Thermodynamic properties of NiTiHf: Cluster Expansion and Monte-Carlo Simulation

Diplomarbeit

Thomas Dengg, Montanunivesität Leoben

Juni, 2013

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

AFFIDAVIT

I declare in lieu of oath, that I wrote this thesis and performed the associated research myself, using only literature cited in this volume.

13.06.2013
Datum

Unterschrift

Inhaltsverzeichnis

Danksagung

Ich möchte mich sehr herzlich bei allen Mitarbeitern des Materials Center Leoben bedanken: Bei dem Leiter der Simulationsabteilung Herrn Dipl.-Ing. Dr. Werner Ecker für die Möglichkeit meine Diplomarbeit auf diesem Institut zu schreiben. Bei Herrn Dr. Peter Supancic, dem Betreuer der Diplomarbeit, für die Unterstützung während der Arbeit und der Hilfe beim Erstellen dieser. Außerdem möchte ich mich bei Herrn Dr. Jürgen Spitaler bedanken, der mich unterstützt hat und sich immer Zeit genommen hat Fragen zu beantworten. Der Österreichischen Bundesregierung (insbesondere dem Bundesministerium für Verkehr, Innovation und Technologie und dem Bundesministerium für Wirtschaft, Familie und Jugend) sowie dem Land Steiermark, vertreten durch die Österreichische Forschungsförderungsgesellschaft mbH und die Steirische Wirtschaftsförderungsgesellschaft mbH, wird für die finanzielle Unterstützung der Forschungsarbeiten im Rahmen des von der Materials Center Leoben Forschung GmbH abgewickelten K2 Zentrums für "Materials, Processing and Product Engineering im Rahmen des Österreichischen COMET Kompetenzzentren Programms sehr herzlich gedankt. Zuletzt möchte ich mich besonders bei meinen Eltern und bei meinen Freunden bedanken, die mich während des Studiums immer unterstützt haben.

Kapitel 1

Einführung

Seit der Entdeckung des Formgedächtniseffekts haben sich zahlreiche Anwendungsbereiche vor allem in den Bereichen der Medizintechnik, Raumfahrt und Automobilindustrie für Legierungen, die diesen Effekt zeigen, ergeben [1]. Die Grundlage des Effekts liegt in der Martensitischen Umwandlung einer höhersymmetrischen Hochtemperaturphase in eine niedersymmetrische Tieftemperaturphase, welche eine makroskopische Formänderung zur Folge hat. Die am häufigsten eingesetzte Legierung, die diese Eigenschaften trägt, ist eine binäre Legierung aus Nickel und Titan, auch Nitinol genannt. Diese weist jedoch eine niedrige Umwandlungstemperatur von etwa 60°C auf. Um die Umwandlungstemperatur zu erhöhen kann Hafnium zulegiert werden [2]. Dies führt zum System NiTiHf, welches die Grundlage für die Untersuchungen dieser Arbeit darstellt. Ziel dieser Arbeit ist eine theoretische Untersuchung der thermodynamischen Eigenschaften des Systems Nickel-Titan-Hafnium, genauer der B19' Hochtemperaturphase derselben, mit Hilfe von ab-initio Methoden [3]. Die Methoden zur Bestimmung der thermodynamischen Daten reichen von Dichtefunktionaltheorie (DFT) zur Ermittlung der Grundzustandsenergie, über Cluster-Expansion (CE), um die Abhängigkeit der Grundzustandsenergie von der Zusammensetzung der Legierung darzustellen, bis hin zur Monte Carlo (MC) Simulation zur Erfassung der Temperaturabhängigkeit. Die Cluster-Expansion wurde im Vorfeld der Arbeit durch Jürgen Spitaler durchgeführt [4]. Aufbauend auf diese werden Monte-Carlo-Simulationen durchgeführt, um den Einfluss der Temperatur auf die Stabilitätsbereiche der Phasen zu analysieren. Die DFT Grundzustandsrechnungen wurden mit dem auf Dichtefunktionaltheorie basierenden Code Wien2k [5] durchgeführt. Die darauf aufbauende CE sowie die MC-Simulationen wurden mit dem Programmpaket ATAT (Alloy-Theoretic-Automated-Toolkit) [6] durchgeführt. Als Schnittstelle zwischen den Grundzustandsrechnungen und der CE wurde ATAT@wien2k [7] verwendet. Weiters wurde ein eigener Code zur MC Simulation von binären Legierungen im semi-großkanonischen Ensemble programmiert. In der vorliegenden Arbeit werden zunächst die für die ab-initio Be-

handlung der Thermodynamik von Legierungen nötigen Grundlagen erarbeitet. Anfangs werden die Grundzüge der Dichtefunktionaltheorie erläutert. Nach einem kurzen Überblick über die Grundlagen der Thermodynamik von Phasenumwandlungen wird näher auf die statistische Thermodynamik eingegangen, die mittels der Methode der Monte Carlo Simulation behandelt wird. Darauf aufbauend wird die Methode der Thermodynamischen Integration (TI) eingeführt und einige Überlegungen zu Lösung von numerischen Problemen angestellt. Nach Erarbeitung der theoretischen Grundlagen wird ein einfaches Beispiel am System Titan-Aluminium vorgestellt, welches die praktische Durchführung der Simulation veranschaulichen soll. Es folgt eine Darstellung der im Laufe der Arbeit erhaltenen Ergebnisse. Unter anderem werden die Ergebnisse der Konvergenztests und der Berechnung von Phasenübergängen in der Hochtemperaturphase des Systems NiTiHf vorgestellt. Weiters wird der eigens geschriebene MC Code vorgestellt und einige Ergebnisse mit ATAT verglichen.

Kapitel 2

Theorie

2.1 Der Formgedächtniseffekt

Wissenschaftlich untersucht wurde der Formgedächtniseffekt von NiTi erstmals durch Buehler [8], angeregt durch eine Entdeckung, die Schiffsbauer in einem amerikanischen U-Bootwerk beim Schweißen von Nickel-Titan Legierungen im Jahr 1953 machten [9]. Durch Erwärmen der Schweißstellen wurden die zuvor gebogenen Bleche wieder vollkommen flach. Dieses Verhalten ist heute als Einwegeffekt bekannt.

2.1.1 Martensitische Umwandlung

Im Allgemeinen wird als Martensitische Umwandlung ein Phasenübergang ohne Diffusions-Vorgänge von einer Hochsymmetriephase in eine Phase niedriger Symmetrie bezeichnet. Die Bezeichnung stammt eigentlich vom Sytem Fe-C und bezeichnet hier die Umwandlung von Austenit (kfz) nach Martensit (trz) welche bei hoher Abkühlgeschwindigkeit eintritt. Aufgrund der hohen Abkühlgeschwindigkeit kann der überschüssige Kohlenstoff im Gitter nicht schnell genug abdiffundieren und im Gitter bilden sich Spannungen. Um diese abzubauen, wird das Gitter tetragonal verzerrt und es bilden sich sogenannte Zwillingsgrenzen an invarianten Ebenen, auch Habitusebenen genannt. Die Zwillinge gewährleisten eine Erhaltung der äußeren Form durch Abbau von inneren Spannungen.

Die martensitische Umwandlung ist eine Umwandlung 1. Ordnung und findet somit nicht schlagartig bei einer kritischen Temperatur statt, sondern in einem Temperaturbereich zwischen A_s (Martensit Starttemperatur) und A_f (Martensit Endtemperatur).

2.1.2 Einwegeffekt

Beim Einwegeffekt wird durch Aufbringen einer Scherspannung eine Martensitische Transformation induziert. Zunächst wird der verzwillingte Martensit elastisch verformt, bis die Spannungen hoch genug sind, um ein Umklappen der Zwillinge in Belastungsrichtung zu bewirken. Ab dieser Spannung tritt keine Verfestigung mehr auf. Das Spannungs-Dehnungs-Diagramm zeigt ein Plateau, wie in Abbildung 2.1 zu erkennen. Ist der gesamte Kristall vollständig entzwillingt, verformt er sich bei weiterer Erhöhung der Spannung erneut elastisch bis der plastische Bereich erreicht wird. Die weitere Verformung erfolgt plastisch durch Versetzungsbildung und deren Bewegung. Dieses Verhalten ist in Abbildung 2.2 zu erkennen, welche einen schematischen Spannungs-Dehnungs-Verlauf zeigt. Bei Entlastung aus dem zwillingsarmen Plateaubereich bleibt zunächst die Verformung bestehen. Findet nun eine Erwärmung über die Austenitisierungstemperatur statt, findet eine Phasenumwandlung der tetragonal verzerrten Martensitstruktur in die kubische Austenitphase statt. Dies bewirkt eine Rückkehr in die Ausgangsform. Beim erneuten Abkühlen in den Bereich der Niedertemperaturphase, bleibt die makroskopische Form erhalten. Daher auch der Name Einwegeffekt. Das beschriebene Verhalten wird in Abbildung 2.2 veranschaulicht.

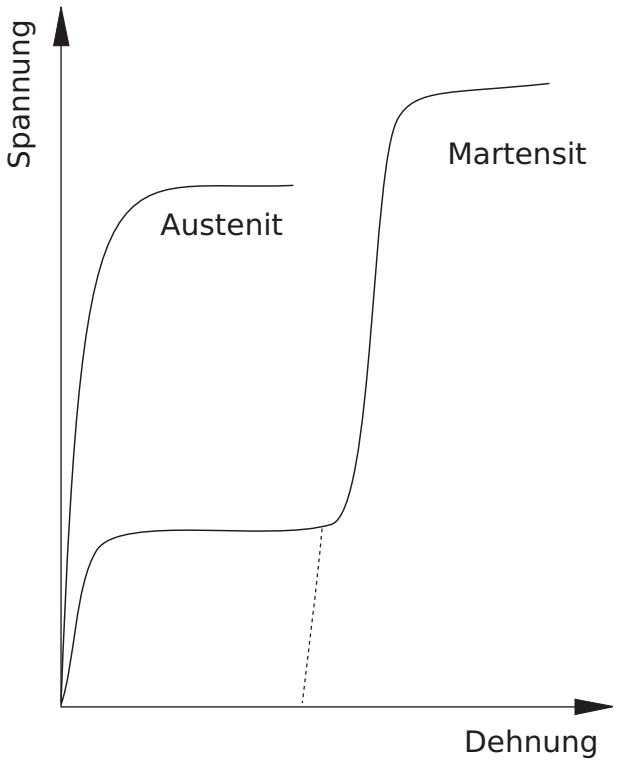


Abbildung 2.1: Schematisches Spannungs-Dehnungs-Diagramm von Austenit und Martensit.

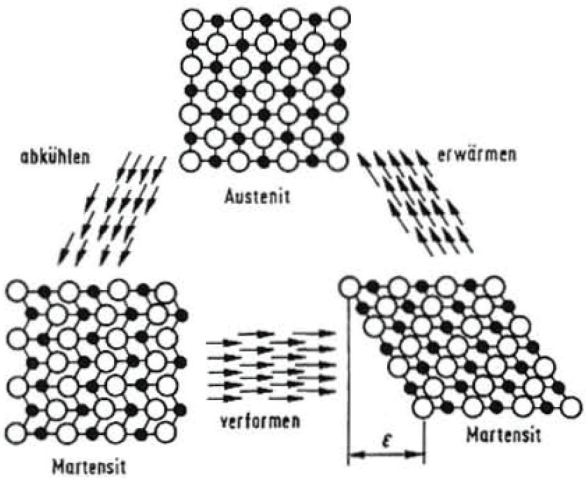


Abbildung 2.2: Darstellung des Verhaltens eines Formgedächtniskristalls unter Temperaturänderung und mechanischer Verformung [9].

2.1.3 Zweiwegeffekt

Im Gegensatz zum Einwegeffekt, bei dem nur bei Abkühlung unter die Austenitisierungstemperatur eine Formänderung auftritt, tritt diese beim Zweiwegeffekt auch beim Aufheizen auf [8]. Dies wird erreicht, indem man den Kristall zyklisch über den vorhin erwähnten Plateaubereich hinaus belastet. Es stellt sich eine bestimmte Versetzungsstruktur ein. Der reversible Anteil der Verformung wird beim Erwärmen wieder rückgängig gemacht. Das vorhandene Spannungsfeld der Versetzungen bewirkt beim Abkühlen eine Bevorzugung bestimmter Martensitvarianten. Dadurch tritt eine makroskopische Verformung sowohl beim Erwärmen als auch beim Abkühlen ein. Es werden jedoch wesentlich geringere Dehnungen erreicht als beim Einwegeffekt.

2.1.4 Superelastizität

Der Effekt der Superelastizität tritt in einem Temperaturbereich auf in dem die Hochtemperaturphase stabil ist, in dem jedoch auch spannungsinduzierte Martensitbildung noch möglich ist. Bringt man nun Spannung auf, wird Martensit gebildet der jedoch bei Entlastung nicht stabil ist. Dadurch können sehr hohe Dehnungen erreicht werden.

