\vec{b}\right)+2\cos\left(\vec{k}\left(\vec{a}+\vec{b}+\vec{c}\right)\right) \\
+6\frac{C_{\text{eff}}}{C}\left(\cos\left(\vec{k}\vec{b}+\vec{k}\vec{c}\right)-1\right) & & \\
-2\cos\left(\vec{k}\vec{a}\right)+2\cos\left(\vec{k}\vec{c}\right) & 2\cos\left(\vec{k}\vec{a}\right)+2\cos\left(\vec{k}\vec{c}\right)+2\cos\left(\vec{k}\vec{b}\right) & 2\cos\left(\vec{k}\vec{a}\right)-2\cos\left(\vec{k}\vec{c}\right) \\
-2\cos\left(\vec{k}\vec{b}\right)+2\cos\left(\vec{k}\left(\vec{a}+\vec{b}+\vec{c}\right)\right) & +2\cos\left(\vec{k}\left(\vec{a}+\vec{b}+\vec{c}\right)\right)+\frac{3m\omega^2}{C}-8 & -2\cos\left(\vec{k}\vec{b}\right)+2\cos\left(\vec{k}\left(\vec{a}+\vec{b}+\vec{c}\right)\right) \\
 & +6\frac{C_{\text{eff}}}{C}\left(\cos\left(\vec{k}\vec{a}+\vec{k}\vec{c}\right)-1\right) & \\
-2\cos\left(\vec{k}\vec{a}\right)-2\cos\left(\vec{k}\vec{c}\right) & 2\cos\left(\vec{k}\vec{a}\right)-2\cos\left(\vec{k}\vec{c}\right) & 2\cos\left(\vec{k}\vec{a}\right)+2\cos\left(\vec{k}\vec{c}\right)+2\cos\left(\vec{k}\vec{b}\right) \\
+2\cos\left(\vec{k}\vec{b}\right)+2\cos\left(\vec{k}\left(\vec{a}+\vec{b}+\vec{c}\right)\right) & -2\cos\left(\vec{k}\vec{b}\right)+2\cos\left(\vec{k}\left(\vec{a}+\vec{b}+\vec{c}\right)\right) & +2\cos\left(\vec{k}\left(\vec{a}+\vec{b}+\vec{c}\right)\right)+\frac{3m\omega^2}{C}-8 \\
 & & +6\frac{C_{\text{eff}}}{C}\left(\cos\left(\vec{k}\vec{a}+\vec{k}\vec{b}\right)-1\right)
\end{pmatrix}
\begin{pmatrix} u^x_{l,m,n} \\ u^y_{l,m,n} \\ u^z_{l,m,n} \end{pmatrix} = \vec{0}
$$

$$
\begin{vmatrix}
-\cos\left(\frac{a}{2}\left(-k_x+k_y+k_z\right)\right)-\cos\left(\frac{a}{2}\left(k_x+k_y-k_z\right)\right) & \cos\left(\frac{a}{2}\left(-k_x+k_y+k_z\right)\right)-\cos\left(\frac{a}{2}\left(k_x+k_y-k_z\right)\right) & \cos\left(\frac{a}{2}\left(-k_x+k_y+k_z\right)\right)+\cos\left(\frac{a}{2}\left(k_x+k_y-k_z\right)\right) \\
-\cos\left(\frac{a}{2}\left(k_x-k_y+k_z\right)\right)-\cos\left(\frac{a}{2}\left(k_x+k_y+k_z\right)\right) & +\cos\left(\frac{a}{2}\left(k_x-k_y+k_z\right)\right)-\cos\left(\frac{a}{2}\left(k_x+k_y+k_z\right)\right) & -\cos\left(\frac{a}{2}\left(k_x-k_y+k_z\right)\right)-\cos\left(\frac{a}{2}\left(k_x+k_y+k_z\right)\right) \\
+4-\frac{3m\omega^2}{2C}-3\frac{C_{\text{eff}}}{C}\left(\cos\left(a\cdot k_x\right)-1\right) & & \\
\cos\left(\frac{a}{2}\left(-k_x+k_y+k_z\right)\right)-\cos\left(\frac{a}{2}\left(k_x+k_y-k_z\right)\right) & -\cos\left(\frac{a}{2}\left(-k_x+k_y+k_z\right)\right)-\cos\left(\frac{a}{2}\left(k_x+k_y-k_z\right)\right) & -\cos\left(\frac{a}{2}\left(-k_x+k_y+k_z\right)\right)+\cos\left(\frac{a}{2}\left(k_x+k_y-k_z\right)\right) \\
+\cos\left(\frac{a}{2}\left(k_x-k_y+k_z\right)\right)-\cos\left(\frac{a}{2}\left(k_x+k_y+k_z\right)\right) & -\cos\left(\frac{a}{2}\left(k_x-k_y+k_z\right)\right)-\cos\left(\frac{a}{2}\left(k_x+k_y+k_z\right)\right) & +\cos\left(\frac{a}{2}\left(k_x-k_y+k_z\right)\right)-\cos\left(\frac{a}{2}\left(k_x+k_y+k_z\right)\right) \\
 & +4-\frac{3m\omega^2}{2C}-3\frac{C_{\text{eff}}}{C}\left(\cos\left(a\cdot k_y\right)-1\right) & \\
\cos\left(\frac{a}{2}\left(-k_x+k_y+k_z\right)\right)+\cos\left(\frac{a}{2}\left(k_x+k_y-k_z\right)\right) & -\cos\left(\frac{a}{2}\left(-k_x+k_y+k_z\right)\right)+\cos\left(\frac{a}{2}\left(k_x+k_y-k_z\right)\right) & -\cos\left(\frac{a}{2}\left(-k_x+k_y+k_z\right)\right)-\cos\left(\frac{a}{2}\left(k_x+k_y-k_z\right)\right) \\
-\cos\left(\frac{a}{2}\left(k_x-k_y+k_z\right)\right)-\cos\left(\frac{a}{2}\left(k_x+k_y+k_z\right)\right) & +\cos\left(\frac{a}{2}\left(k_x-k_y+k_z\right)\right)-\cos\left(\frac{a}{2}\left(k_x+k_y+k_z\right)\right) & -\cos\left(\frac{a}{2}\left(k_x-k_y+k_z\right)\right)-\cos\left(\frac{a}{2}\left(k_x+k_y+k_z\right)\right) \\
 & & +4-\frac{3m\omega^2}{2C}-3\frac{C_{\text{eff}}}{C}\left(\cos\left(a\cdot k_z\right)-1\right)
\end{vmatrix} = 0 \tag{2.6}
$$

3 Programmierung

3.1 Allgemeines

Bei der Programmierung dieses Problems ist es wichtig, auf die Geschwindigkeit zu achten, weil für ein brauchbares Ergebnis sehr viele k-Werte berechnet werden müssen. Deshalb gehen wir hier genauer auf unser Programm ein.

Zur Berechnung der Density of States haben wir die Funktion `get_dos`, die einige Argumente für Optionen hat, auf die später noch eingegangen wird. Hier werden die k-Werte erzeugt und, wenn sie in der 1. Brillouin-Zone liegen, die zugehörigen Frequenzen berechnet, sowie das Histogramm erstellt.

Die Überprüfung, ob ein bestimmter k-Vektor in der 1. Brillouin-Zone liegt, übernimmt die Funktion `inbril`, die dazu k-Werte und die Eckpunkte der Brillouin-Zone benötigt. Für die Berechnung der Frequenzen zu einem k-Vektor gibt es die Funktion `getFrequencies`, die das Verhältnis der effektiven Federkonstante der nächst-nächsten Nachbarn zur Federkonstante der nächsten Nachbarn braucht. Setzt man dieses auf 0, erhält man das Modell ohne Kopplung der nächst-nächsten Nachbarn. Die Berechnung der Eigenwerte der 3×3-Matrizen erfolgt dann mit Hilfe der Cardanischen Formeln in der Funktion `cardan`.

Zur Berechnung der Dispersionsrelation gibt es die Funktion `disp_rel`, die die Symmetriepunkte und die Auflösung der Linien braucht, und die Funktionswerte entlang der Verbindungslinien sowie die Zeilenindizes der Symmetriepunkte im Ergebnisvektor zurückgibt.

3.2 Eigenwerte von 3×3-Matrizen

Die Berechnung von Eigenwerten mit MATLAB-Funktionen ist relativ zeitintensiv. Weil wir nur die Eigenwerte ohne Eigenvektoren brauchen, stellen wir ein Polynom 3. Grades auf, das dann effizient gelöst werden kann. Dazu gehen wir vom allgemeinen Eigenwertproblem einer 3×3-Matrix aus.

$$\begin{vmatrix} a_{11}-\lambda & a_{12} & a_{13} \\ a_{21} & a_{22}-\lambda & a_{23} \\ a_{31} & a_{32} & a_{33}-\lambda \end{vmatrix} = 0 \tag{3.1}$$

Mit der Regel von Sarrus kann man für die Determinante direkt ein Polynom 3. Grades in λ gewinnen, dessen Nullstellen die Eigenwerte sind. Durch Ausmultiplizieren und Sortieren der Terme nach Potenzen von λ ergibt sich folgendes Polynom.

$$\begin{aligned} 0 = &-\lambda^3 + \lambda^2(a_{11}+a_{22}+a_{33}) - \lambda[a_{11}(a_{22}+a_{33}) - a_{12}a_{21} - a_{13}a_{31} \\ &+ a_{22}a_{33} - a_{23}a_{32}] + a_{11}(a_{22}a_{33} - a_{23}a_{32}) - a_{12}(a_{21}a_{33} - a_{23}a_{31}) \\ &+ a_{13}(a_{21}a_{32} - a_{22}a_{31}) \end{aligned} \tag{3.2}$$

3.3 cardan.m

Weil **`roots(c)`** immer jeweils nur ein Polynom lösen kann, verwenden wir die Cardanischen Formeln zur Lösung vieler Gleichungen gleichzeitig. Die genaue Herleitung kann z.B. auf de.wikipedia.org/wiki/Cardanische_Formeln nachgeschlagen werden, wir verwenden hier nur die endgültigen Formeln.

Ausgangspunkt der Rechnung ist ein Polynom 3. Ordnung in Normalform

$$x^3 + bx^2 + cx + d = 0 \tag{3.3}$$

Zur Lösung schreiben wir eine Funktion `cardan`, die b, c und d als gleich lange Spaltenvektoren mit Länge n übernimmt und die Lösungen als $n \times 3$-Matrix mit den drei Nullstellen jeder Gleichung pro Zeile zurückgibt.

```
function lsg = cardan(b,c,d)
```

Aus der Normalform wird mit den Substitutionen

$$x = z - \frac{b}{3}$$
$$p = c - \frac{b^2}{3}$$
$$q = \frac{2b^3}{27} - \frac{bc}{3} + d$$

die Gleichung zu

$$z^3 + pz + q = 0 \tag{3.4}$$

Diese Werte werden in der Funktion als erstes berechnet.

```
p=c-b.^2./3;
q=2.*b.^3./27-b.*c./3+d;
```

Welche Formel verwendet werden muss, ergibt sich aus dem Vorzeichen der Diskriminante.

$$D = \left(\frac{q}{2}\right)^2 + \left(\frac{p}{3}\right)^3 \tag{3.5}$$

Deshalb berechnen wir diese als nächstes.

```
dis=(q./2).^2+(p./3).^3;
```

Weil je nach Vorzeichen eine andere Formel verwendet werden muss, erstellen wir logische Spaltenvektoren, die jeweils in der Zeile `true` sind, in der eine bestimmte Bedingung zutrifft.

```
dis1=(dis>0);
dis2=(dis<0);
dis3=(dis==0);
```

Nach dem Initialisieren der Lösungsmatrix

```
lsg=nan(numel(dis),3);
```

behandeln wir den ersten Fall $D > 0$. In diesem Fall gibt es eine reelle und zwei komplex konjugierte Lösungen.

$$z_1 = u + v \tag{3.6}$$

$$z_{2,3} = -\frac{u+v}{2} \pm \frac{u-v}{2} i\sqrt{3} \tag{3.7}$$

mit

$$u = \sqrt[3]{-\frac{q}{2} + \sqrt{D}}$$

$$v = \sqrt[3]{-\frac{q}{2} - \sqrt{D}}$$

Um die Wurzel nicht zwei mal berechnen zu müssen, speichern wir sie in einer Variable.

```
sqdis=sqrt(dis(dis1));
u=nthroot(-q(dis1)/2+sqdis,3);
v=nthroot(-q(dis1)/2-sqdis,3);
lsg(dis1,1)=u+v;
lsg(dis1,2)=(-u-v)./2+(u-v)./2.*1i.*sqrt(3);
lsg(dis1,3)=conj(lsg(dis1,2));
```

Der nächste Fall ist $D < 0$. Hier sind alle 3 Lösungen reell und gegeben durch

$$z_1 = \quad \sqrt{-\frac{4p}{3}} \cdot \cos\left(\frac{1}{3}\arccos\left(-\frac{q}{2}\cdot\sqrt{-\frac{27}{p^3}}\right)\right) \tag{3.8}$$

$$z_2 = -\sqrt{-\frac{4p}{3}} \cdot \cos\left(\frac{1}{3}\arccos\left(-\frac{q}{2}\cdot\sqrt{-\frac{27}{p^3}}\right) - \frac{\pi}{3}\right) \tag{3.9}$$

$$z_3 = -\sqrt{-\frac{4p}{3}} \cdot \cos\left(\frac{1}{3}\arccos\left(-\frac{q}{2}\cdot\sqrt{-\frac{27}{p^3}}\right) + \frac{\pi}{3}\right) \tag{3.10}$$

Zur besseren Übersicht und Einsparung von Rechenzeit wird hier der arccos, der bei allen drei Lösungen das gleiche Argument hat, zuerst berechnet und dann in die Gleichungen eingesetzt.

```
arco=acos(-q(dis2)./2.*sqrt(-27./(p(dis2).^3)));
lsg(dis2,1)= sqrt(-4/3*p(dis2)).*cos(arco./3);
lsg(dis2,2)=-sqrt(-4/3*p(dis2)).*cos(arco./3-pi/3);
lsg(dis2,3)=-sqrt(-4/3*p(dis2)).*cos(arco./3+pi/3);
```

Der letzte Fall ist $D = 0$. Hier gibt es eine einfache und eine doppelte reelle Lösung.

$$z_1 = \sqrt[3]{-4q} \tag{3.11}$$

$$z_{2,3} = -\frac{z_1}{2} \tag{3.12}$$

```
lsg(dis3,1)=nthroot(-4*q(dis3),3);
lsg(dis3,2)=-lsg(dis3,1)./2;
lsg(dis3,3)=lsg(dis3,2);
```

Im Feld `lsg` stehen jetzt die Lösungen für z. Um die Ergebnisse für x zu erhalten, muss wieder nach Gleichung 3.4 zurücktransformiert werden, also von jeder Wert $\frac{b}{3}$ abgezogen werden. Weil hier b für jede Zeile einen anderen Wert hat, verwenden wir **`bsxfun`**.

```
lsg=bsxfun(@minus,lsg,b./3);
```

3.4 inbril.m

Zur Berechnung der Frequenzen dürfen nur k-Vektoren verwendet werden, die in der ersten Brillouin-Zone liegen. Deshalb prüfen wir diese Bedingung in der Funktion `inbril`. Übergeben werden die Eckpunkte der Brillouin-Zone in `brilCorners` mit den x-, y- und z-Koordinaten in Spalten und einem Punkt pro Zeile, sowie die zu prüfenden Punkte in `points` im gleichen Format. Zurückgegeben werden nur jene Punkte, die innerhalb der Brillouin-Zone liegen, ebenfalls im gleichen Format.

```
function inpoints = inbril(brilCorners,points)
```

Als erstes brauchen wir Dreiecke, die die Brillouin-Zone aufspannen. Weil diese per Konstruktion konvex ist, können wir dazu **`convhulln(X)`** verwenden. Als Rückgabe

bekommen wir die Eckpunkte von Dreiecken, die die Hülle aufspannen. Diese Eckpunkte erhalten wir aber nur als Indizes der übergebenen Punkte. Eine Rückgabe von `[1, 2, 5]` bedeutet also ein Dreieck, das von den Punkten in den Zeilen 1, 2 und 5 gebildet wird.

```
tri=convhulln(brilCorners);
```

Als nächstes brauchen wir Vektoren, die normal auf diesen Flächen stehen. Dazu erzeugen wir zunächst aus den Eckpunkten der Dreiecke (in `tri`) pro Fläche zwei linear unabhängige Vektoren, die in der Fläche liegen. Am einfachsten ist es, einfach die Strecken entlang zweier Kanten zu nehmen. `brilCorners(tri(:,1),:)` liefert dabei die drei Koordinaten aller ersten Punkte der Dreiecke, weil in `tri(:,1)` alle Zeilenindizes dieser Punkte stehen. Ziehen wir von diesen Koordinaten jeweils die Koordinaten der zweiten Punkte der Dreiecke ab, erhalten wir einen Vektor, der entlang einer Dreieckskante vom Eckpunkt 1 zum Eckpunkt 2 zeigt. Analoges gilt für die Strecke von Punkt 1 zu Punkt 3.

```
vecA=brilCorners(tri(:,1),:)-brilCorners(tri(:,3),:);
vecB=brilCorners(tri(:,1),:)-brilCorners(tri(:,2),:);
```

`vecA` und `vecB` haben nun in jeder Zeile Vektoren, die in der Ebene des jeweiligen Dreiecks liegen. Das Kreuzprodukt dieser beiden Vektoren steht dann normal auf den Dreiecken.

```
normVec=cross(vecA,vecB);
```

Weiters brauchen wir pro Fläche einen Punkt, der darauf liegt, um einen Vektor von der Fläche zu den zu prüfenden Punkten konstruieren zu können. Dazu verwenden wir einfach den zweiten Eckpunkt jedes Dreiecks.

```
PointPerFace=brilCorners(tri(:,2),:);
```

Diese Vektoren zeigen vom Ursprung auf die einzelnen Flächen. Weil der Ursprung immer in der ersten Brillouin-Zone liegt, sind sie alle nach außen gerichtet. Um nun zu prüfen, ob die berechneten Normalvektoren nach innen zeigen, brauchen wir nur die Skalarprodukte dieser Vektoren mit den Flächennormalen zu berechnen. Sind sie positiv, zeigt die Flächennormale ebenfalls nach außen, sind sie negativ, zeigt sie nach innen. `sum(A.*B,2)` berechnet dabei das Skalarprodukt $\vec{A} \cdot \vec{B}$, wenn, wie in unserem Fall, `A` und `B` Zeilenvektoren sind.