2.1.5 Das System Nickel-Titan

Die bis heute am meisten eingesetzte Formgedächtnislegierung ist das System Nickel-Titan, auch Nitinol genannt. Das Phasendiagramm des binären Systems

ist in Abbildung 2.3 zu erkennen. Die für den Formgedächtniseffekt interessante Legierung ist jene mit stöchiometrischer Zusammensetzung aus 50% Nickel und 50% Titan. Hier tritt die Hochtemperaturphase B2 in einem engen Konzentrationsbereich auf. Die B2 Phase ist kubisch und weist eine Cäsium-Chlorid-Struktur auf, bei der sich eine Spezies an den Kanten des Kristalls, die andere in der Mitte befindet. Kühlt man Nitinol aus der B2 Hochtemperaturphase ab, findet ein Phasenübergang in die monokline B19' Phase statt, die die Tieftemperaturphase darstellt. Wie bereits beschrieben, ist das Auftreten der martensitischen B19' Phase ausschlaggebend für die Eigenschaften als Formgedächtnislegierung. Die meisten anderen Systeme in denen die B2 Phase stabil ist, wandeln beim Abkühlen in eine orthorombische B19 Phase um. Diese Umwandlung zeigt jedoch keinen Formgedächtniseffekt.

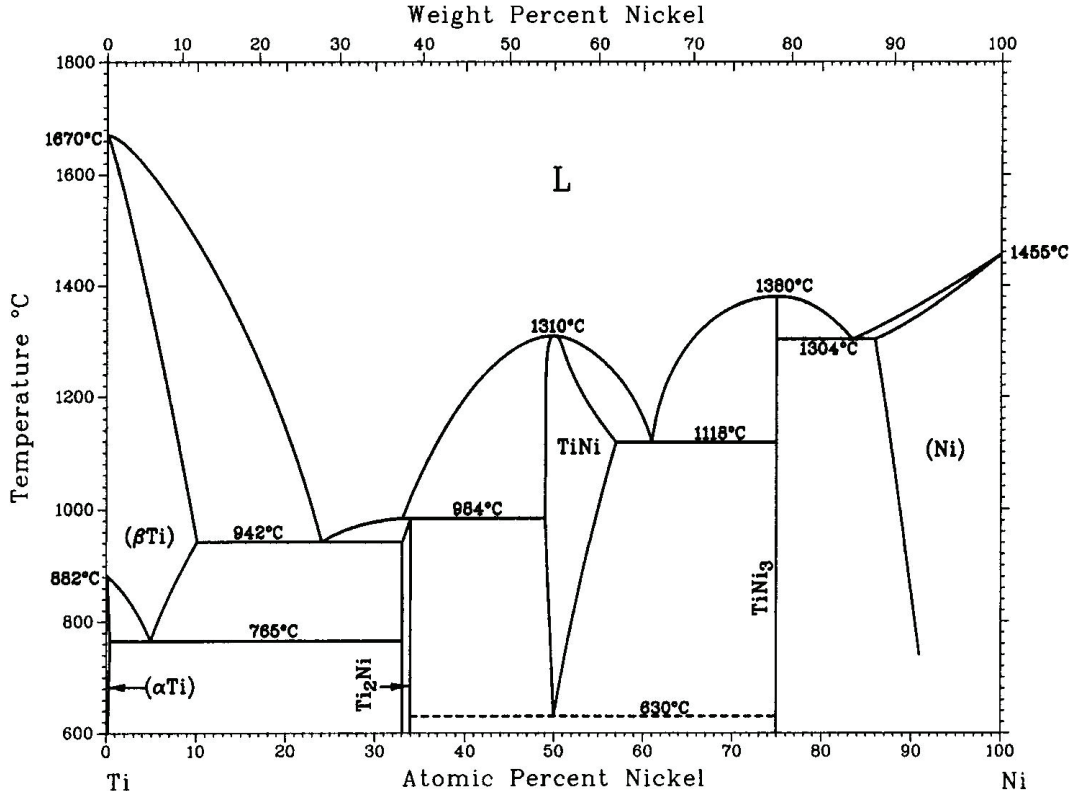


Abbildung 2.3: Phasendiagramm von NiTi laut Nash [10].

Um die Umwandlungstemperatur zu erhöhen, kann das binäre System Ni-Ti noch mit Hafnium legiert werden. [2, 11].

2.2 Dichtefunktionaltheorie

Mit der Einführung der Quantenmechanik und im speziellen der Schrödingergleichung, ist es nun prinzipiell möglich, die Eigenschaften eines Festkörpers ab-initio, also nur mithilfe von Naturkonstanten, zu berechnen. Die Hauptschwierigkeit dabei liegt in der Lösung des Vielteilchensystems. Im Allgemeinen kann ein System aus mehr als drei wechselwirkenden Teilchen nicht mehr analytisch gelöst werden. Möchte man jedoch einen Festkörper quantenmechanisch behandeln, muss die Schrödingergleichung für ein System aus sehr vielen wechselwirkenden Teilchen gelöst werden. Dies ist auch unter Verwendung numerischer Methoden nicht mehr exakt durchführbar. Es müssen alternative Verfahren eingeführt werden. Ein weit verbreitetes Verfahren ist die Dichtefunktionaltheorie, kurz DFT [12]. Eine erste Vereinfachung des Systems ist durch Entkopplung der Elektronen von den Atomkernen, der sogenannten Born-Oppenheimer Näherung [13], möglich. Diese ist plausibel, da die Elektronen viel weniger Masse und damit Trägheit aufweisen als die Atomkerne. Im Folgenden wird demnach die Wellenfunktion des Gesamtsystems als ein Produkt der Wellenfunktionen von Atomkernen und Elektronen laut $\Psi = \Psi_n \times \Psi_e$ angesehen. Durch die vergleichsweise große Masse der Atomkerne wird deren de Broglie Wellenlänge relativ groß sein. Ihre Wechselwirkung mit den Elektronen kann in klassischer Näherung durch ein externes Potential $V_{ext}(r)$ erfasst werden.

Die grundlegende Gleichung der Dichtefunktionaltheorie ist die Kohn-Sham-Gleichung [14], welche sich aus dem Theorem von Hohenberg und Kohn [15] ableitet. Dieses besagt, dass die Gesamtenergie eines Systems aus wechselwirkenden Elektronen in einem externen Potential ein Funktional der Elektronendichte ist:

$$E = F[\rho(r)] + \int V_{ext}(r)\rho(r)\mathrm{d}r. \tag{2.1}$$

Diese nimmt laut Hohenberg und Kohn [15] am Grundzustand ihr Minimum an. Es gilt die Kohn-Sham-Gleichung für ein fiktives System aus nichtwechselwirkenden Elektronen, welche die gleiche Elektronendichte aufweist wie das reale, wechselwirkende Elektronengas:

$$\hat{H}\Psi_i(r) = \epsilon_i\Psi_i(r). \tag{2.2}$$

Dabei bezeichnet Ψ_i allgemein den i-ten Eigenzustand mit dem Energieeigenwert ϵ_i und $\hat{H}$ stellt den Hamiltonoperator dar, der wie folgt geschrieben werden kann:

$$\hat{H} = -\bigtriangledown^2 + V_{ext}(r) + V_C(\rho(r)) + V_{xc}(\rho(r)). \tag{2.3}$$

Die kinetische Energie der Elektronen wird dabei durch den Operator $-\bigtriangledown^2$ beschrieben. Das durch die Atomrümpfe i erzeugte externe Potential lautet:

$$V_{ext}(r) = -\sum_i \frac{Z_i}{|r - R_i|}. \tag{2.4}$$

Die Coulomb-Wechselwirkung der Elektronen wird durch

$$V_C(\rho(r)) = \int \frac{\rho(r')}{|r - r'|} \mathrm{d}r', \tag{2.5}$$

beschrieben. Das Austausch Wechselwirkungsfunktional

$$V_{xc}(\rho(r)) = \frac{\delta E_{xc}(\rho(r))}{\delta \rho(r)} \tag{2.6}$$

stellt die einzige unbekannte Größe dar. Dieses kann nur durch Näherungen beschrieben werden. Die gängigsten Näherungen für Festkörper stellen die sogenannte Lokale-Dichte-Approximation (LDA) und Generalisierte-Gradienten-Approximation (GGA) [16] dar.

Dabei sind Hartree-Operator und Austauschkorrelationsoperator von der Grundzustandsdichte abhängig. Daher erfolgt die Berechnung selbstkonsistent. Es wird eine genäherte Grundzustandsdichte angenommen und daraus der Hamiltonoperator gebildet. Durch Lösen der Kohn-Sham-Gleichung erhält man die Eigenfunktionen $\Psi_i(r)$ der Elektronen. Daraus kann mit

$$\rho(r) = \sum_i |\Psi_i(r)|^2, \tag{2.7}$$

eine neue Grundzustandsdichte errechnet werden. Die neue Grundzustandsdichte wird mit jener aus dem vorigen Iterationsschritt gemischt, um besseres Konvergenzverhalten zu erreichen und in den nächsten Iterationsschritt überführt. Dies wird fortgeführt bis die Grundzustandsdichte stationär wird. Das zugrunde liegende Iterationsschema ist in Abbildung 2.4 dargestellt.

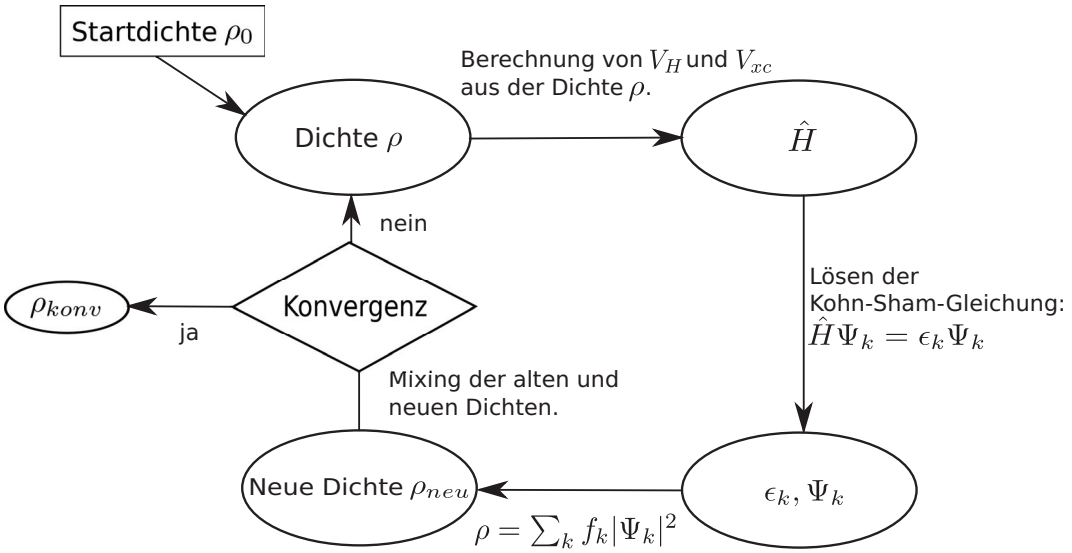


Abbildung 2.4: Schematischer Ablauf des Selbstkonsistenzzyklus der DFT-Rechnung: Es wird eine Startdichte ρ_0 angenommen und daraus über Hartree- und Austauschwechselwirkungspotential der Hamiltonoperator ermittelt. Durch Lösen der Kohn-Sham-Gleichung werden Eigenenergien und Wellenfunktionen berechnet, man erhält eine neue Dichte. Ist das Konvergenzkriterium nicht erfüllt, werden alte und neue Dichte gemischt (Mixing) und in den nächsten Iterationsschritt überführt.

2.3 Cluster-Entwicklung

Ein Problem bei der quantenmechanischen Behandlung von Legierungen ist die Berechnung eines Gitters, welches im allgemeinen keine Translationssymmetrie aufweist. Dabei müssen repräsentative Ausschnitte der Legierung im Modell abgebildet werden, die möglichst groß sein sollen, um dem statistischen Aspekt von Legierungen gerecht zu werden. Besteht die Berechnungszelle aus mehr als 1000 Atomen, wird es praktisch jedoch unmöglich dessen Grundzustands-Energie alleine mittels gängiger ab-initio Methoden zu bestimmen. Eine praktikable Lösung bietet die Einführung der sogenannten Cluster-Entwicklung [17], deren Grundprinzip im folgenden dargestellt werden soll.

Ziel ist es eine stetige Funktion der Bildungsenergie eines Kristallsystems für den Grundzustand in Abhängigkeit von der Konfiguration der Atome im Gitter zu erhalten. Die Darstellung erfolgt durch eine Reihenentwicklung in Cluster-Funktionen $\Phi_\alpha(\sigma)$ und zugehörige Koeffizienten E_α, die sogenannten ECI (effective cluster interactions). Wie in [17] gezeigt, bilden die Clusterfunktionen eine orthogonale vollständige 2^N-dimensionale Basis. Es kann gezeigt werden, dass Cluster welche mehr als fünf Atome beinhalten in den meisten Systemen vernachlässigt werden können [18]. Damit reduziert sich die Anzahl der einzubeziehenden Therme für die meisten Legierungssysteme auf eine Größenordnung von zehn Clustern.