Die Information über die Verktoren, die in die falsche Richtung zeigen, speichern wir

und drehen diese einfach um.

```
wrongDir=sum(PointPerFace.*normVec,2)>0;
normVec(wrongDir,:)=-normVec(wrongDir,:);
```

Jetzt haben wir für jede Fläche einen Normalenvektor, der in die Brillouin-Zone zeigt. Mit diesen können wir für jeden beliebigen Punkt überprüfen, ob er innen oder außen liegt. Dazu brauchen wir für jeden Punkt einen Vektor, der von einem Punkt auf der Oberfläche jedes Dreiecks zu diesem zeigt.

$$\vec{a} = \vec{P} - \vec{F}$$

wobei $\vec{F}$ der Punkt auf der Fläche und $\vec{P}$ der zu prüfende Punkt sind. Sind nun alle Skalarprodukte der nach innen zeigenden Flächennormalen $\vec{N}$ mit diesen Verbindungsvektoren $\vec{a}$ nicht negativ, so zeigen die Vektoren von allen Flächen zum Punkt ebenfalls nach innen, und er liegt in oder auf der Brillouin-Zone. Um Rechenzeit zu sparen, schreiben wir

$$\vec{a} \cdot \vec{N} = (\vec{P} - \vec{F}) \cdot \vec{N} = \vec{P} \cdot \vec{N} - \vec{F} \cdot \vec{N} \tag{3.13}$$

und berechnen den zweiten Term, der für alle Punkte gleich ist, sofort.

```
fn=sum(PointPerFace.*normVec,2);
```

Zum Speichern der Information, ob ein Punkt innerhalb oder außerhalb der Brillouin-Zone liegt, erstellen wir einen logischen Spaltenvektor in der Größe der Anzahl der zu prüfenden Punkte.

```
inlog=false(size(points,1),1);
```

Den ersten Term in Gleichung 3.13 können wir mittels Matrixmultiplikation auswerten, indem wir `normVec*points'` berechnen.

$$\begin{pmatrix} N_1^x & N_1^y & N_1^z \\ N_2^x & N_2^y & N_2^z \\ N_3^x & N_3^y & N_3^z \\ \vdots & \vdots & \vdots \\ N_n^x & N_n^y & N_n^z \end{pmatrix} \cdot \begin{pmatrix} P_1^x & P_2^x & \cdots & P_m^x \\ P_1^y & P_2^y & \cdots & P_m^y \\ P_1^z & P_2^z & \cdots & P_m^z \end{pmatrix}$$

$$= \begin{pmatrix} N_1^x P_1^x + N_1^y P_1^y + N_1^z P_1^z & N_1^x P_2^x + N_1^y P_2^y + N_1^z P_2^z & \cdots & N_1^x P_m^x + N_1^y P_m^y + N_1^z P_m^z \\ N_2^x P_1^x + N_2^y P_1^y + N_2^z P_1^z & N_2^x P_2^x + N_2^y P_2^y + N_2^z P_2^z & \cdots & N_2^x P_m^x + N_2^y P_m^y + N_2^z P_m^z \\ N_3^x P_1^x + N_3^y P_1^y + N_3^z P_1^z & N_3^x P_2^x + N_3^y P_2^y + N_3^z P_2^z & \cdots & N_3^x P_m^x + N_3^y P_m^y + N_3^z P_m^z \\ \vdots & \vdots & \ddots & \vdots \\ N_n^x P_1^x + N_n^y P_1^y + N_n^z P_1^z & N_n^y P_2^x + N_n^y P_2^y + N_n^z P_2^z & \cdots & N_n^y P_m^x + N_n^y P_m^y + N_n^z P_m^z \end{pmatrix}$$

$$= \begin{pmatrix} \vec{N_1}\cdot\vec{P_1} & \vec{N_1}\cdot\vec{P_2} & \cdots & \vec{N_1}\cdot\vec{P_m} \\ \vec{N_2}\cdot\vec{P_1} & \vec{N_2}\cdot\vec{P_2} & \cdots & \vec{N_2}\cdot\vec{P_m} \\ \vec{N_3}\cdot\vec{P_1} & \vec{N_3}\cdot\vec{P_2} & \cdots & \vec{N_3}\cdot\vec{P_m} \\ \vdots & \vdots & \ddots & \vdots \\ \vec{N_n}\cdot\vec{P_1} & \vec{N_n}\cdot\vec{P_2} & \cdots & \vec{N_n}\cdot\vec{P_m} \end{pmatrix}$$

Hier sind N_i^α die α-Komponenten des i-ten Normalenvektors und P_j^β die β-Komponenten des j-ten zu prüfenden Punktes. Es stehen dann also alle Skalarprodukte für einen zu prüfenden Punkt in einer Spalte.

Es zeigt sich, dass die Matrixmultiplikation in MATLAB mit maximal 10^6 Elementen im Ergebnis wesentlich schneller ist, als riesige Matrizen auf einmal zu multiplizieren. Außerdem ist das blockweise Abarbeiten speicherplatzsparender. Weil das Produkt `normVec*points'` $n \cdot m$ (n... Zahl der Dreiecke, m... Zahl der zu prüfenden Punkte) Elemente enthält, berechnen wir zunächst, wie viele Punkte wir mit 10^6 Elementen auf einmal prüfen können.

```
blockel=floor(1E6/size(normVec,1));
```

Dividieren wir die Anzahl der zu prüfenden Punkte durch diese Zahl, so erhalten wir die Anzahl der Blöcke. Wenn wir zusätzlich abrunden, ergibt das die Zahl der vollständigen Blöcke, wobei evtl. ein unvollständiger Block mit weniger als 10^6 Elementen übrig bleibt.

```
blo=floor(size(points,1)/blockel);
```

Die vollständigen Blöcke arbeiten wir in einer Schleife ab, wobei wir von jeder Spalte nach Gleichung 3.13 noch das zuvor bereits berechnete Produkt $\vec{F}\cdot\vec{N}$ abziehen müssen. Dafür verwenden wir **`bsxfun(@minus,a,b)`**, das von jeder Spalte in `a` den Spaltenvektor `b` abzieht. **`all(a,1)`** überprüft dann, ob in einer Spalte alle Werte `true` sind, also in unserem Fall alle Skalarprodukte nach Gleichung 3.13 nicht negativ sind. Dann ist der Punkt in der ersten Brillouin-Zone.

```
for k=1:blo
  inlog((k-1)*blockel+1:k*blockel)=all(bsxfun(@minus,...
      normVec*points((k-1)*blockel+1:k*blockel,:)',fn)>=0,1);
end
```

Zum Schluss müssen wir noch die übrig gebliebenen Elemente berechnen. Der Aufruf ist fast der selbe, lediglich der Index läuft hier nur bis **`end`**.

```
inlog(blo*blockel+1:end)=all(bsxfun(@minus,...
    normVec*points(blo*blockel+1:end,:)',fn)>=0,1);
```

Jetzt enthält `inlog` in allen Zeilen `true`, deren entsprechende Punkte in der ersten Brillouin-Zone liegen. Damit können wir eine Liste der innen liegenden Punkte zurückgeben.

```
inpoints=points(inlog,:);
```

3.5 getFrequencies.m

Um zu gegebenen k-Vektoren die drei jeweils zugehörigen Frequenzen zu berechnen, verwenden wir die Funktion `getFrequencies` mit den folgenden Parametern.

- `k` enthält die die zu bearbeitenden k-Vektoren in Zeilen, also einer $n \times 3$-Matrix. Die k-Vektoren gehen dabei nicht bis $\frac{2\pi}{a}$, sondern immer bis 2π.
- `ceff_c` ist die effektive Federkonstante zu den nächst-nächsten Nachbarn im Verhältnis zur Federkonstante der nächsten Nachbarn.

Zurückgegeben wird ebenfalls eine $n \times 3$-Matrix, die pro Zeile die zu einem bestimmten k-Vektor gehörenden Frequenzen enthält.

```
function omega = getFrequencies(k,ceff_c)
```

Als erstes reservieren wir Speicherplatz für alle Matrizen, deren Eigenwerte wir berechnen wollen. Das ist pro k-Vektor eine 3×3-Matrix.

```
matr=nan(3,3,size(k,1));
```

Als nächstes tragen wir in alle Matrizen die Werte laut Gleichung 2.6 ein. Hier bräuchten wir eigentlich den Parameter a. Weil aber die k-Vektoren bis $\frac{2\pi}{a}$ gehen und hier wieder mit a multipliziert werden, lassen wir uns gleich nur k-Vektoren von -2π bis $+2\pi$ übergeben und kürzen a.

```
matr(1,1,:)=4-cos(1/2.*(-k(:,1)+k(:,2)+k(:,3)))...
             -cos(1/2.*( k(:,1)+k(:,2)-k(:,3)))...
             -cos(1/2.*( k(:,1)-k(:,2)+k(:,3)))...
             -cos(1/2.*( k(:,1)+k(:,2)+k(:,3)))...
```

```
              +ceff_c*3*(1-cos(k(:,1)));
matr(1,2,:)=  cos(1/2.*(-k(:,1)+k(:,2)+k(:,3)))...
              -cos(1/2.*( k(:,1)+k(:,2)-k(:,3)))...
              +cos(1/2.*( k(:,1)-k(:,2)+k(:,3)))...
              -cos(1/2.*( k(:,1)+k(:,2)+k(:,3)));
matr(1,3,:)=  cos(1/2.*(-k(:,1)+k(:,2)+k(:,3)))...
              +cos(1/2.*( k(:,1)+k(:,2)-k(:,3)))...
              -cos(1/2.*( k(:,1)-k(:,2)+k(:,3)))...
              -cos(1/2.*( k(:,1)+k(:,2)+k(:,3)));
matr(2,1,:)=  cos(1/2.*(-k(:,1)+k(:,2)+k(:,3)))...
              -cos(1/2.*( k(:,1)+k(:,2)-k(:,3)))...
              +cos(1/2.*( k(:,1)-k(:,2)+k(:,3)))...
              -cos(1/2.*( k(:,1)+k(:,2)+k(:,3)));
matr(2,2,:)=4-cos(1/2.*(-k(:,1)+k(:,2)+k(:,3)))...
              -cos(1/2.*( k(:,1)+k(:,2)-k(:,3)))...
              -cos(1/2.*( k(:,1)-k(:,2)+k(:,3)))...
              -cos(1/2.*( k(:,1)+k(:,2)+k(:,3)))...
              +ceff_c*3*(1-cos(k(:,2)));
matr(2,3,:)= -cos(1/2.*(-k(:,1)+k(:,2)+k(:,3)))...
              +cos(1/2.*( k(:,1)+k(:,2)-k(:,3)))...
              +cos(1/2.*( k(:,1)-k(:,2)+k(:,3)))...
              -cos(1/2.*( k(:,1)+k(:,2)+k(:,3)));
matr(3,1,:)=  cos(1/2.*(-k(:,1)+k(:,2)+k(:,3)))...
              +cos(1/2.*( k(:,1)+k(:,2)-k(:,3)))...
              -cos(1/2.*( k(:,1)-k(:,2)+k(:,3)))...
              -cos(1/2.*( k(:,1)+k(:,2)+k(:,3)));
matr(3,2,:)= -cos(1/2.*(-k(:,1)+k(:,2)+k(:,3)))...
              +cos(1/2.*( k(:,1)+k(:,2)-k(:,3)))...
              +cos(1/2.*( k(:,1)-k(:,2)+k(:,3)))...
              -cos(1/2.*( k(:,1)+k(:,2)+k(:,3)));
matr(3,3,:)=4-cos(1/2.*(-k(:,1)+k(:,2)+k(:,3)))...
              -cos(1/2.*( k(:,1)+k(:,2)-k(:,3)))...
              -cos(1/2.*( k(:,1)-k(:,2)+k(:,3)))...
              -cos(1/2.*( k(:,1)+k(:,2)+k(:,3)))...
              +ceff_c*3*(1-cos(k(:,3)));
```

Jetzt wollen wir die Koeffizienten der Polynome 3. Ordnung berechnen. Dazu reservieren wir zuerst Speicherplatz.

```
L=nan(size(k,1),3);
```

Danach tragen wir die Koeffizienten nach Gleichung 3.2 ein. In den Spalten stehen dabei die Potenzen in der Reihenfolge λ^2, λ^1 und λ^0.

```
L(:,1)=-matr(1,1,:)-matr(2,2,:)-matr(3,3,:);
L(:,2)= matr(1,1,:).*(matr(2,2,:)+matr(3,3,:))...
       -matr(1,2,:).*matr(2,1,:)-matr(1,3,:).*matr(3,1,:)...
       +matr(2,2,:).*matr(3,3,:)-matr(2,3,:).*matr(3,2,:);
L(:,3)=-matr(1,1,:).*...
       (matr(2,2,:).*matr(3,3,:)-matr(2,3,:).*matr(3,2,:))...
       +matr(1,2,:).*...
       (matr(2,1,:).*matr(3,3,:)-matr(2,3,:).*matr(3,1,:))...
       -matr(1,3,:).*...
       (matr(2,1,:).*matr(3,2,:)-matr(2,2,:).*matr(3,1,:));
```

Mit der Funktion `cardan` können wir dann sofort die zugehörigen Nullstellen berechnen.

```
eigenw=cardan(L(:,1),L(:,2),L(:,3));
```

Nach Gleichung 2.5 müssen die Eigenwerte noch um einen Faktor korrigiert und die Wurzel gezogen werden, um $\omega\sqrt{\frac{m}{C}}$ zu erhalten.

```
omega=sqrt(2.*eigenw./3);
```

3.6 get_dos.m

In `get_dos` werden die k-Vektoren erstellt, geprüft, ob sie in der ersten Brillouin-Zone liegen, die Frequenzen berechnet und das Histogramm für die DoS errechnet. Dazu werden folgende Parameter benötigt.

- `brilCorners` enthält die Eckpunkte der ersten Brillouin-Zone mit x, y und z in den Spalten und je einem Punkt pro Zeile.
- `getFreq` ist ein Function Handle für eine Funktion, die zu gegebenen k-Vektoren in einer $n \times 3$-Matrix die Frequenzen berechnet und ebenfalls in einem $n \times 3$-Array zurückgibt.
- `divs` gibt die Anzahl der Unterteilungen der Brillouin-Zone pro Raumrichtung an. Es werden also `divs^3` Punkte im k-Raum erzeugt.
- `divsPerBlock` gibt die Anzahl der Unterteilungen an, die gleichzeitig bearbeitet werden sollen. Das ist bei der Berechnung sehr vieler Werte wichtig, weil eine gleichzeitige Auswertung aller Punkte zu viel Speicher verbrauchen würde.
- `num_histbuckets` ist die Anzahl der Unterteilungen des Histogramms.

- `randompoints` ist `true`, wenn die Punkte im k-Raum zufällig ausgewählt werden sollen. Wird hier `false` übergeben, sind die ausgewerteten k-Punkte gleichverteilt auf einem Gitter mit `divs` Unterteilungen pro Raumrichtung.
- `savemem` gibt an, ob ω-Werte, die bereits im Histogramm gespeichert sind, aus Speicherplatzgründen gleich wieder freigegeben werden sollen. Wird hier `true` übergeben, muss zusätzlich `omegamax` angegeben werden. Ansonsten wird der maximale ω-Wert automatisch errechnet.
- `omegamax` ist die Frequenz, bis zu der das Histogramm berechnet wird, wenn `savemem==true`.

Zurückgegeben werden die x-Werte (`limits`), die y-Werte der DoS (`buckets`) und die Anzahl der k-Vektoren, die in der ersten Brillouin-Zone waren (`numk`).

```
function [limits,buckets,numk]=get_dos(brilCorners,getFreq,divs,
    divsPerBlock,num_histbuckets,randompoints,savemem,omegamax)
```

Als erstes berechnen wir, wie viele Blöcke wir pro Raumrichtung brauchen werden. Dazu teilen wir einfach die Anzahl der Gesamtunterteilungen durch die Zahl der Teilungen pro Block und runden auf.

```
nblocks=ceil(divs/divsPerBlock);
```

Damit wir immer gleich große Blöcke bearbeiten, teilen wir die Gesamtzahl der Unterteilungen wieder durch diese Zahl und runden auf. Das kann dazu führen, dass wir insgesamt mehr als `divs` Unterteilungen berechnen, was aber für unsere Anwendung keinen Unterschied macht.

```
divsPerBlock=ceil(divs/nblocks);
```

Als nächstes reservieren wir Platz zum Speichern der Frequenzen. Wenn wir alle ω-Werte zwischenspeichern sollen (`savemem==false`), brauchen wir für jeden Block ein Feld, das `3*divsPerBlock^3` Elemente speichern kann. Wegen der leichteren Indizierung in den Schleifen verwenden wir dazu ein 5-dimensionales Array.

Sollen die Ergebnisse nicht zwischengespeichert werden (also `savemem==true`), brauchen wir nur die Frequenzen eines Blocks auf einmal zu speichern. Es reicht dazu ein `3*divsPerBlock^3` großes Feld. Außerdem kennen wir in dem Fall bereits die maximale Frequenz, können damit die x-Werte für das Histogramm berechnen und das Feld für die Histogrammdaten mit Nullen initialisieren.

```
if (~savemem)
```

```
    omega=nan(nblocks,nblocks,nblocks,divsPerBlock.^3,3);
else
    omega=nan(3*divsPerBlock.^3,1);
    limits=linspace(0,omegamax,num_histbuckets)';
    buckets=zeros(size(limits));
end
```

Die nächsten Teile befinden sich in 3 verschachtelten Schleifen, die die Boxen in allen 3 Raumrichtungen durchlaufen.

```
for k=1:nblocks
    for l=1:nblocks
        for m=1:nblocks
```

Wenn wir zufällige k-Werte verwenden sollen, können wir einfach eine Matrix der Größe `divsPerBlock^3×3` mit Zufallszahlen von -2π bis $+2\pi$ erzeugen. Wenn nicht, müssen wir den aktuellen Bereich pro Achse in `divsPerBlock` Einheiten teilen und dann mit **meshgrid** alle Punktkombinationen erzeugen. Die drei von **meshgrid** zurückgegebenen Matrizen als Spalten ergeben dann jeweils die x-, y- und z-Werte der zu prüfenden Punkte.

```
if (randompoints)
    testpts=rand(divsPerBlock.^3,3)*4*pi-2*pi;
else
    testlinx=linspace(((k-1)*divsPerBlock+1)*4*pi/(nblocks*
       divsPerBlock),k*divsPerBlock*4*pi/(nblocks*divsPerBlock),
       divsPerBlock)-2*pi;
    testliny=linspace(((l-1)*divsPerBlock+1)*4*pi/(nblocks*
       divsPerBlock),l*divsPerBlock*4*pi/(nblocks*divsPerBlock),
       divsPerBlock)-2*pi;
    testlinz=linspace(((m-1)*divsPerBlock+1)*4*pi/(nblocks*
       divsPerBlock),m*divsPerBlock*4*pi/(nblocks*divsPerBlock),
       divsPerBlock)-2*pi;
    [testptsx,testptsy,testptsz]=meshgrid(testlinx, testliny,
       testlinz);
    testpts=[testptsx(:),testptsy(:),testptsz(:)];
end
```

Jetzt müssen wir prüfen, ob die erzeugten Punkte in der ersten Brillouin-Zone liegen.

```
inpts=inbril(brilCorners,testpts);
```

Nun können wir die zugehörigen Frequenzen berechnen. Wie wir diese speichern, hängt wieder davon ab, ob wir alle Werte aufbewahren wollen oder nicht. Sollen die Zahlen gespeichert bleiben, brauchen wir nur an die entsprechende Stelle im `omega`-Feld schreiben. Sollen sie nicht gespeichert bleiben, müssen wir die Werte zwischenspeichern, sie mit `histc` zum bestehenden Histogramm hinzufügen und danach den Zwischenspeicher für den nächsten Schleifendurchlauf wieder auf **`nan`** setzen.

```
if(~savemem)
    omega(k,l,m,1:size(inpts,1),1:3)=real(getFreq(inpts));
else
    omega(1:numel(inpts))=real(getFreq(inpts));
    buckets=buckets+histc(omega,limits);
    omega(:)=nan;
end
```

Hier enden alle drei Schleifen über die Boxen in den Raumrichtungen.

```
        end
    end
end
```

Wenn wir bis jetzt die Frequenzen alle nur gespeichert haben, müssen wir noch das Histogramm erstellen. Dazu teilen wir den Bereich von 0 bis knapp über dem Maximalwert aller Frequenzen in `num_histbuckets` Teile auf und erstellen dann mit `histc` ein Histogramm. Damit **`max`** und `histc` nicht dimensionsweise arbeiten, machen wir zuerst aus `omega` einen Spaltenvektor.

```
if (~savemem)
    omega=omega(:);
    limits=linspace(0,max(omega).*1.05,num_histbuckets)';
    buckets=histc(omega,limits);
end
```

Zum Schluss müssen wir das Histogramm noch normieren. Dazu berechnen wir zunächst, wie viele Frequenzwerte wir insgesamt berechnet haben.

```
numk=sum(buckets);
```

Weil die Fläche unter der DoS-Kurve in den verwendeten dimensionslosen Einheiten 6 sein muss, rechnen wir die aktuelle Fläche A aus und multiplizieren dann alle Werte mit $\frac{6}{A}$. Nicht normiert erzeugt jeder Punkt im Histogramm eine Fläche von Δx, dem

Abstand zweier Punkte auf der x-Achse. Weil die Werte auf der x-Achse alle den gleichen Abstand voneinander haben, ist Δx einfach `limits(2)-limits(1)`. Für die gesamte Fläche muss dieser Wert lediglich mit der Gesamtzahl der Frequenzwerte multipliziert werden.

```
buckets=buckets.*6./(numk.*(limits(2)-limits(1)));
```

Weil `numk` die Zahl der k-Vektoren sein soll und wir 3 mal so viele Frequenzen errechnet haben, müssen wir die Zahl noch um den Faktor 3 korrigieren.

```
numk=numk./3;
```

3.7 disp_rel.m

Um auch die Dispersionsrelation in verschiedenen Raumrichtungen darstellen zu können, haben wir die Funktion `disp_rel` geschrieben. Sie braucht folgende Paramter.