Jedem Gitterplatz wird eine Spin-Variable zugeordnet, welche dem Gitterplatz eine Atomart zuordnet. Im Beispiel einer binären Legierung wird der Spin σ meist mit $\sigma_j \in \{-1, +1\}$ definiert (vergleiche Ising-Modell [19]). Die folgenden Betrachtungen beziehen sich auf binäre Legierungen, sind jedoch auch für den Fall von mehr als zwei Spezies gültig.

Die Entwicklung der Bildungsenergie nach Cluster-Funktionen lautet:

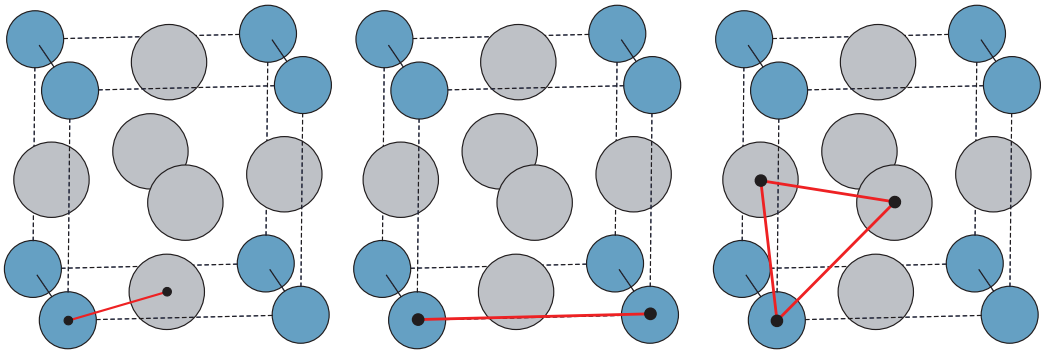
$$\Delta E(\vec{\sigma}) = \sum_\alpha E_\alpha \Phi_\alpha(\vec{\sigma}) \tag{2.8}$$

Dabei gibt α den Index des betrachteten Clusters an. Abbildung 2.5 zeigt eine mögliche Zuordnung. Der Index $\alpha = 1$ beschreibt hier Paare nächster Nachbarn im Gitter. Cluster mit dem Index $\alpha = 2$ enthalten zweit-nächste Nachbarn. Höherindizierte Cluster können Triplets nächster Nachbarn enthalten und so weiter.

Die Cluster-Funktionen sind das Produkt der Spin-Variablen über den gesamten betrachteten Cluster mit

$$\Phi = \prod_j \sigma_j \qquad \sigma_j \in \{-1, +1\}\,. \tag{2.9}$$

Im Beispiel von Abbildung 2.5 würden sich folgende Cluster-Funktionen ergeben:



(a) $\alpha = 1$ **(b)** $\alpha = 2$ **(c)** $\alpha = 3$

Abbildung 2.5: Mögliche Indizierung α von Clustern im fcc Gitter.

$$\Phi_1 = (+1)(-1) = -1$$

$$\Phi_2 = (+1)(+1) = 1$$

$$\Phi_3 = (+1)(-1)(-1) = 1$$

Weiters kann eine Vereinfachung durch Symmetrieüberlegungen geschehen. Daraus ergibt sich mit der Multiplizität m_α der symmetrieäquivalenten Cluster für die Energie:

$$\Delta E(\vec{\sigma}) = \sum_\alpha m_\alpha E_\alpha \langle \Phi_\alpha(\vec{\sigma}) \rangle. \tag{2.10}$$

Hier wird über alle Cluster Funktionen des Kristallsystems gemittelt. Im Falle nächster Nachbarn der $L1_2$ Struktur aus Abbildung 2.5 b geschieht dies wie folgt:

$\langle \Phi_2 \rangle = 1/48 \times \{12 \times (-1)(+1) + 3 \times [4 \times (-1)(+1) + 8 \times (+1)(+1)]$

Die gemittelten Cluster Funktionen und deren Multiplizität sind in folgender Tabelle angeführt:

α	m	$\langle \Phi_\alpha \rangle$
1	1	1
2	1	1/2
3	6	0

Daraus folgt für die Energie laut Gleichung 2.10:

$$\Delta E(\vec{\sigma}) = E_0 + \frac{1}{2} E_1. \tag{2.11}$$

Die Bestimmung der Grundzustände mit der niedrigsten Energie erfolgt durch Konstruktion der konvexen Hülle, welche die niedrigsten Werte für die Grundzustandsenergie verbindet. Durch Fitten der Datenpunkte der ab-initio Rechnungen

mit einem Polynom geeigneten Grades können die ECI's, also die Koeffizienten der Reihenentwicklung, bestimmt werden. Man erhält so die Grundzustandsenergie als Funktion der Konfiguration.

2.4 Thermodynamik

Im folgenden Kapitel werden die für diese Arbeit relevanten thermodynamischen Größen definiert [20].

2.4.1 Zustandsgrößen

Zustandsgrößen werden nur durch die momentanen Systemvariablen beschrieben und sind nicht vom Verlauf eines thermodynamischen Prozesses abhängig.

Innere Energie

Die Innere Energie U stellt, im Gegensatz zur Wärme Q und Arbeit W, ein totales Differential dar und ist somit auch eine Zustandsfunktion, da sie nur von den momentanen Systemeigenschaften abhängt. Die Änderung der Inneren Energie ist wie folgt definiert:

$$dU = \delta W + \delta Q. \tag{2.12}$$

Diese kann auch als Gesamtenergie, also als Summe der kinetischen und potentiellen Energie des betrachteten Systems, aufgefasst werden. Das totale Differenzial der inneren Energie kann geschrieben werden als:

$$dU = \left(\frac{\partial U}{\partial T}\right)_{V,\{n\}} + \left(\frac{\partial U}{\partial V}\right)_{T,\{n\}} + \sum_j \left(\frac{\partial U}{\partial n_j}\right)_{V,\{n\}\setminus j}. \tag{2.13}$$

Gibbs'sche freie Energie und chemisches Potential

Die Gibbs'sche freie Energie oder auch Freie Enthalpie stellt ebenfalls eine Zustandsfunktion dar:

$$G = H - TS. \tag{2.14}$$

Die Freie Enthalpie stellt die an einem System geleistete reversible Arbeit unter isobaren und isothermen Bedingungen dar. Das totale Differential dG der freien Enthalpie lautet:

$$dG = \left(\frac{\partial G}{\partial T}\right)_{V,\{n\}} + \left(\frac{\partial G}{\partial V}\right)_{T,\{n\}} + \sum_j \left(\frac{\partial G}{\partial n_j}\right)_{V,\{n\}\setminus j}. \tag{2.15}$$

Durch Vergleich der Differentialquotienten mit der differentiellen Form der Freien Enthalpie folgt:

$$dG = -SdT + Vdp + \sum \mu_i dn_i. \tag{2.16}$$

Dabei stellt μ_i das chemische Potential der i-ten Komponente dar:

$$\mu = \left(\frac{\partial G}{\partial n_i}\right)_{V,\{n\}} \tag{2.17}$$

Freie Energie

Als für die weiteren Betrachtungen wichtige Zustandsfunktion wird nun die Helmholtz'sche freie Energie A eingeführt:

$$A = U - TS. \tag{2.18}$$

Die Bedingung für das Gleichgewicht eines abgeschlossenen Systems ist demnach $dA_{V,T} = 0$, für eine thermodynamisch gesehen spontan ablaufende Reaktion $dA_{V,T} < 0$.

Thermodynamische Definition der Entropie

Die von Clausius [21] eingeführte Definition der Entropie, ursprünglich auch als reduzierte Wärme bezeichnet, lautet:

$$\mathrm{d}S = \frac{\mathrm{d}Q_{rev}}{T}. \tag{2.19}$$

Dabei bezeichnet $\mathrm{d}Q_{rev}$ die reversible Änderung der Wärmemenge. Sie ist gleichbedeutend mit einer Energiedissipation durch thermische Bewegung. Die thermodynamische Entropie liefert also eine Aussage über die Irreversibilität eines Prozesses. Damit gilt für die Entropieänderung des abgeschlossenen Gesamtsystems $\mathrm{d}S \geq 0$. Wobei die Entropieänderung nur bei adiabatischem Verhalten des Gesamtsystems (reversibler Prozess) zu null wird. Eine physikalisch intuitive Interpretation ist erst durch Einführung der statistischen Thermodynamik möglich. Auf diese wird später noch genauer eingegangen.

2.4.2 Thermodynamik von Phasenumwandlungen

Zwei Phasen α und β befinden sich im thermodynamischen Gleichgewicht, wenn sie dasselbe chemische Potential μ aufweisen:

$$\mu_\alpha = \mu_\beta \tag{2.20}$$

Die Darstellung der Freien Enthalpie über die Konzentration zweier Phasen in einem Zweistoffsystem zeigt Abbildung 2.6. Sind die chemischen Potentiale und somit die Ableitung der Freien Enthalpie nach der Konzentration für beide Phasen gleich, herrscht laut Gleichung 2.19 thermodynamisches Gleichgewicht. Die Schnittpunkte der gemeinsamen Tangente ergeben die jeweilige Gleichgewichtskonzentration für eine bestimmte Temperatur. Aus diesen Daten kann, wie in Abbildung 2.6 veranschaulicht, das Phasendiagramm konstruiert werden.

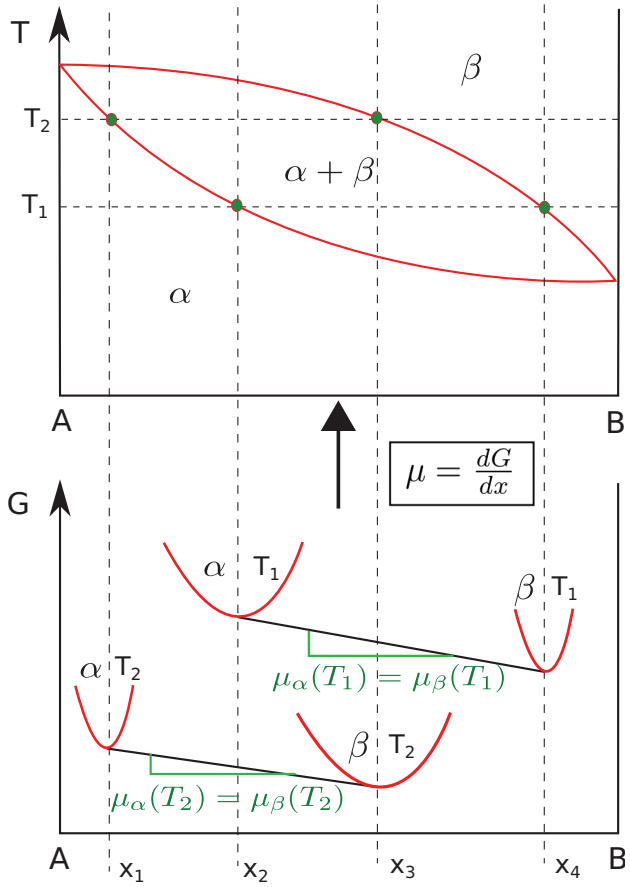


Abbildung 2.6: Schematische Darstellung eines binären Phasendiagramms (Komponenten A und B) mit den Phasen α und β und die zugehörigen G-x Kurven für die Temperaturen T_1 und T_2.

2.5 Grundlagen der statistischen Thermodynamik

Um die makroskopischen Eigenschaften wie Temperatur oder mittlere Energie eines Systems mit vielen Teilchen zu beschreiben, ist es nicht zweckmäßig die Bewegungsgleichungen im einzelnen zu lösen. Abgesehen davon, dass im allgemeinen die genaue Position und der Impuls eines jeden Teilchens nicht bekannt sind, würde man ein System aus gekoppelten Differentialgleichungen zu lösen haben, was die Berechnungskapazität heute verfügbarer Großrechner bei weitem übersteigt. Einen Ausweg bietet die statistische Physik und das Gesetz der großen Zahlen [22], die die Basis für die folgenden Betrachtungen darstellen.

2.5.1 Vom Mikrozustand zu Ensemble Mittelwerten

Ein Mikrozustand beschreibt ein System aus N Teilchen exakt. Ort r_i und Impuls p_i eines jeden Teilchens, beziehungsweise deren Energie e_i, sind im klassischen Grenzfall bekannt. Die Gesamtenergie des Systems ist die Summe über alle Energieniveaus multipliziert mit der jeweiligen Besetzungszahl,

$$E = \sum_{N} e_i n_i. \tag{2.21}$$

Betrachtet man nun hinreichend viele Mikrozustände eines Systems, kann die Besetzungsdichte der Energieniveaus durch eine Wahrscheinlichkeitsdichte ρ für die Besetzung der Energieniveaus ausgedrückt werden. Daraus ergibt sich ein Mittelwert für die Gesamtenergie des Systems, das sogenannte Ensemble-Mittel der Energie [20].