- `points_per_length` gibt an, wie viele Punkte pro Längeneinheit im k-Raum berechnet werden sollen.
- `corners` enthält die Punkte im k-Raum in der Reihenfolge, wie sie im Ergebnisvektor stehen sollen. x, y und z stehen dabei in den Spalten.
- `getFreq` ist ein Function Handle, das zu einer $n \times 3$-Matrix aus k-Vektoren in Zeilen ein gleich großes Feld mit den zugehörigen ω-Werten berechnet.

Zurückgegeben werden in `points` zeilenweise die Frequenzen (jeweils 3, aufsteigend sortiert) und in `corner_idx` die Zeilenindizes von `points`, an denen die zu den k-Punkten in `corners` gehörigen Frequenzen stehen.

```
function [points,corner_idx]=disp_rel(points_per_length,corners,
   getFreq)
```

Als erstes berechnen wir die Längen der Strecken zwischen den Punkten in `corners`. Dazu bestimmen wir einfach die Vektoren, die die Punkte verbinden, und berechnen deren Länge. `sum(A.^2,2)` berechnet dabei $\vec{A}^2$, wenn `A` ein Zeilenvektor ist.

```
lengths=sqrt(sum((corners(1:end-1,:)-corners(2:end,:)).^2,2));
```

Dann bestimmen wir die Anzahl der Punkte, die wir pro Strecke auswerten müssen. Dazu multiplizieren wir die Streckenlängen einfach mit `points_per_length`.

```
linepoints=round(points_per_length*lengths);
```

Weil wir jetzt wissen, wie viele Punkte wir berechnen müssen, können wir die Matrix der k-Vektoren initialisieren und `corner_idx` ausrechnen. Die Indizes der Eckpunkte sind dabei einfach die kummulative Summe aller Strecken davor, wobei wir hier von den Streckenpunkten 1 abziehen, weil sonst die Eckpunkte doppelt vorkommen würden (einmal als Endpunkt der vorherigen und einmal als Anfangspunkt der nächsten Strecke). Weil der erste Eckpunkt immer bei 1 liegt, müssen wir diesen Wert manuell vorne einschieben und ihn zur kummulativen Summe addieren.

```
kvec=nan(sum(linepoints)-numel(linepoints),3);
corner_idx=[1;cumsum(linepoints-1)+1];
```

Jetzt wollen wir alle zu berechnenden k-Vektoren ermitteln. Dafür gehen wir in einer Schleife über `k` alle Verbindungslinien in `linepoints` ab und erzeugen für jede Linie einen linearen Verlauf vom Anfangs- zum Endpunkt. Die Indizes dafür sind bereits in `corner_idx` gespeichert.

```
for k=1:size(linepoints,1)
    kx=linspace(corners(k,1),corners(k+1,1),linepoints(k))';
    ky=linspace(corners(k,2),corners(k+1,2),linepoints(k))';
    kz=linspace(corners(k,3),corners(k+1,3),linepoints(k))';
    kvec(corner_idx(k):corner_idx(k+1),:)=[kx,ky,kz];
end
```

Nun können wir für alle k-Vektoren die Frequenzen gleichzeitig berechnen. Um die Linien leichter plotten zu können, sortieren wir ω noch pro Zeile aufsteigend.

```
points=sort(real(getFreq(kvec)),2);
```

3.8 Skripte zum Aufruf

Zum Aufruf der vorgestellten Funktionen haben wir noch die Skripte `disp_call.m` und `dos_call.m` geschrieben, damit sich Parameter leichter einstellen lassen. Diese sind eigentlich selbsterklärend und zusammen mit den anderen kompletten Listings im Anhang zu finden.

4 Ergebnisse

4.1 Density of States

Die Zustandsdichte wurde mit verschiedenen effektiven Federkonstanten zu den nächst-nächsten Nachbarn berechnet, um den Einfluss dieses Parameters zu demonstrieren. Die Abbildung 3 zeigt die Ergebnisse.

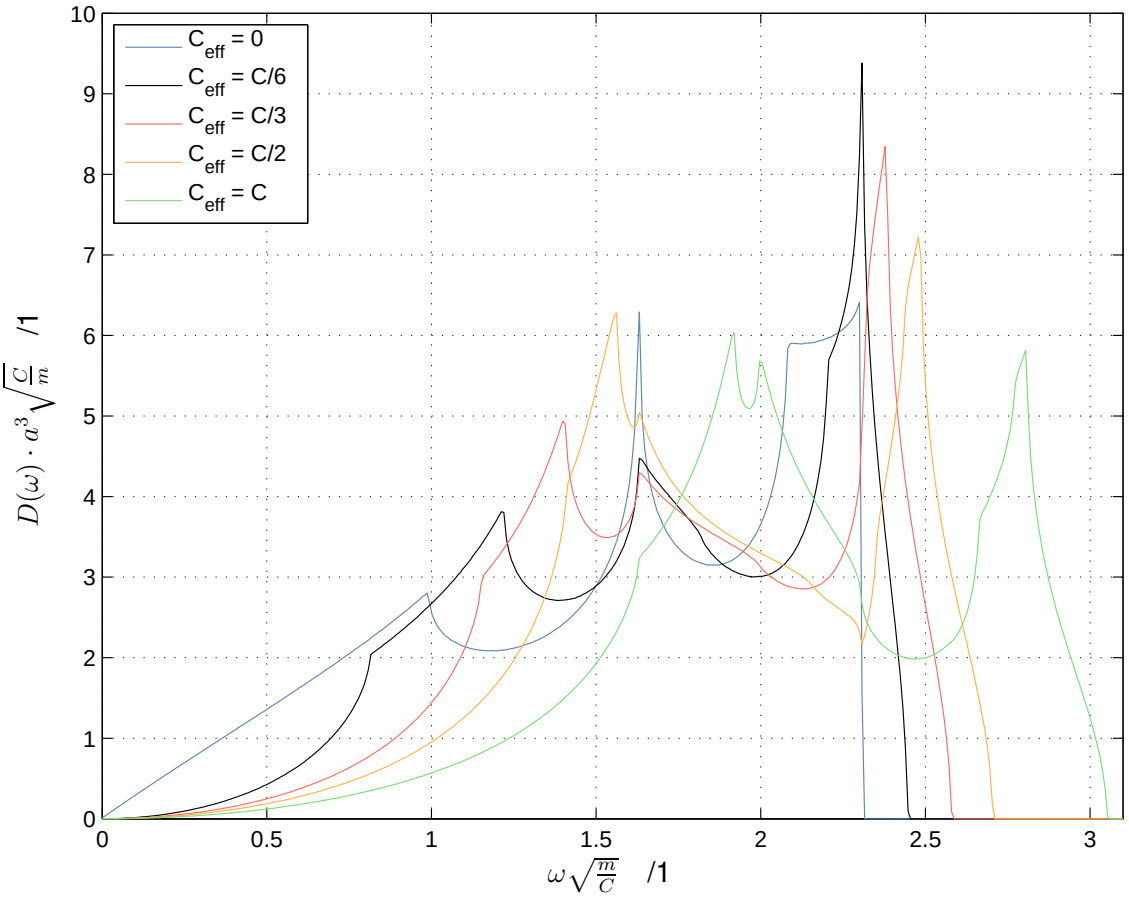

Abbildung 3: DoS für bcc bei verschiedenen effektiven Federkonstanten der nächst-nächsten Nachbarn. k-Werte pro Linie $\approx 250\,000\,000$, Berechnungszeit ≈ 51 min

Es ist deutlich zu erkennen, dass bei keiner Kopplung der nächst-nächsten Nachbarn die Zustandsdichte nicht wie erwartet mit ω^2, sondern linear in ω ansteigt. Koppelt man die Atome auch noch zu den nächst-nächsten Nachbarn, ist der Anstieg aber quadratisch. Die Form insgesamt ist stark von der effektiven Federkonstate abhängig.

Ein Vergleich der Form mit einer Berechnung für α-Fe mit dem Born-von Kármán-Modell in Abbildung 4 zeigt, dass $C_{\text{eff}} = C/6$ dafür wohl die beste Wahl ist. Deshalb ist die DoS in Abbildung 5 für $C_{\text{eff}} = C/6$ nochmals mit mehr berechneten Werten dargestellt.

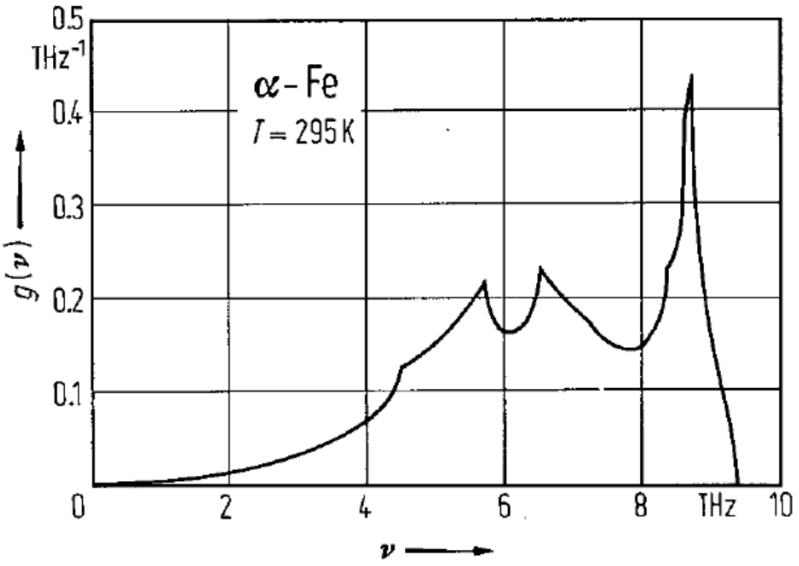

Abbildung 4: DoS von α-Eisen, berechnet mit dem Born-von Kármán-Modell aus den Vorlesungsunterlagen von Prof. Hadley, ursprünglich aus „Springer Materials: Landholt Boernstein Database“

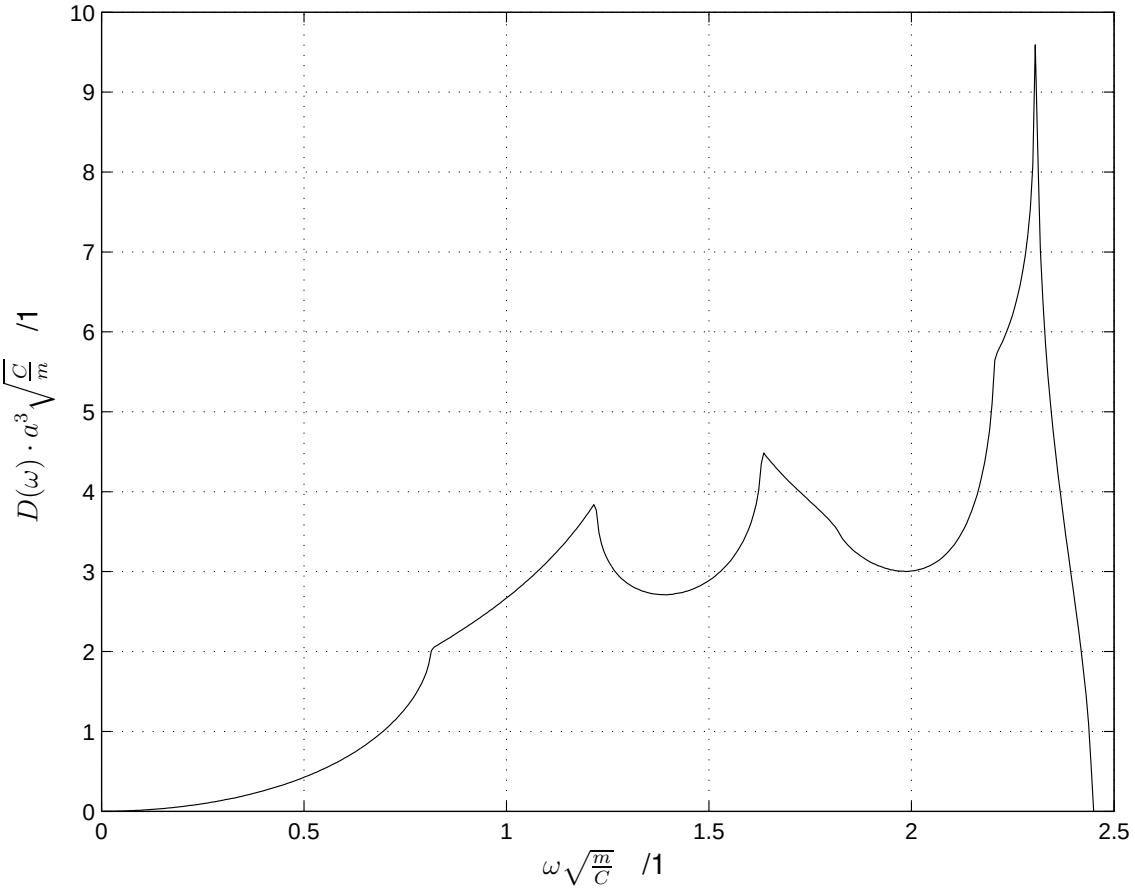

Abbildung 5: DoS für bcc bei $C_{\text{eff}} = C/6$. 843 749 932 k-Werte, Berechnungszeit $\approx$ 36 min

4.2 Dispersionsrelation

Die Kopplung zu den nächst-nächsten Nachbarn hat auch Auswirkungen auf die Dispersionsrelation. Wie in Abbildung 6 zu sehen ist, spalten sich die Transversal- und Longitudinalmoden zwischen Γ und H auf, sobald $C_{\text{eff}} > 0$. Eine Frequenz bleibt bestehen (die 5 Linien überlagern sich dort), die andere erhöht sich mit steigender effektiver Federkonstante. Auch sonst überall erkennt man, dass eine höhere effektive Federkonstante zu höhren Frequenzen führt.

Weiters fällt auch zwischen Γ und N auf, dass bei $C_{\text{eff}} = 0$ eine Mode immer die Frequenz 0 hat und nur die 2 anderen Moden einen Anstieg zeigen. Auch hier bleibt eine der Frequenzen konstant (die 5 Linien überlagern sich auf der 4. Linie von unten), aber die beiden anderen haben bei steigender effektiver Federkonstante immer höhere Frequenzen.

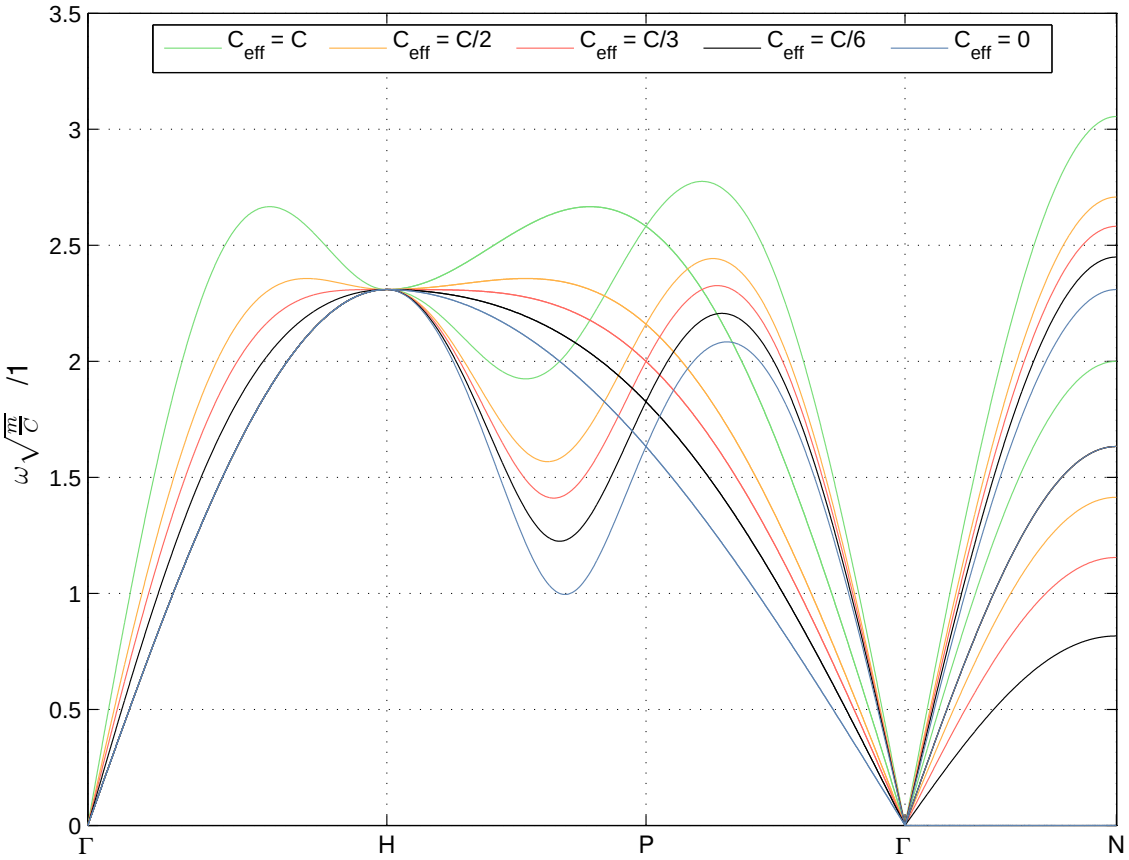

Abbildung 6: Dispersionsrelation von bcc bei verschiedenen effektiven Federkonstanten der nächst-nächsten Nachbarn. Auf der Strecke von Γ nach H ist die unterste Linie eine Überlagerung aller 5 vorkommenden Linien. Ebenso die vierte Line von unten auf der Strecke von Γ nach N.