2.5.2 Boltzmann Verteilung

Um den wahrscheinlichsten Mikrozustand zu ermitteln wird die Verteilungsfunktion eingeführt. Das statistische Gewicht Ω einer Konfiguration von N Teilchen die in den Zuständen n_i vorliegen lautet:

$$\Omega = \frac{N!}{\prod_{i=0}^{r-1} n_i}. \tag{2.22}$$

Im folgenden wird diejenige Konfiguration beziehungsweise der Mikrozustand mit dem größten statistischen Gewicht gesucht. Ist das statistische Gewicht maximal mit $max\{\Omega\}$, bedeutet dies, dass der vorliegende Mikrozustand die größte Anzahl

an möglichen Realisierungen aufweist. Die Wahrscheinlichkeit für das Eintreten des Mikrozustandes wird maximal.

Dies führt laut [20] zur Boltzmann Verteilung:

$$\frac{n_i}{N} = \frac{e^{-\beta\epsilon_i}}{\sum_i e^{-\beta\epsilon_i}}. \tag{2.23}$$

.

2.5.3 Die kanonische Zustandssumme

Zentrale Bedeutung in der statistischen Thermodynamik hat die sogenannte Zustandssumme Z:

$$Z = \sum_i e^{-\beta\epsilon_i}. \tag{2.24}$$

Mit Hilfe der Zustandssumme ist es möglich, alle relevanten thermodynamischen Größen zu ermitteln. Vergleich mit Gleichung 2.17 zeigt, dass diese dem Ausdruck dem Nenner der Boltzmann Verteilung entspricht.

2.5.4 Ensembles

Ein Ensemble beschreibt eine Anzahl von Kopien eines Systems mit der selben Hamilton-Funktion bzw. dem selben Hamiltonoperator, welche unter den selben makroskopischen Bedingungen vorliegen. Je nach den makroskopischen Größen die das Ensemble beschreiben wird zwischen mikrokanonischen, kanonischen und großkanonischen Ensembles unterschieden [22].

Das mikrokanonische Ensemble

Das mikrokanonische Ensemble (Abbildung 2.7) wird durch konstante Innere Energie U, konstantes Volumen V und Teilchenzahl N beschrieben. Das System ist thermodynamisch abgeschlossen.

Das kanonische Ensemble

Im kanonischen Ensemble (Abbildung 2.8) wird die Teilchenzahl N des Systems, sowie das Systemvolumen V festgelegt. Das System tauscht Energie mit seiner Umgebung aus. Das System ist thermodynamisch geschlossen.

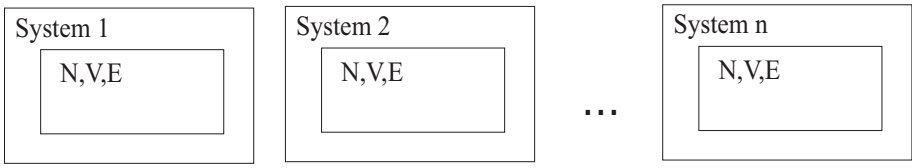


Abbildung 2.7: Mikrokanonisches Ensemble: Jedes System ist abgeschlossen, es findet kein Energietransfer mit der Umgebung statt.

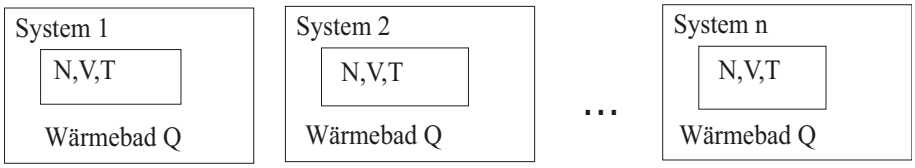


Abbildung 2.8: Kanonisches Ensemble: Jedes System befindet sich in demselben Wärmebad Q.

Das semi-großkanonische Ensemble

Die Gesamtanzahl der Teilchen ist konstant. Es findet lediglich eine fluktuation in der Konzentration statt.

Das großkanonische Ensemble

Hierbei handelt es sich um eine statistische Gesamtheit, deren Energie und Teilchenzahl fluktuiert. Das Volumen des Gesamtsystems ist hingegen konstant (Abbildung 2.9). Es handelt sich um ein thermodynamisch offenes System.

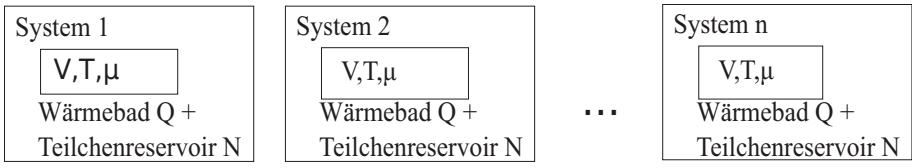


Abbildung 2.9: Großkanonisches Ensemble: Jedes System tauscht Teilchen und Wärme mit der Umgebung aus.

2.5.5 Ensemble Mittelwerte

Kennt man die Zustandssumme eines Systems, können daraus theoretisch alle relevanten thermodynamischen Größen ermittelt werden. Das 1. Postulat der statistischen Mechanik besagt, dass das Ensemblemittel dem Zeitmittel entspricht. Dies ist wesentlich, da nur das Ensemble Mittel für eine sinnvolle numerische Betrachtung in Frage kommt [22].

2.5.6 Thermodynamische Potentiale

Wie eingangs erwähnt, lassen sich aus der Zustandssumme Z alle thermodynamischen Potentiale ableiten. Einige wichtige Potentiale und deren Zusammenhang mit der Zustandssumme sind in folgender Tabelle angegeben:

Innere Energie:	$U = kTln\left(\frac{\partial Z}{\partial T}\right)_V$
Freie Energie:	$A = -kTln(Z)$
Freie Enthalpie:	$G = -kT\left[lnZ - \left(\frac{\partial Z}{\partial V}\right)_T\right]$

Großkanonisches Potential

Das großkanonische Potential Φ ist nach Landau durch die Zustandsvariablen Volumen V, Teilchenzahl N, chemisches Potential μ, und Temperatur T definiert:

$$\Phi = A - \mu N. \tag{2.25}$$

Befindet sich das System im Gleichgewicht, nimmt Φ einen Extremwert an.

2.5.7 Chemisches Potential

Die Ableitung der Freien Energie nach der Teilchenzahl wird als chemisches Potential bezeichnet:

$$\mu_i = \frac{\partial A}{\partial n_i}. \tag{2.26}$$

Anschaulich beschreibt das chemische Potential einer Phase den Drang, Teilchen einer bestimmten Spezies aufzunehmen oder abzugeben. Ist die Differenz des chemischen Potentials zweier Phasen $\Delta\mu \neq 0$ so befinden sie sich nicht im Gleichgewicht und es findet Energie- und Teilchentransfer statt bis $\Delta\mu = 0$ und sich die Phasen somit im Gleichgewicht befinden.

2.5.8 Entropie in der statistischen Betrachtungsweise

Wie in Kapitel 2.4 erläutert, trifft die thermodynamische Definition der Entropie Aussagen über die Irreversibilität von Prozessen. Mit der statistischen Betrachtung ist nun eine Interpretation und ein tieferes Verständnis des Begriffes der Entropie möglich. Bei der Vereinigung zweier nicht wechselwirkender Systeme ist

deren Entropie thermodynamisch gesehen additiv $S_{ges} = S_1 + S_2$. Das zugehörige statistische Gewicht hingegen stellt sich als Produkt der beiden Systeme heraus $\Omega_{ges} = \Omega_1 \, \Omega_2$. Dieser Zusammenhang wird durch die Gleichung

$$S = k \, ln(\Omega) \tag{2.27}$$

beschrieben, wobei der Faktor k wie in [20] dargestellt die Boltzmann-Konstante darstellt. Dieser Zusammenhang ermöglicht eine Interpretation der Entropie als ein Maß für die Unordnung eines Systems.

2.6 Monte Carlo Simulation

Um einen statistischen Mittelwert einer thermodynamischen Größe zu erhalten, muss ein Integral der Form

$$\langle A \rangle = \frac{\int dpdr \; A(p,r) \; e^{\beta U(p,r)}}{Z(p,r)} \tag{2.28}$$

im Phasenraum gelöst werden. Dabei wird mit A(p,r) die zu bestimmende thermodynamische Größe bezeichnet, die von den Parametern r und p abhängt, U(p,r) beschreibt ein Potential und Z(p,r) wird als Zustandssumme bezeichnet (siehe Kapitel 2.5):

$$Z(p,r) = \int dpdr \; e^{\beta U(p,r)}. \tag{2.29}$$

Theoretisch wäre es möglich den Mittelwert mittels numerischer Integration zu bestimmen. Besteht das System jedoch aus vielen Teilchen, wird die Dimension des Phasenraumes ((3+3)N) zu groß um sie in der Praxis noch mit numerischen Standardverfahren zu bewältigen. Weiters werden sehr viele Integrationspunkte benötigt, um nicht-glatte Funktionen zufriedenstellend abzubilden.

Ein häufig angewandtes Verfahren zur Lösung von Integralen in hochdimensionalen Räumen, ist das im Folgenden vorgestellte Monte Carlo Verfahren.

2.6.1 Erzeugung von Zufallszahlen

Basis jeder Monte Carlo Simulation bildet die Erzeugung von Zufallszahlen. Da ein Computer deterministischer Natur ist, ist es nicht möglich, Zufallszahlen im natürlichen Sinne zu erzeugen. Daher muss man für praktische Zwecke mit sogenannten pseudo-Zufallszahlen das Auslangen finden.

Ein weit verbreiteter Algorithmus zur Erzeugung von Pseudo-Zufallszahlen ist der Multiplicative Linear Congruential Generator [23, 24] der eine Sequenz von Zufallszahlen errechnet die sich nach einer Periodenlänge wiederholt. Die Periodenlänge und Korrelation sind ausschlaggebend für die Qualität eines Zufallsgenerators.

Verteilungsfunktionen

Mit diesen Generatoren ist es möglich qualitativ ausreichende Zufallszahlen zu generieren, die allerdings primär gleichverteilt im Bereich zwischen Null und Eins auftreten. Für viele physikalische Modelle ist es jedoch wesentlich Zufallszahlen zu generieren, die einer bestimmten Verteilung folgen.

Daraus ergibt sich die Notwendigkeit Methoden einzuführen, die aus einer gleichverteilten Folge aus Zufallszahlen eine Folge zu generieren, die einer bestimmten Verteilung genügt.

2.6.2 Metropolis Algorithmus

Um eine Boltzmann-verteilte Folge an Zufallszahlen zu generieren, wird im folgenden ein Algorithmus nach Metropolis [25] angewendet:

1.) Generieren einer zufallsverteilten Startkonfiguration.

2.) Neue Konfiguration suchen.

3.) Entscheidung ob neuer Zustand (n) übernommen wird oder alter Zustand (o) beibehalten wird. Dies geschieht mit der Übergangswahrscheinlichkeit $\Pi(o \to n)$.

Ist die Verteilung einmal erreicht, darf sie nicht wieder zerstört werden. Dies wird durch die Überprüfung der detailed-balance [22] gewährleistet:

$$N(o)\ \Pi(o \to n) = N(n)\ \Pi(n \to o). \tag{2.30}$$

Dies bedeutet nichts anderes, als dass die Anzahl der Übergänge von alten Zustand mit der Besetzungszahl $N(o)$ zum neuen mit der Besetzungszahl $N(n)$ gleich sein muss der Anzahl der Übergänge von allen anderen Zuständen in den alten Zustand.

Im allgemeinen Fall ist die Übergangswahrscheinlichkeit Π richtungsabhängig und lautet:

$$\Pi(o \to n) = \alpha(o \to n)\ acc(o \to n). \tag{2.31}$$

Der Faktor $\alpha(o \to n)$ beschreibt die Richtungsabhängigkeit und $acc(o \to n)$ gibt die Wahrscheinlichkeit an, mit der der neue Zustand akzeptiert wird.

Nach der Bedingung für detailed-balance gilt dann:

$$N(o)\ \alpha(o \to n)\ acc(o \to n) = N(n)\ \alpha(n \to o)\ acc(n \to o). \tag{2.32}$$

Setzen wir die Übergangswahrscheinlichkeit als symmetrisch voraus, d.h. $\alpha(o \to n) = \alpha(o \to n)$, vereinfacht sich die Gleichung:

$$N(o)\ acc(o \to n) = N(n)\ acc(n \to o) \tag{2.33}$$

$$\frac{acc(o \to n)}{acc(n \to o)} = \frac{N(n)}{N(o)} = e^{-\beta(U(n)-U(o))}. \tag{2.34}$$

Somit haben wir die Zustandssumme aus der Gleichung eliminiert und können ohne größeren Aufwand die Gleichung lösen.

Beim Metropolis Algorithmus wird folgende Bedingung für die Übergangswahrscheinlichkeit, von Konfiguration i zu Konfiguration i+1 in der Markov Kette, festgelegt:

$$acc(o \to n) = \begin{cases} \frac{N(n)}{N(o)} & \text{if } N(n) < N(o) \\ 1 & \text{if } N(n) > N(o) \end{cases} . \tag{2.35}$$

2.6.3 Anwendung auf Cluster Entwicklung

Speziell auf die Cluster Entwicklung angewandt bedeutet dies für den Algorithmus:

1. Wahl einer Startkonfiguration von Spinzuständen $(\sigma)_i$ für $i = 1...N$.

2. Zufällige Wahl eines Spinzustandes $\sigma_j \in (\sigma)_i$ und Umkehrung des Vorzeichens $\sigma_j = -\sigma_j \Rightarrow \sigma_{new}$, wie in Abbildung 2.10 veranschaulicht.