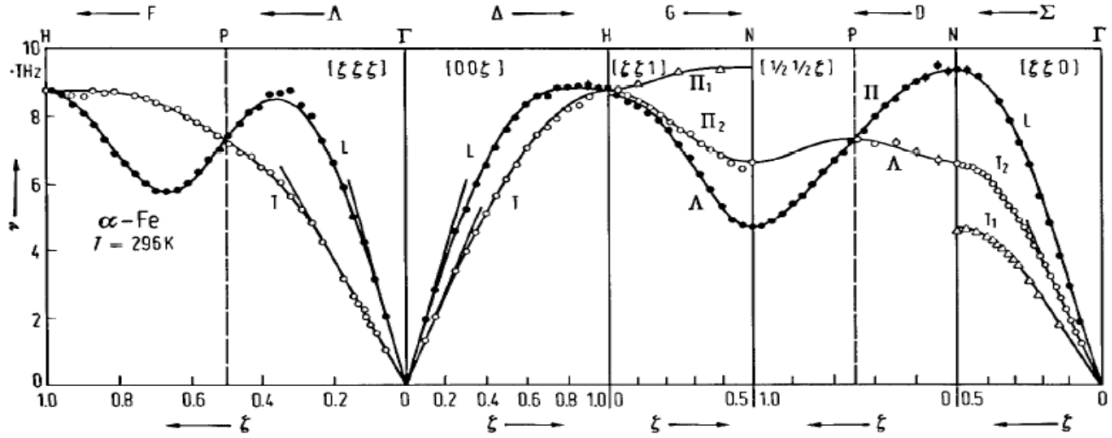

Abbildung 7: Dispersionsrelation von α-Eisen bei 296 K, berechnet mit dem fünfte-Nachbarn Born-von Kármán-Modell (durchgezogene Linie) aus den Vorlesungsunterlagen von Prof. Hadley, ursprünglich aus „Springer Materials: Landholt Boernstein Database“

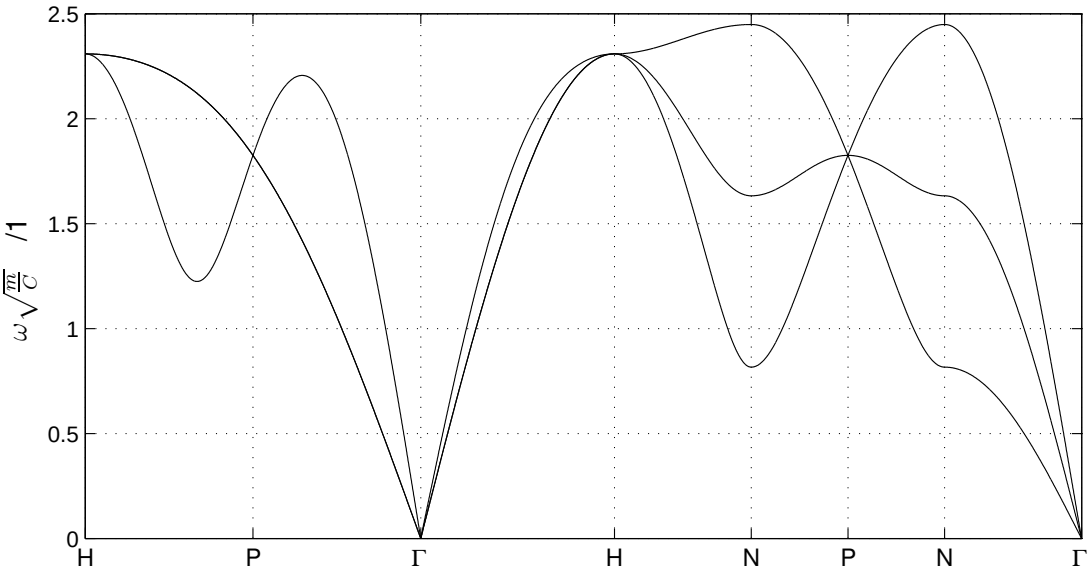

Abbildung 8: Dispersionsrelation von bcc bei $C_{\text{eff}} = C/6$.

Die zuvor für die DoS gewählte effektive Federkonstante von $C_{\text{eff}} = C/6$, die gute Übereinstimmung mit anderen Berechnungen zeigte, wurde auch hier verwendet, um die Dispersionsrelation mit einer Berechnung nach dem Born-von Kármán-Modell zu vergleichen. Betrachtet man Abbildungen 7 und 8, sind sehr gute Übereinstimmungen zu sehen. Es fehlt lediglich die auf der Strecke N–P–N in unserer Berechnung von oben nach unten laufende Mode in der Referenzberechnung, was dort aber nur der besseren Übersicht dient, da die Strecke N–P–N ja symmetrisch ist.

A Komplette Listings

Listing 1: cardan.m

```
% cardan - berechnet die Lsgen von kubischen Polynomen der Form
%          x^3+b*x^2+c*x+d=0 mit den cardanischen Formeln
% In:  b,c,d - Spaltenvektoren der Koeffizienten b, c und d
% Out: lsg   - nx3-Matrix mit den 3 Loesungen in den Zeilen
function lsg = cardan(b,c,d)
p=c-b.^2./3;
q=2.*b.^3./27-b.*c./3+d;
dis=(q./2).^2+(p./3).^3;
dis1=(dis>0);
dis2=(dis<0);
dis3=(dis==0);
lsg=nan(numel(dis),3);

sqdis=sqrt(dis(dis1));
u=nthroot(-q(dis1)/2+sqdis,3);
v=nthroot(-q(dis1)/2-sqdis,3);
lsg(dis1,1)=u+v;
lsg(dis1,2)=(-u-v)./2+(u-v)./2.*1i.*sqrt(3);
lsg(dis1,3)=conj(lsg(dis1,2));

arco=acos(-q(dis2)./2.*sqrt(-27./(p(dis2).^3)));
lsg(dis2,1)= sqrt(-4/3*p(dis2)).*cos(arco./3);
lsg(dis2,2)=-sqrt(-4/3*p(dis2)).*cos(arco./3-pi/3);
lsg(dis2,3)=-sqrt(-4/3*p(dis2)).*cos(arco./3+pi/3);

lsg(dis3,1)=nthroot(-4*q(dis3),3);
lsg(dis3,2)=-lsg(dis3,1)./2;
lsg(dis3,3)=lsg(dis3,2);

lsg=bsxfun(@minus,lsg,b./3);
```

Listing 2: inbril.m

```
% inbril - bestimmt, ob Punkte innerhalb der ersten
%          Brillouin-Zone liegen
% In:  brilCorners - Eckpunkte der Brillouin-Zone
%                    mit x, y und z in den Spalten
%      points      - Punkte, die geprueft werden sollen
%                    mit x, y und z in Spalten
% Out: inpoints    - Punkte, die in der 1. Brillouin-Zone
%                    liegen (Format wie points)
function inpoints = inbril(brilCorners,points)
tri=convhulln(brilCorners);
vecA=brilCorners(tri(:,1),:)-brilCorners(tri(:,3),:);
vecB=brilCorners(tri(:,1),:)-brilCorners(tri(:,2),:);
normVec=cross(vecA,vecB);
PointPerFace=brilCorners(tri(:,2),:);
wrongDir=sum(PointPerFace.*normVec,2)>0;
normVec(wrongDir,:)=-normVec(wrongDir,:);
fn=sum(PointPerFace.*normVec,2);
inlog=false(size(points,1),1);
blockel=floor(1E6/size(normVec,1));
blo=floor(size(points,1)/blockel);
for k=1:blo
  inlog((k-1)*blockel+1:k*blockel)=all(bsxfun(@minus,normVec*
     points((k-1)*blockel+1:k*blockel,:)',fn)>=0,1);
end
inlog(blo*blockel+1:end)=all(bsxfun(@minus,normVec*points(blo*
   blockel+1:end,:)',fn)>=0,1);
inpoints=points(inlog,:);
```

Listing 3: getFrequencies.m

```
% getFrequencies - berechnet die Frequenzen zu gegebenen
%                   k-Vektoren
% In:  k          - nx3-Matrix mit den zu berechnenden k-Vektoren
%                   in Zeilen
%      ceff_c     - Verhaeltnis der effektiven Federkonstante der
%                   naechst-nachsten Nachbarn zur Federkonstante
%                   der naechsten Nachbarn
% Out: omega      - nx3-Matrix mit den Frequenzen des i-ten
%                   k-Vektors in der i-ten Zeile
function omega = getFrequencies(k,ceff_c)
matr=nan(3,3,size(k,1));

matr(1,1,:)=4-cos(1/2.*(-k(:,1)+k(:,2)+k(:,3)))-cos(1/2.*(k(:,1)+k
    (:,2)-k(:,3)))-cos(1/2.*(k(:,1)-k(:,2)+k(:,3)))-cos(1/2.*(k
    (:,1)+k(:,2)+k(:,3)))+ceff_c*3*(1-cos(k(:,1)));
matr(1,2,:)=  cos(1/2.*(-k(:,1)+k(:,2)+k(:,3)))-cos(1/2.*(k(:,1)+k
    (:,2)-k(:,3)))+cos(1/2.*(k(:,1)-k(:,2)+k(:,3)))-cos(1/2.*(k
    (:,1)+k(:,2)+k(:,3)));
matr(1,3,:)=  cos(1/2.*(-k(:,1)+k(:,2)+k(:,3)))+cos(1/2.*(k(:,1)+k
    (:,2)-k(:,3)))-cos(1/2.*(k(:,1)-k(:,2)+k(:,3)))-cos(1/2.*(k
    (:,1)+k(:,2)+k(:,3)));
matr(2,1,:)=  cos(1/2.*(-k(:,1)+k(:,2)+k(:,3)))-cos(1/2.*(k(:,1)+k
    (:,2)-k(:,3)))+cos(1/2.*(k(:,1)-k(:,2)+k(:,3)))-cos(1/2.*(k
    (:,1)+k(:,2)+k(:,3)));
matr(2,2,:)=4-cos(1/2.*(-k(:,1)+k(:,2)+k(:,3)))-cos(1/2.*(k(:,1)+k
    (:,2)-k(:,3)))-cos(1/2.*(k(:,1)-k(:,2)+k(:,3)))-cos(1/2.*(k
    (:,1)+k(:,2)+k(:,3)))+ceff_c*3*(1-cos(k(:,2)));
matr(2,3,:)= -cos(1/2.*(-k(:,1)+k(:,2)+k(:,3)))+cos(1/2.*(k(:,1)+k
    (:,2)-k(:,3)))+cos(1/2.*(k(:,1)-k(:,2)+k(:,3)))-cos(1/2.*(k
    (:,1)+k(:,2)+k(:,3)));
matr(3,1,:)=  cos(1/2.*(-k(:,1)+k(:,2)+k(:,3)))+cos(1/2.*(k(:,1)+k
    (:,2)-k(:,3)))-cos(1/2.*(k(:,1)-k(:,2)+k(:,3)))-cos(1/2.*(k
    (:,1)+k(:,2)+k(:,3)));
matr(3,2,:)= -cos(1/2.*(-k(:,1)+k(:,2)+k(:,3)))+cos(1/2.*(k(:,1)+k
    (:,2)-k(:,3)))+cos(1/2.*(k(:,1)-k(:,2)+k(:,3)))-cos(1/2.*(k
    (:,1)+k(:,2)+k(:,3)));
matr(3,3,:)=4-cos(1/2.*(-k(:,1)+k(:,2)+k(:,3)))-cos(1/2.*(k(:,1)+k
    (:,2)-k(:,3)))-cos(1/2.*(k(:,1)-k(:,2)+k(:,3)))-cos(1/2.*(k
    (:,1)+k(:,2)+k(:,3)))+ceff_c*3*(1-cos(k(:,3)));

L=nan(size(k,1),3);
L(:,1)=-matr(1,1,:)-matr(2,2,:)-matr(3,3,:);
L(:,2)=matr(1,1,:).*(matr(2,2,:)+matr(3,3,:))-matr(1,2,:).*matr
    (2,1,:)-matr(1,3,:).*matr(3,1,:)+matr(2,2,:).*matr(3,3,:)-matr
```

```
    (2,3,:).*matr(3,2,:);
L(:,3)=-matr(1,1,:).*(matr(2,2,:).*matr(3,3,:)-matr(2,3,:).*matr
    (3,2,:))+matr(1,2,:).*(matr(2,1,:).*matr(3,3,:)-matr(2,3,:).*
    matr(3,1,:))-matr(1,3,:).*(matr(2,1,:).*matr(3,2,:)-matr
    (2,2,:).*matr(3,1,:));
eigenw=cardan(L(:,1),L(:,2),L(:,3));
omega=sqrt(2.*eigenw./3);
```