3. Berechnen der Differenz des Thermodynamischen Potentials:

$$\Delta\Phi = E(\sigma_{old}) - E(\sigma_{new}) - \mu\sigma_j. \tag{2.36}$$

4. Berechnung der Übergangswahrscheinlichkeit $\Pi(\sigma_i \to -\sigma_j)$:

$$\Pi(\sigma_j \to -\sigma_j) = \begin{cases} \frac{N(n)}{N(o)} = e^{-\beta U} & \text{if } \Delta\Phi > 0 \\ 1 & \text{if } \Delta\Phi < 0 \end{cases} \tag{2.37}$$

5. Ziehen einer Zufallszahl Z_i im Intervall (0,1) und Vergleich mit $\Pi(\sigma_i \to -\sigma_j)$:
   Wenn $\Pi(\sigma_i \to -\sigma_j) > Z_i$ wird der Spinzustand geändert (Spin-Flip), ansonsten behält man den Ursprungszustand bei.

2.6.4 Erreichen des Gleichgewichts

Auch bei sehr großer Anzahl der Schritte in einer Markov Kette kann der thermodynamische Gleichgewichtszustand nicht erreicht werden, weil dazu eine unendliche Zahl an Schritten notwendig wäre. Um die Simulationszeit nach oben hin zu begrenzen, werden Grenzen für die Genauigkeit des Verfahrens gesetzt [26].

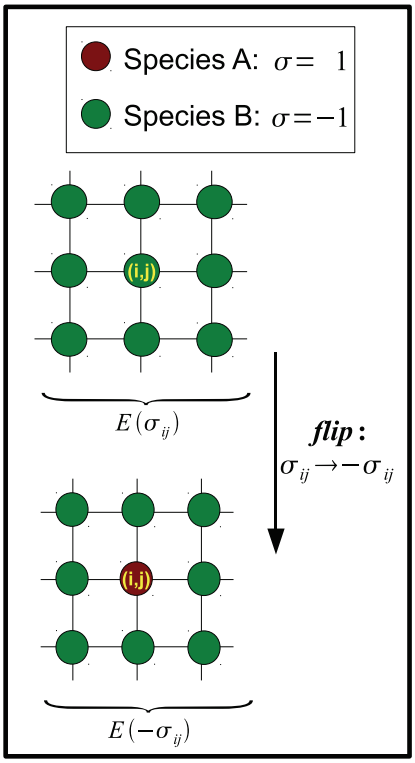


Abbildung 2.10: Darstellung eines Spin-Flip-Event: Austausch von Spezies B durch Spezies A am betrachteten Gitterplatz i, j. Es findet eine Änderung der Gitterenergie von $E(\sigma_{ij})$ zu $E(-\sigma_{ij})$ statt.

Zeit bis Mittelwert stationär wird

Ein Maß für die Abweichung vom Mittelwert $\langle Q \rangle$ der Observablen Q ist die Varianz $Var(\langle Q \rangle)$ definiert als

$$Var(\langle Q \rangle) = \langle Q^2 \rangle - \langle Q \rangle^2 = \frac{1}{L^2} \sum_{t}^{L} \sum_{s}^{L} Cov(Q_t, Q_s) \tag{2.38}$$

Die Summation läuft über alle L Schritte der Markov-Kette und $Cov(Q_t, Q_s)$ beschreibt die Kovarianz zwischen Q_t und Q_s. Die Ermittlung der Varianz auf diesem Wege ist sehr rechenintensiv mit einer Anzahl an Rechenoperationen von der Ordnung $O(L^2)$. Daher werden die im folgenden beschriebenen Vereinfachungen durchgeführt.

Die Markov Kette der Monte Carlo Simulation kann man laut [26] als eine Linearkombination von Komponenten mit exponentiell abfallender Autokorrelationsfunktion betrachten. Dabei wird diejenige Komponente mit der größten Relaxationszeit τ ausschlaggebend sein und die restlichen Therme können vernachlässigt

werden. Daraus ergibt sich, dass die Kovarianz V zu einem Zeitpunkt l geschrieben werden kann als:

$$V_l = V_0 \rho^{-|l|}. \tag{2.39}$$

Der Erwartungswert der Kovarianz zum Zeitpunkt l lautet:

$$\langle Cov(Q_l) \rangle = \frac{1}{L-l} \sum_t Q_t Q_{t+l} - \langle Q \rangle^2. \tag{2.40}$$

Somit hat man eine exponentiell mit l abfallende Kovarianz mit

$$\rho = e^{(-\tau^{-1})}, \tag{2.41}$$

wobei τ die Relaxationszeit angibt. Durch Einsetzen in Gleichung 2.37 und Auswertung der geometrischen Summe, erhält man schließlich für die Varianz:

$$Var(\langle Q \rangle) = \frac{1}{L} \sum_{t=1}^{L} \sum_{l=-\infty}^{\infty} Var(Q_0) \rho^{-|l|} = \frac{Var(Q_0)}{L} \left(\frac{1+\rho}{1-\rho} \right) \tag{2.42}$$

Zeit zum Erreichen des Gleichgewichts

Um zu beurteilen ob ein thermodynamisches Gleichgewicht mit hinreichender Genauigkeit erreicht wird, betrachtet man den Mittelwert $\langle Q \rangle(\Delta t)$ der zeitabhängigen Messgröße $Q(t)$ in zwei aufeinander folgenden Zeitabschnitten $\Delta t_{12} = t_2 - t_1$ $\Delta t_{23} = t_3 - t_2$. Ist nun die Differenz der Mittelwerte statistisch betrachtet nicht signifikant verschieden von Null, kann das Erreichen des Gleichgewichts ab t_1 angenommen werden [26]. Die Genauigkeit p steigt mit vergrößern der Intervalle ΔT. Ist nun die Differenz der Erwartungswerte für die betrachteten Zeitabschnitte kleiner als die gewünschte Genauigkeit p

$$|\langle Q \rangle(t_1, (t_1+t_2)/2) - \langle Q \rangle((t_1+t_2)/2, t_1)| \leq p, \tag{2.43}$$

wird sich die betrachtete Größe Q nicht mehr wesentlich ändern. Es kann näherungsweise von einem Gleichgewicht ausgegangen werden.

Werden für die Analyse der Konvergenz der Energiemittelwerte immer alle N bisher berechneten Werte verwendet, ist dies jedoch relativ rechenintensiv mit $O(N^2)$. Daher werden in der Praxis nur Werte, die eine definierte Zeitspanne zurückliegen, in die Mittelwertbildung einbezogen. Diese Blockmittelwerte benötigen nicht viel Speicher und beinhalten genügend Information um die Konvergenz zu beurteilen.

2.7 Thermodynamische Integration

2.7.1 Methodik

Um das thermodynamische Potential aus Monte-Carlo Rechnungen zu erhalten wird eine thermodynamische Integration ausgeführt [26]. Das thermodynamische Potential im semi-großkanonischen System kann als

$$\phi = A - \mu x \tag{2.44}$$

geschrieben werden (siehe Kapitel 2.5). Das totale Differential des thermodynamischen Potentials lautet:

$$d(\beta\phi) = (E - \mu x)d\beta - \beta x d\mu. \tag{2.45}$$

Durch Integration erhält man das semi-großkanonische Potential:

$$\beta_1\phi(\beta_1, \mu_1) = \beta_0\phi(\beta_0, \mu_0) \int_{\beta_0}^{\beta_1} (\langle E\rangle - \mu\langle x\rangle)d\beta - \int_{\mu_0}^{\mu_1} \beta\langle x\rangle d\mu. \tag{2.46}$$

Dieses Integral kann sehr einfach numerisch gelöst werden, da die mittlere Energie $\langle E\rangle$ und die mittlere Konzentration $\langle x\rangle$ je Atom aus den Monte Carlo Rechnungen bekannt sind.

2.7.2 Näherungen und Anfangsbedingungen

Um Anfangsbedingungen für die thermodynamische Integration zu erhalten und die Anzahl der erforderlichen Monte Carlo Schritte auf ein Minimum zu beschränken, führen wir semi-analytische Näherungen ein. Diese werden im jeweils gültigen Bereich angewandt, welcher beispielsweise durch Vergleich mit einer Mean-Field-Näherung verifiziert wird.

High Temperature Expansion

Ein Beispiel für eine analytische Näherung für $\beta \to 0$ ist die sogenannte HTE (High Temperature Expansion):

$$\beta\phi(\beta, \mu) = -ln(2) \qquad \beta \to 0 \tag{2.47}$$

Dabei wird angenommen, dass die Konzentration im Limes hoher Temperatur gegen 1/2 strebt.

Low Temperature Expansion

Ausgehend von einem bestimmten Grundzustand kann der Anfangswert des thermodynamischen Potentials durch die LTE (Low Temperature Expansion) ausgedrückt werden mit

$$\phi(\beta,\mu) = E_g - \mu x_g - \frac{1}{N\beta}\sum_s e^{-\beta(\Delta\epsilon_{s,g}-\mu\Delta\eta_{s,g})} \tag{2.48}$$

wobei E_g die Grundzustandsenergie je Atom und x_g die Konzentration im Grundzustand darstellen. Der Term $\Delta\epsilon_{s,g}$ bezeichnet die Energie für einen Austausch der Spezies am Gitterplatz s im Grundzustand g, $\Delta\eta_{s,g}$ die Änderung der Konzentration. Die Summe kann wegen der Translationsinvarianz des Gitters auf die Einheitszelle begrenzt werden. Die Rechtfertigung für diesen Ansatz liegt im Auftreten von einfachen Spin-Flips bei tiefen Temperaturen. Wird die Temperatur über einen kritischen Wert erhöht treten auch mehrere Spin-Flips simultan auf und die Näherung verliert ihre Gültigkeit.

Kapitel 3

TiAl: Ein einführendes Beispiel

Zur Einarbeitung in ATAT und um den eigenen Monte Carlo Code zu testen, wurden Rechnungen am System Ti-Al durchgeführt. Im folgenden wird die praktische Ausführung Schritt für Schritt von der Berechnung der Grundzustände über die Cluster Expansion bis hin zur Monte Carlo Simulation zur Berechnung der Phasenfläche für einen ausgewählten Grundzustand gezeigt. In Abbildung 3.1 ist ein Schema des Programmflusses zu sehen.

3.1 Grundzustandsrechnungen mit EXCITING

Um den ATAT Code für die Berechnung der Grundzustände mittels DFT zu konfigurieren sind einige Vorarbeiten notwendig. Speziell für den EXCITING code ist dies in [27] erläutert. Die Parameter für den DFT Code werden in exciting.wrap definiert:

```
1  kppra = 1728 # no. of k-points per reciprocal atom ...
       1728=12^3
2
3  ___EXCITINGINPUT-BEGIN___
4  <?xml version="1.0" encoding="UTF-8"?>
5  <?xml-stylesheet href="http://xml.exciting-code.org/
       inputfileconverter/inputtohtml.xsl" type="text/xsl"?>
6  <input xsi:noNamespaceSchemaLocation="http://static.
       exciting-code.org/hydrogen/excitinginput.xsd"
7   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
8     >
9     <title>TiAl</title>
10    <structure speciespath="~/exciting/species/">
11 ___CRYSTAL_AND_SPECIES___
```

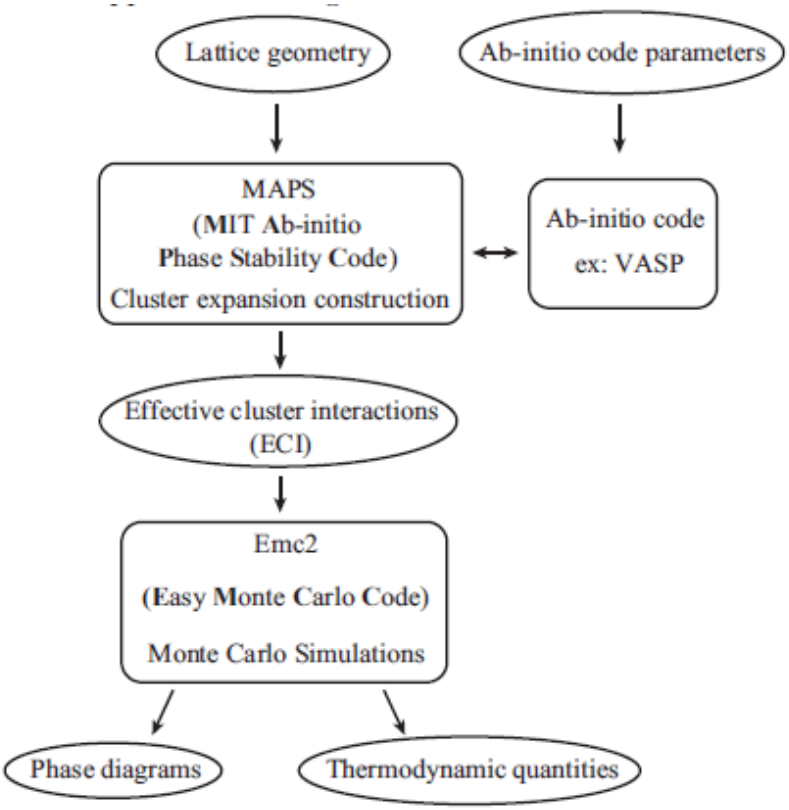


Abbildung 3.1: Schema des Programmflusses des ATAT Programmpakets [3].

```
12    </structure>
13    <groundstate ngridk="5 5 5"
14    rgkmax="6.0"
15    nktot="125" mixer="msec"
16    xctype="GGAPerdew-Burke-Ernzerhof"
17    nwrite="1"
18    dlinengyfermi="-0.1"
19    fermilinengy="true"
20    findlinentype="advanced"
21    ></groundstate>
22 </input>
23
24 ___EXCITINGINPUT-END___
```

Für eine ausführliche Beschreibung aller Parameter siehe [28].