Listing 4: get_dos.m

```
% get_dos - berechnet die Zustandsdichte fuer gegebene Parameter
% In:  brilCorners     - Eckpunkte der Brillouin-Zone mit x, y und
%                        z in Spalten und Eckpunkten in Zeilen
%      getFreq         - Function Handle, das zu k-Vektoren in
%                        Zeilen mit x, y und z in Spalten eine
%                        gleich grosse Matrix mit den
%                        zugehoerigen omega-Werten liefert
%      divs            - Anzahl der Unterteilungen pro
%                        Raumrichtung
%      divsPerBlock    - Anzahl der gleichzeitig zu berechnenden
%                        Unterteilungen pro Raumrichtung
%      num_histbuckets - Anzahl der Teilungen des Histogramms
%      randompoints    - true wenn die Punkte zufaellig
%                        ausgewaehlt werden sollen, false wenn
%                        die Brillouin-Zone in regelmaessige
%                        Punkte zerteilt werden soll
%      savemem         - true, wenn die omega-Werte nach dem
%                        Berechnen eines Blocks sofort wieder
%                        freigegeben werden sollen
%      omegamax        - wenn savemem==true, muss hier die max.
%                        Freq. im Histogramm uebergeben werden
% Out: limits          - Eckpunkte des Histogramms
%      buckets         - Zustandsdichte pro Punkt in limits
%      numk            - Anzahl der k-Vektoren, die in der
%                        1. Brillouin-Zone gelegen sind
function [limits,buckets,numk] = get_dos(brilCorners,getFreq,divs,
    divsPerBlock,num_histbuckets,randompoints,savemem,omegamax)
nblocks=ceil(divs/divsPerBlock);
divsPerBlock=ceil(divs/nblocks);
if (~savemem)
    omega=nan(nblocks,nblocks,nblocks,divsPerBlock.^3,3);
else
    omega=nan(3*divsPerBlock.^3,1);
    limits=linspace(0,omegamax,num_histbuckets)';
```

```
34      buckets=zeros(size(limits));
35  end
36  for k=1:nblocks
37      for l=1:nblocks
38          for m=1:nblocks
39              if (randompoints)
40                  testpts=rand(divsPerBlock.^3,3)*4*pi-2*pi;
41              else
42                  testlinx=linspace(((k-1)*divsPerBlock+1)*4*pi/(
                        nblocks*divsPerBlock),k*divsPerBlock*4*pi/(
                        nblocks*divsPerBlock),divsPerBlock)-2*pi;
43                  testliny=linspace(((l-1)*divsPerBlock+1)*4*pi/(
                        nblocks*divsPerBlock),l*divsPerBlock*4*pi/(
                        nblocks*divsPerBlock),divsPerBlock)-2*pi;
44                  testlinz=linspace(((m-1)*divsPerBlock+1)*4*pi/(
                        nblocks*divsPerBlock),m*divsPerBlock*4*pi/(
                        nblocks*divsPerBlock),divsPerBlock)-2*pi;
45                  [testptsx,testptsy,testptsz]=meshgrid(testlinx,
                        testliny, testlinz);
46                  testpts=[testptsx(:),testptsy(:),testptsz(:)];
47              end
48              inpts=inbril(brilCorners,testpts);
49              if(~savemem)
50                  omega(k,l,m,1:size(inpts,1),1:3)=real(getFreq(
                        inpts));
51              else
52                  omega(1:numel(inpts))=real(getFreq(inpts));
53                  buckets=buckets+histc(omega,limits);
54                  omega(:)=nan;
55              end
56          end
57      end
58  end
59  if (~savemem)
60      omega=omega(:);
61      limits=linspace(0,max(omega).*1.05,num_histbuckets)';
62      buckets=histc(omega,limits);
63  end
64  numk=sum(buckets);
65  buckets=buckets.*6./(numk.*(limits(2)-limits(1)));
66  numk=numk./3;
```

Listing 5: get_dos.m

```
% disp_rel - berechnet die Dispersionsrelation entlang gegebener
%             Linien im k-Raum
% In:  points_per_length - Anzahl der zu berechnenden Punkte
%                          pro Laengeneinheit im k-Raum
%      corners           - Eckpunkte, zwischen denen die zu
%                          berechnenden Linien liegen in Zeilen
%      getFreq           - Function Handle der Funktion, die zu
%                          k-Vektoren die Frequenzen berechnet
% Out: points            - y-Werte der Dispersionsrelation, je
%                          Zeile aufsteigend sortiert
%      corner_idx        - Zeilenindizes von points, an denen
%                          die uebergebenen Eckpunkte liegen
function [points,corner_idx]=disp_rel(points_per_length,corners,
    getFreq)
lengths=sqrt(sum((corners(1:end-1,:)-corners(2:end,:)).^2,2));
linepoints=round(points_per_length*lengths);
kvec=nan(sum(linepoints)-numel(linepoints),3);
corner_idx=[1;cumsum(linepoints-1)+1];
for k=1:size(linepoints,1)
    kx=linspace(corners(k,1),corners(k+1,1),linepoints(k))';
    ky=linspace(corners(k,2),corners(k+1,2),linepoints(k))';
    kz=linspace(corners(k,3),corners(k+1,3),linepoints(k))';
    kvec(corner_idx(k):corner_idx(k+1),:)=[kx,ky,kz];
end
points=sort(real(getFreq(kvec)),2);
```

Listing 6: disp_call.m

```
% Function Handle zum Berechnen der Frequenzen
getFreq=@(k) getFrequencies(k,1/6);
% Punkte der Dispersionsrelation
corners=[2*pi,2*pi,2*pi;pi,pi,pi; 0, 0, 0;0,0,2*pi;...
           pi,  pi,2*pi;pi,pi,pi;pi,pi, 0;0,0,0];
% Beschriftung der Punkte auf der x-Achse
labels={'H','P','Gamma','H','N','P','N','Gamma'};

% Funktionswerte und x-Werte der Eckpunkte berechnen
[p,idx]=disp_rel(100,corners,getFreq);

% alle x-Werte zum Zeichnen der Dispersionsrelation
xpoints=0:size(p,1)-1;

% neue weisse Figure
figure('color','white');
% alle 3 Linien der Dispersionskurve zeichnen
plot(xpoints,p(:,1),'-k',xpoints,p(:,2),'-k',xpoints,p(:,3),'-k');
% Gitter auf Eckpunkte setzen und richtige Labels einfuegen
set(gca,'XTick',idx-1,'XTickLabel',labels);
grid on
xlim([0,numel(xpoints)]);
ylabel('\omega (m/C)^{(1/2)}');
```

Listing 7: dos_call.m

```
randompoints=true;      % Punkte zufaellig auswaehlen?
savemem=true;           % berechnete Werte wieder freigeben?
omegamax=2.5;           % maximales Omega des Histogramms
num_histbuckets=400;    % Anzahl der Histogrammteilungen
divs=500;               % Unterteilungen pro Raumrichtung

% maximale gleichzeitig bearbeitete Unterteilungen
% pro Raumrichtung
divsPerBlock=100;
% Eckpunkte der Brillouin-Zone
brilCorners=...
    [   0  ,    0  ,   0  ,    0  , 2*pi , -2*pi ,  pi,...
       -pi ,    pi ,   pi ,    pi ,  -pi ,   -pi , -pi;...
        0  ,    0  , 2*pi , -2*pi ,   0  ,    0  ,  pi,...
        pi ,   -pi ,   pi ,   -pi ,   pi ,   -pi , -pi;...
      2*pi , -2*pi ,   0  ,    0  ,   0  ,    0  ,  pi,...
        pi ,    pi ,  -pi ,   -pi ,  -pi ,    pi , -pi;]';

% Function Handle fuer die Frequenzen
getFreq=@(k) getFrequencies(k,1/6);
tic
% DoS berechnen
[limits,buckets,numk]=get_dos(brilCorners,getFreq,divs,
    divsPerBlock,num_histbuckets,randompoints,savemem,omegamax);
time=toc;
% Statistik ausgeben
disp(['Calculating Frequencies for ',num2str(numk),' k-Values took
    ',num2str(time),'s.'])
disp(['(',num2str(divs.^3-numk),' Values were outside the 1st
    Brillouin-Zone)']);

% Weisse Figure erstellen, DoS zeichnen und formatieren
figure('color','white');
plot(limits,buckets,'-k');
grid on;
xlabel('\omega (m/C)^{(1/2)}');
ylabel('D(\omega) a^3 (C/m)^{1/2}');
```