3.2 Cluster Entwicklung mit ATAT: MAPS

Ist die Schnittstelle zum ab-initio Code eingerichtet, sind lediglich zwei Input-Files notwendig um die CE auszuführen. Die Gittergeometrie wird durch lat.in definiert:

```
4.040387   0.000000   0.000000
0.000000   4.040387   0.000000
0.000000   0.000000   4.040387
0.000000   0.500000   0.500000
0.500000   0.000000   0.500000
0.500000   0.500000   0.000000
0.000000   0.000000   0.000000  Al,Ti
```

Die ersten drei Zeilen definieren Basisvektoren der primitiven Einheitszelle. In diesem Fall entsprechen sie einem kubischen Gitter mit einem Gitterparameter in Einheiten des Bohr'schen Radius von 4.040387. Danach werden die primitiven Gittervektoren angegeben, in diesem Fall jene eines kubisch flächenzentriert (fcc) Gitters. Die folgenden Zeilen stehtn für Positionen der Atome in der Basis und die variable Besetzung der Gitterplätze, in diesem Fall mit Aluminium (Al) beziehungsweise Titan (Ti). Sind sowohl die Geometrie des Gitters, als auch die Eingabeparameter für den DFT-Code vorhanden, kann maps mit den Default Parametern aufgerufen werden:

```
maps -d &
```

Folgender Befehl startet den automatisierten Prozess zur Auswahl von Strukturen und deren ab-initio Berechnung:

```
pollmach runstruct_exciting &
```

Um maps zu beenden wird die Datei `stoppoll` durch,

```
touch stoppoll
```

erstellt. Mit dem Befehl `mapsrep` kann unter anderem eine Graphische Darstellung der Cluster-Entwicklung aufgerufen werden. Diese ist für vorliegendes System in Abbildung 3.2 zu sehen. Es sind nun alle nötigen Eingabedateien für die anschließende Monte Carlo Simulation vorhanden.

3.3 Monte Carlo Simulation mit ATAT: emc2

Um die Monte Carlo Simulation auszuführen werden die Parameter in einem XML File mit dem Namen `mcinput.xml` definiert:

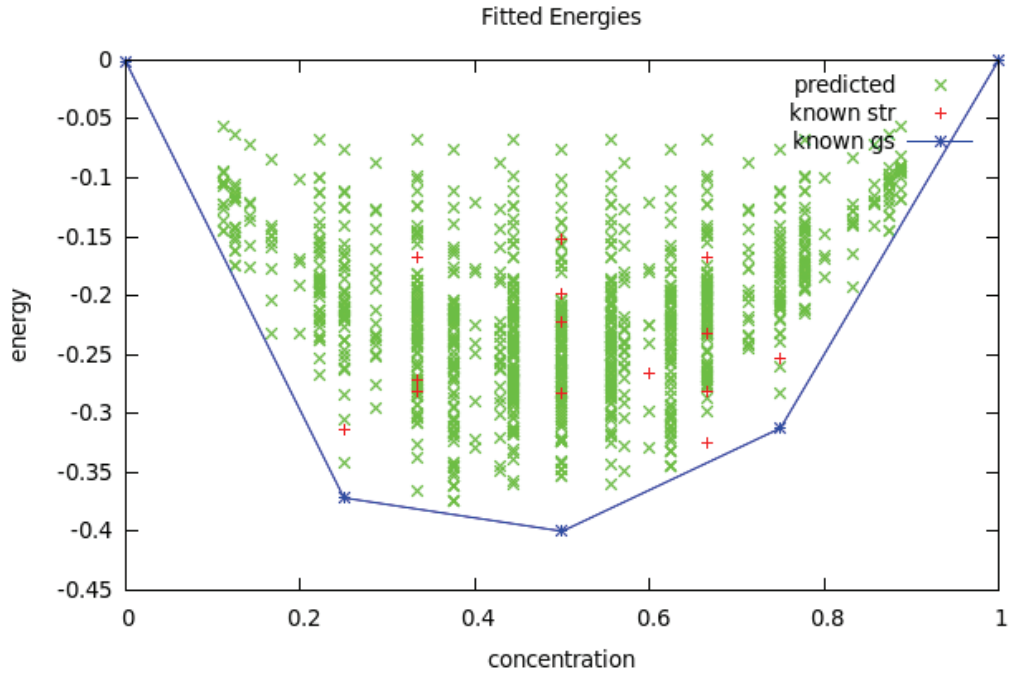


Abbildung 3.2: Cluster Entwicklung des Beispielsystems TiAl: Die Einhüllende definiert zugleich die errechneten Grundzustände.

```
<?xml version="1.0" encoding="UTF-8"?>
<experiment path='~/TiAl/'>
        <fixedpar o='gs_dT.out' T0='273' T1='593' dT='50'
            mu0='2.5' mu1='1.5' dmu='0.04' k='8.617e-5' dx=
            '1e-3' gs='2' er='64'/>
        <set mu0='2.5' mu1='1.5' path='gs20'/>
        <set mu0='1.5' mu1='0.5' path='gs21'/>
</experiment>
```

Es werden Berechnungen für den Grundzustand 2 im Temperaturbereich von 273K bis 593K ausgeführt. Dabei wird ausgehend vom chemischen Potential des Grundzustandes dieses simultan in positiver und negativer Richtung variiert. Durch Ausführen des Skripts `calc.py`

```
python calc.py mcinput.xml
```

wird eine hierarchische Dateistruktur erzeugt in der folgende Einzelrechnungen ausgeführt werden:

```
emc2 -gs=2 -mu0=2.5 -mu1=1.5 -dmu=0.04 -T0=300 -T1=1000 -
    dT=50 -k=8.617e-5 -dx=1e-3 -er=10 -o=gs20.out
```

```
emc2 -gs=2 -mu0=2.5 -mu1=1.5 -dmu=0.04 -T0=300 -T1=1000 -
    dT=50 -k=8.617e-5 -dx=1e-3 -er=10 -o=gs21.out
```

Ist die Monte Carlo Simulation abgeschlossen, kann die Auswertung durch

```
python read.py
```

gestartet werden. Es wird eine XML Datei `data.xml` erstellt, welche die wesentlichen Daten der Simulation beinhaltet. Um eine graphische Darstellung der Phasenfläche zu erhalten, kann das Skript `plot.py` ausgeführt werden:

```
python plot_3d.py data.xml
```

Die Darstellung erfolgt in einem interaktivem Fenster, wie in Abbildung 3.3 zu sehen. In diesem Beispiel die Phasenfläche für den Grundzustand 2.

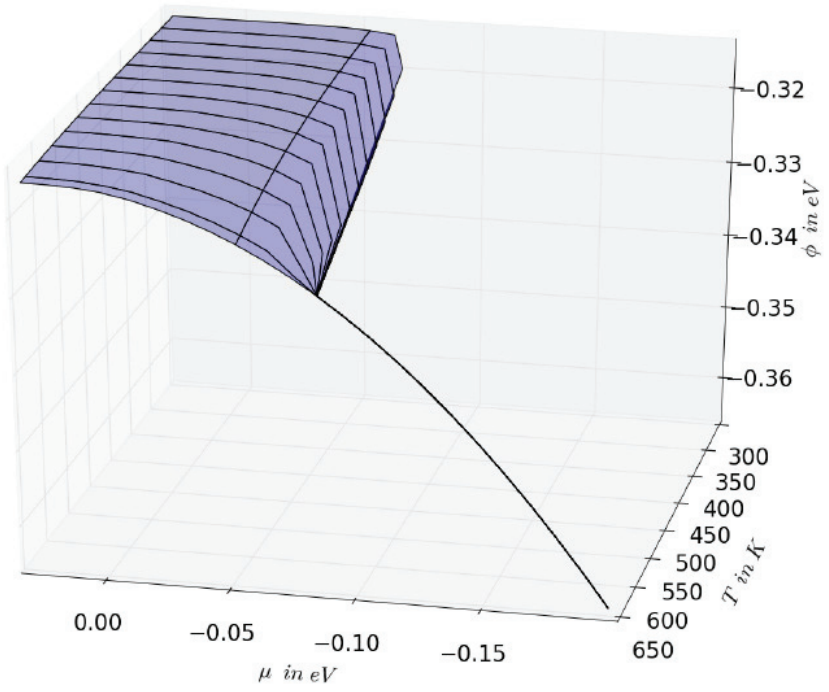


Abbildung 3.3: Phasenfläche für Grundzustand 2 von TiAl.

Es ist zu bemerken, dass die Ergebnisse dieses Beispiels keine hohe Qualität bezüglich der errechneten freien Energie aufweisen. Dies ist hauptsächlich auf die geringe Zellengröße der Monte Carlo Simulation zurückzuführen.

Kapitel 4

Ergebnisse

4.1 Ausgangsdaten

Um thermodynamische Daten aus ab-initio Rechnungen zu erhalten, wird zunächst eine Cluster-Entwicklung durchgeführt um die Abhängigkeit der Grundzustandsenergie am absoluten Temperaturnullpunkt von der Konfiguration der Atome im Gitter zu erhalten (siehe Kapitel 2.3). Die Koeffizienten der Cluster-Entwicklung wurden bereits im Vorfeld dieser Arbeit von Dr. Jürgen Spitaler ermittelt [4]. Aufbauend auf die Cluster-Entwicklung wird im weiteren die Vorgehensweise beschrieben die schlussendlich zu Daten über bestimmte Phasenübergänge des betrachteten Systems Ni-Ti-Hf führen soll.

4.2 Vorgehensweise

Ausgehend von den verschiedenen Grundzuständen, wird jeweils eine Monte Carlo Simulation durchgeführt. Die im Vorfeld ermittelte Cluster-Entwicklung, sowie die Geometrie des Gitters, dienen als Eingabeparameter für den Monte Carlo Code emc2 [3] des Programmpakets ATAT [6]. Dieser ermittelt die Phasenfläche der Freien Energie über Temperatur T und Chemischem Potential μ. Weiters können Phasenübergänge detektiert werden.

4.3 emc2 - Monte Carlo Code

Der Code emc2 ist ein auf der Programmiersprache C basierender Code, welcher eine Monte Carlo Simulation im semi-großkanonischem Ensemble implementiert. Beim semi-großkanonischen Ensemble bleibt die Gesamtanzahl der Teilchen im System konstant, wobei bei gegebenem chemischen Potential die Konzenration

der Spezies im System fluktuiert, wie in Kapitel 2.5 erläutert. Der Grund für die Einführung des semi-großkanonischen Ensembles liegt darin, dass für ein beliebiges chemisches Potential lediglich Einphasen-Gleichgewichte auftreten was die numerische Stabilität positiv beeinflusst.

4.3.1 Input

Alle Eingabedaten werden in den Einheiten für Energie in eV und Temperatur in K angegeben. Diese werden, wie im folgenden beschrieben, dem Code als ASCII Files zur Verfügung gestellt.

1. Gittergeometrie File `lat.in`

```
4.040387   0.000000   0.000000
0.000000   4.040387   0.000000
0.000000   0.000000   4.040387
0.000000   0.500000   0.500000
0.500000   0.000000   0.500000
0.500000   0.500000   0.000000
0.000000   0.000000   0.000000  Al, Ti
```

2. Cluster-Entwicklung: Diese wird durch die Files `clusters.out` und `eci.out` beschrieben, welche bei Anwendung von `maps` automatisch generiert werden. Sie können aber grundsätzlich von jeder beliebigen Cluster Entwicklung stammen. Die ECI's der Cluster-Entwicklung werden hier mit aufsteigender Ordnung der zugehörigen Cluster angegeben. Im vorliegenden Fall sind dies Cluster eins bis sieben (siehe Kapitel 2.3).

```
-0.249416
 0.029914
 0.032573
-0.023921
 0.008674
-0.003671
 0.010472
```

```
0.000000   0.000000  -0.001254   0
0.250000  -0.400759  -0.371763  27
0.500000  -0.396533  -0.400078   3
0.750000  -0.293720  -0.312484  28
1.000000   0.000000  -0.000156   1
```

3. Grundzustände als Startkonfigurationen für den Markov-Prozess werden in `gs_str.out` festgelegt.

4.3.2 Kontrollparameter

Um die Startwerte für die thermodynamische Integration festzulegen Das chemische Potential wird in relativen Werten bezogen auf das Gleichgewicht im gewählten Grundzustand wie folgt angegeben:

$$\mu = \begin{cases} 0 & \text{für Grundzustand 0} \\ 1 & \text{für Grundzustand 1} \\ \vdots & \\ n & \text{für Grundzustand n} \end{cases} \tag{4.1}$$

Wie in Abbildung 4.1 veranschaulicht, verläuft die thermodynamische Integration entlang der Integrationspfade bei konstanter Temperatur, ausgehend von den Startwerten T_{start} und μ_{start}. Wird ein Phasenübergang detektiert, wird die Temperatur um die Schrittweite ΔT geändert $T \to T + \Delta T$ und die die Integration beginnt erneut bei μ_{start}.

4.3.3 Output

4.4 Automatisierung

Zur einfacheren Analyse des Konvergenzverhaltens, sowie der simultanen Berechnung mehrerer Phasen, wurde ein Script zur Automatisierung der Monte Carlo Rechnungen in Python implementiert. Als input sowie output Format dient xml da dieses einfachen Zugriff über die Abfragesprache XPath bietet und ein übersichtliches, hierarchisches Konzept darstellt.

4.4.1 Input Format

```
<input mod="serial">
        <par T0="273.0" T1="573.0" dT= dmu0=0.004598 gs=1/
           >
        <var er=2/>
        <var er=4/>
        <var er=8/>
</input>
```

Das Input-File wird durch ein xsl Template in mehrere Datensätze von xml konvertiert. Diese werden dann sequenziell, lokal an der Workstation oder parallel auf einem Großrechner abgearbeitet. Die Namen der Output-Dateien `*.xml`, setzen sich aus den zu variierenden Parametern `<var/>` zusammen. Die konstant gehaltenen Werte `p1,p2...` werden mit `<par p1= p2= .../>` deklariert. Für die Variation der Zellengröße mit den hier gewählten Parametern würden also folgende Operationen ausgeführt:

```
emc2 -er=2 -gs=1 -dmu= -dT= -mu0= -mu1= -T0=273.0 -T1
    =573.0 -o=er2.out
emc2 -er=4 -gs=1 -dmu= -dT= -mu0= -mu1= -T0=273.0 -T1
    =573.0 -o=er4.out
emc2 -er=8 -gs=1 -dmu= -dT= -mu0= -mu1= -T0=273.0 -T1
    =573.0 -o=er8.out
```

Es wird eine hierarchische Dateistruktur angelegt in denen die Einzelrechnungen ausgeführt werden.

4.4.2 Output Format

Das Skript `read.py` liest die wichtigsten Daten zur weiteren Analyse aus den Output-Dateien von emc2. Und fasst diese als XML Datei zusammen `data.xml`.

```
<phase name="gs1_0.out">
<cells er="er8/">
    <data T="273.0" mu="0.004598" phi="-0.004821" x="
        0.983453"/>
    <data T="273.0" mu="0.00472" phi="-0.004941" x="
        0.983167"/>
    <data T="273.0" mu="0.004842" phi="-0.005061" x="
        0.982281"/>
    <data T="273.0" mu="0.004963" phi="-0.005181" x="
        0.983135"/>
    <data T="273.0" mu="0.005085" phi="-0.0053" x="
        0.983208"/>
     .
     .
     .
</cells>
<cells er="er16/">
        <data T="273.0" mu="-0.004537" phi="-0.00375" x="
            -0.995312"/>
    <data T="293.0" mu="-0.004537" phi="-0.003785" x="
        -0.992608"/>
```

```
15      <data T="313.0" mu="-0.004537" phi="-0.003835" x="
            -0.988983" />
16      <data T="333.0" mu="-0.004537" phi="-0.003903" x="
            -0.984129" />
17      <data T="353.0" mu="-0.004537" phi="-0.003994" x="
            -0.977508" />
18      .
19      .
20      .
21  </cells>
22  </phase>
```

Das Auslesen der Daten erfolgt mittels XPath Anweisungen.

Dies wurde in einem Python Skript implementiert um automatisch Graphen für Konvergenzverhalten, dreidimensionale Darstellung von Phasenflächen und G-x Kurven zu erhalten.

4.5 Konvergenztests

Die Betrachtung des Konvergenzverhaltens ist notwendig um eine hinreichende Genauigkeit der nachfolgenden Simulation sicherzustellen, wobei die Rechenzeit minimal gehalten werden soll. Untersucht wurde das Konvergenzverhalten bezüglich der Zellengröße der Monte Carlo Simulation sowie der Schrittweite in Temperatur und chemischem Potential der thermodynamischen Integration.

4.5.1 Zellengröße

Als repräsentatives System für die Untersuchung der Konvergenz wurde in diesem Fall Grundzustand 1 mit 25% Hf gewählt. Laut [26] ist davon auszugehen, dass die erforderliche Zellengröße nahezu unabhängig von anderen Kontrollparametern ist. Die Zellengröße wurde dabei logarithmisch in Schritten von 2^n im Bereich $n = 1 \ldots 7$ erhöht. Abbildung 4.2 zeigt das Konvergenzverhalten der Konzentration mit der Zellengröße. Daraus folgt eine notwendige Zellengröße von 50x50x50 Einheitszellen. Folgendes Input-File wurde für die Analyse der Konvergenz bezüglich Zellengröße verwendet:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <experiment path='/home/tde/cedata/runtime/mcrun2/'>
3          <fixedpar o='gs2.out' T0='273' T1='593' dT='20'
               mu0='2.5' mu1='1.5' dmu='0.04' k='8.617e-5' dx=
               '1e-3' gs='2'/>
```

```
4                   <set er = '2' path='er2'/>
5                   <set er = '4' path='er4'/>
6                   <set er = '8' path='er8'/>
7                   <set er = '16' path='er16'/>
8                   <set er = '32' path='er32'/>
9                   <set er = '64' path='er64'/>
10                  <set er = '128' path='er128'/>
11                  <set er = '256' path='er256'/>
12                  <set er = '512' path='er512'/>
13                  <set er = '1024' path='er1024'/>
14 </experiment>
```

Wie im Abbildung 4.3 zu erkennen, ist die Konvergenz auch bezüglich der Umwandlungstemperatur für eine Zellengröße von 50x50x50 gegeben.

4.5.2 Schrittweite des chemischen Potentials

Durch Variation der Schrittweite des chemischen Potentials wurde deren Einfluss auf das Konvergenzverhalten der Monte Carlo Simulation untersucht:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <experiment path='/home/tde/cedata/runtime/conv_dmu/'>
3         <fixedpar o='gs2_0.out' T0='273' T1='593' dT='20'
              mu0='2.5' mu1='1.5' k='8.617e-5' dx='1e-3' gs='
              2' er='64'/>
4                   <set dmu = '0.16' path='dmu32'/>
5                   <set dmu = '0.08' path='dmu64'/>
6                   <set dmu = '0.04' path='dmu2'/>
7                   <set dmu = '0.02' path='dmu4'/>
8                   <set dmu = '0.01' path='dmu8'/>
9 </experiment>
```

Wie sich zeigte ist ein Zellenradius der Superzelle von 32 Einheitszellen ausreichend.

4.5.3 Schrittweite der Temperatur

Die Analyse der Konvergenz bezüglich der Schrittweite der Temperatur, wie sie in der thermodynamischen Integration verwendet wird, erfolgte mit folgenden Eingabeparametern:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <experiment path='/home/tde/cedata/runtime/conv_dT/'>
```

```
3         <fixedpar o='gs_dT.out' T0='273' T1='593' mu0='2.5
              ' mu1='1.5' dmu='0.04' k='8.617e-5' dx='1e-3'
              gs='2' er='64'/>
4               <set dT = '2' path='dT2'/>
5               <set dT = '4' path='dT4'/>
6               <set dT = '8' path='dT8'/>
7               <set dT = '16' path='dT16'/>
8               <set dT = '32' path='dT32'/>
9               <set dT = '64' path='dT64'/>
10              </experiment>
```

Das Konvergenzverhalten ist in Abbildung 4.4 zu erkennen. Daraus folgt ein notwendiger Zellenradius der Superzelle von 16 Einheitszellen.

4.6 Detektion von Phasenübergängen

Phasenübergänge zeichnen sich durch eine Unstetigkeit einer thermodynamischen Funktion über den Integrationsbereich ab. Beispielsweise weist die Funktion der Freien Energie über das chemische Potential bei einem Phasenübergang erster Ordnung einen Knick auf. Dieser kann wie im Folgenden beschrieben, durch eine Extrapolation der Daten durch ein Polynom und Vergleich mit der Simulation detektiert werden. Dabei ist die Ordnung des Polynoms zu bestimmen. Die Vorhersagekraft der Extrapolation wird durch den sogenannten Cross Validation Score (CV) ausgedrückt [26, 29].

Das Polynom, welches die Funktion am besten beschreibt sei gegeben wenn

$$CV = \frac{1}{n}\sum_{i=1}^{n}(\langle Q_i\rangle - \acute{Q}_i)^2 \tag{4.2}$$

minimal wird. Es wird über alle Datenpunkte Q_i summiert, wobei $\acute{Q}_i$ einen least-squares Fit der Datenpunkte $\{Q_1, \ldots, Q_{i-1}, Q_{i+1}, \ldots, Q_n\}$ darstellt.

Ist das Polynom gefunden welches (Gleichung) minimiert, wird dieses zur Vorhersage von Q_{n+1} angewendet. Nun wird ein Kriterium gesucht mit dem man eine Phasentransformation von statistischen Fluktuationen der Datenpunkte unterscheiden kann. Es gilt also herauszufinden ob der Vorhersagefehler $Q_{n+1} - \acute{Q}_{n+1}$ statistisch gesehen signifikant von Null abweicht. Ist dies der Fall wird eine Phasentransformation detektiert. Mit der Substitution

$$X_{ij} = (C_i)^{j-1} \tag{4.3}$$

wobei C_i natürliche Variable der Funktion Q_i sind und

$$Y_i = \langle Q_i\rangle, \tag{4.4}$$

kann man den Erwartungswert für den Residuenvektor $\hat{a}$ in Matrixform schreiben als:

$$\hat{a} = (X^T X)^{-1} X^T Y \tag{4.5}$$

mit der Kovarianzmatrix

$$V = \sigma^2 (X^T X)^{-1}. \tag{4.6}$$

Die Varianz der Residuen σ^2 kann durch eine Mittelung der Varianzen der Datenpunkte abgeschätzt werden [30].

$$\sigma^2 = \frac{1}{n} \sum_i Var(Q_i). \tag{4.7}$$

Mit $x_j = (C_{n+1})^{j-1}$ ist die Vorhersage für den Datenpunkt bei $n+1$ gegeben als

$$Q_{n+1}^{'} = x^T \hat{a}, \tag{4.8}$$

und dessen Varianz mit

$$v = x^T V x. \tag{4.9}$$

Da nun v die Varianz der Vorhersage $Q_{n+1}^{'}$ und σ^2 die Varianz des wahren Wertes Q_{n+1} darstellt, ist eine statistisch signifikante Varianz der Werte größer oder gleich deren Summe:

$$|Q_{n+1} - Q_{n+1}^{'}|^2 \geq v + \sigma^2. \tag{4.10}$$

Somit ist ein Kriterium für das Auftreten einer Phasenumwandlung gefunden.

4.7 Phasenflächen

Nach Untersuchung der Konvergenz und Bestimmung der Parameter für Zellengröße, Schrittweite in Temperatur und chemisches Potential wurden Monte Carlo Simulationen zur Bestimmung der Phasenfläche über einen Temperaturbereich von 273 bis 573K durchgeführt. Die Phasenfläche lässt sich als Fläche im dreidimensionalen Raum Darstellung: das semi-großkanonische Potential wird dabei über Temperatur und chemisches Potential aufgetragen. Ausgehend vom Grundzustand wurden chemisches Potential und Temperatur solange verändert bis ein Phasenübergang detektiert wurde.

Als Parameter der Simulation wurden eine Zellengröße von 50x50x50 Einheitszellen und eine Schrittweite der Temperatur von 20K verwendet. Die Schrittweite im chemischen Potential betrug 1.22e-4 eV. Es wurden Simulationen für die Grundzustände 0,1,2 und 3 durchgeführt. Diese weisen folgende chemische Zusammensetzung auf:

Grundzustand	Anteil durch Hf substituierter Ti Plätze %	gemittelter Spin
0	0	-1
1	25	-0.5
2	33	-0.33
3	100	1

Mit folgendem Inputfile wurde die Phasenfläche ausgehend von den Grundzuständen für den Temperaturbereich von 273K bis 593K, mit einem Superzellen-Radius von 64 Einheitszellen berechnet.

```
<?xml version="1.0" encoding="UTF-8"?>
<experiment path=''>
        <fixedpar o='gs2_0.out' T0='273' T1='593' dT='20'
            mu0='2.5' mu1='1.5' dmu='0.04' k='8.617e-5' dx=
            '1e-3' gs='2' er='64'/>
                <set gs='0' mu0='0.5' mu1='-0.5' path='
                    gs0_0'/>
                <set gs='0' mu0='0.5' mu1='1.5' path='
                    gs0_1'/>
                <set gs='1' mu0='1.5' mu1='0.5' path='
                    gs1_0'/>
                <set gs='1' mu0='1.5' mu1='2.5' path='
                    gs1_1'/>
                <set gs='2' mu0='2.5' mu1='1.5' path='
                    gs2_0'/>
                <set gs='2' mu0='2.5' mu1='3.5' path='
                    gs2_1'/>
                <set gs='3' mu0='3.5' mu1='2.5' path='
                    gs3_0'/>
                <set gs='3' mu0='3.5' mu1='4.5' path='
                    gs3_1'/>
</experiment>
```

Phasenübergänge sind in der Phasenfläche als "Knick"(hebbare Unstetigkeit) zu beobachten. In Abbildung 4.5 ist dies für die Grundzustände eins und zwei zu erkennen.

Die gesamte Phasenfläche ausgehend von den Grundzustandskonfigurationen ist in Abbildung 4.6 gezeigt.

4.8 G-x Kurven

Zur anschaulichen Darstellung wurde ein Skript geschrieben, welches die automatisierte Darstellung und Auswertung von G-x Kurven ermöglicht. Folgender Zusammenhang besteht zwischen Freier Energie G und dem Großkanonischen Potential ϕ, μ und der Konzentration x:

$$G = \phi - \mu x. \tag{4.11}$$

Dabei stammen ϕ, μ und x aus der vorher durchgeführten Berechnung der Phasenflächen.

Für die Grundzustände eins und zwei, ist dies in Abbildung 4.7 dargestellt. Mit Hilfe der Tangentenkonstruktion können nun Konzentration und Temperatur der Phasengrenzen ermittelt werden.

Wie in Abbildung 4.6 und Abbildung 4.8 ersichtlich, sind außer den Grundzuständen noch weitere Phasenübergänge zu erwarten. Dies ist einerseits durch die freien Flächen in der Darstellung des thermodynamischen Potentials (siehe Abbildung 4.6) und andererseits durch eine Unvollständigkeit in der Tangentenkonstruktion ersichtlich. Durch Erhöhen des chemischen Potentials über den Phasenübergang hinaus konnten einige mögliche Kandidaten gefunden werden. Ein Problem besteht in der großen Streuung der Werte für manche detektierten Phasen. Dadurch wird eine Beurteilung schwierig da nicht genau zu erkennen ist, ob tatsächlich eine Phase vorliegt oder es sich nur um Artefakte handelt.

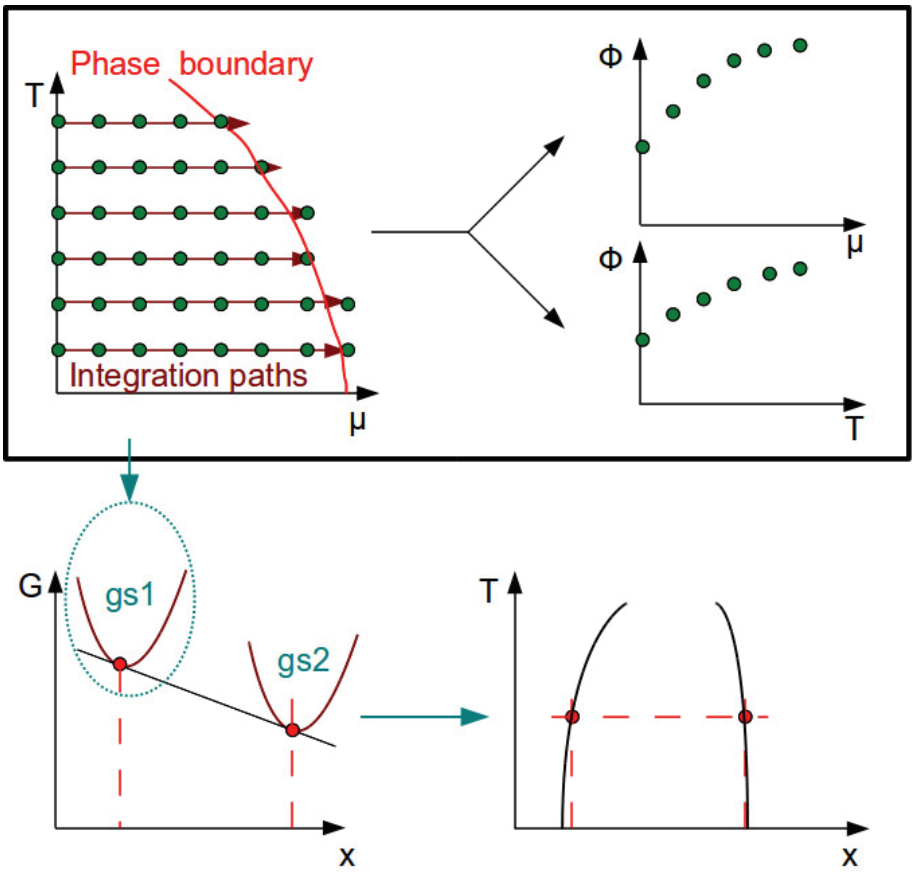


Abbildung 4.1: Schema der thermodynamischen Integration: Ausgehend von den Startwerten für Temperatur T und chemischem Potential μ, wird die Integration bei konstanter Temperatur durchgeführt bis ein Phasenübergang detektiert wird. Die Integration wird in diskreten Temperaturschritten, mit gegebenem Intervall, durchgeführt. Man erhält das thermodynamische Potential als Funktion von μ und T. Über die Tangenten-Konstruktion der Freien Enthalpie, kann ein Phasendiagramm erstellt werden.

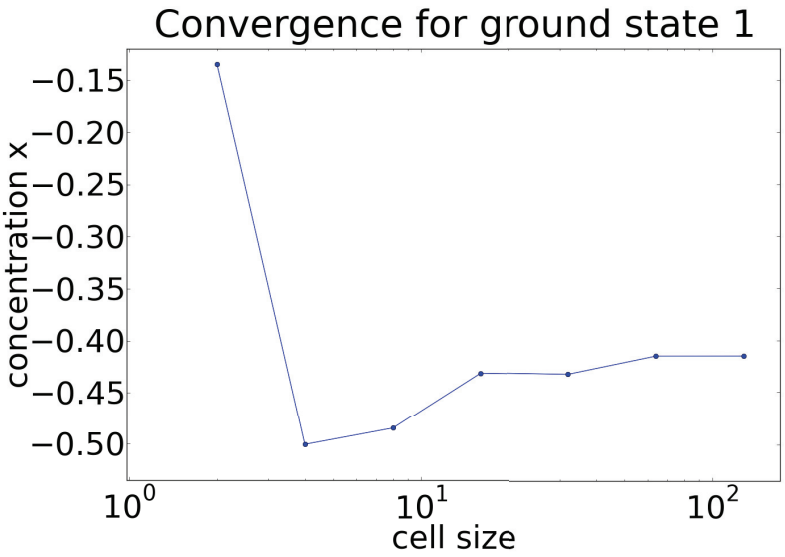


Abbildung 4.2: Änderung der mittleren Konzentration mit steigender Zellengröße für Grundzustand 1.

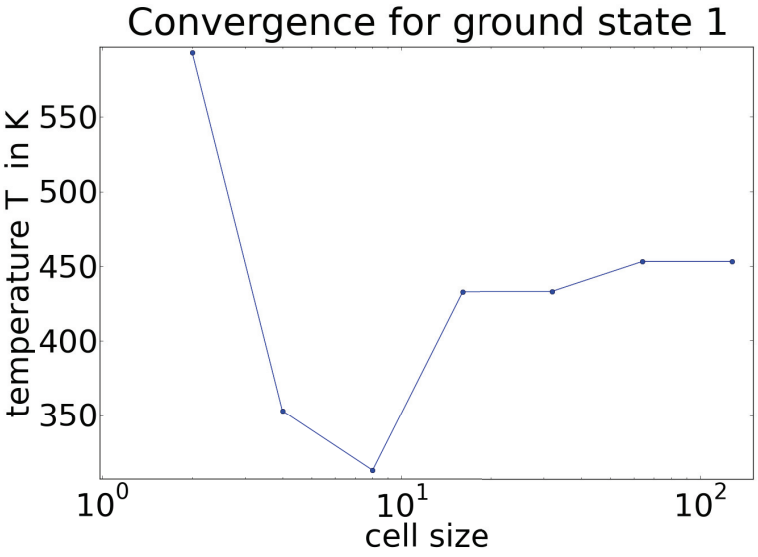


Abbildung 4.3: Änderung der Übergangstemperatur mit steigender Zellengröße für Grundzustand 1.

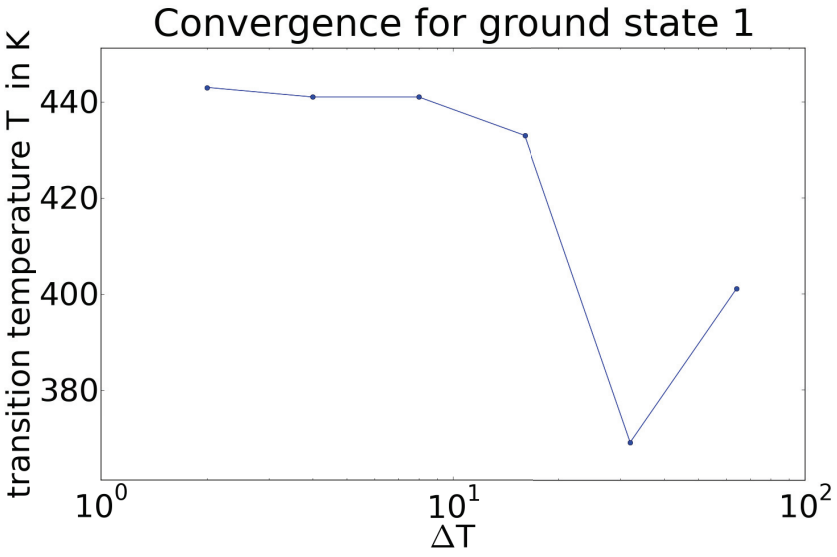


Abbildung 4.4: Konvergenzverhalten der Umwandlungstemperatur bezüglich der Schrittweite der Temperatur in der thermodynamischen Integration.

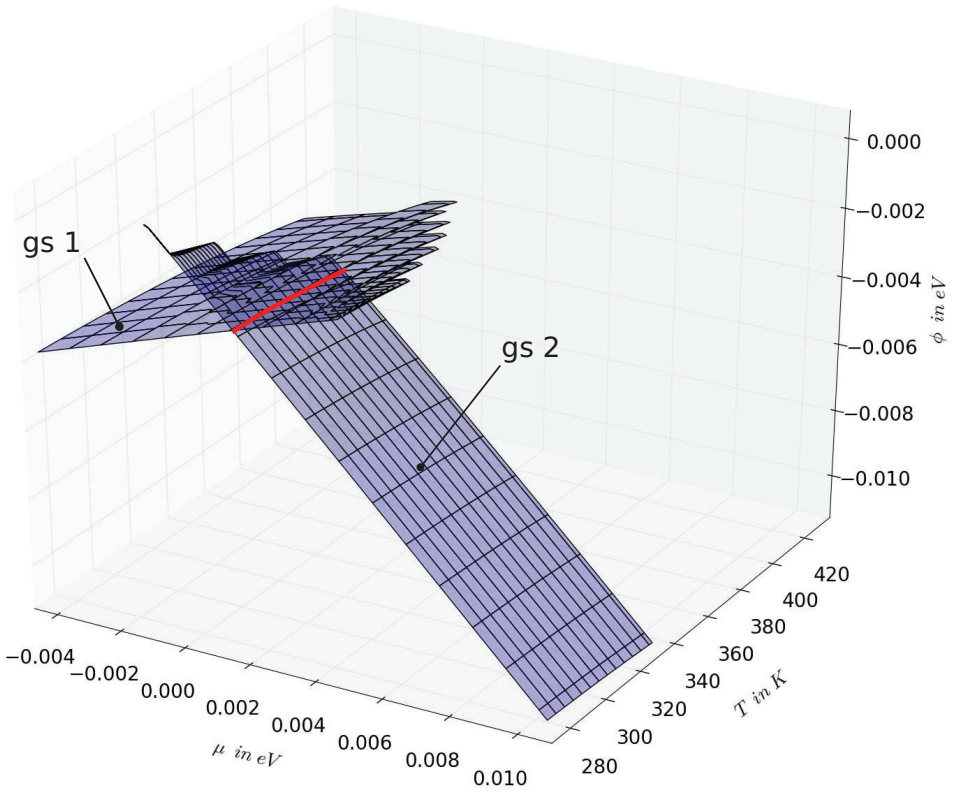


Abbildung 4.5: Phasenfläche der Grundzustände eins (gs 1) und zwei (gs 2). Die Schnittlinie der beiden Flächen, stellt einen Phasenübergang dar.

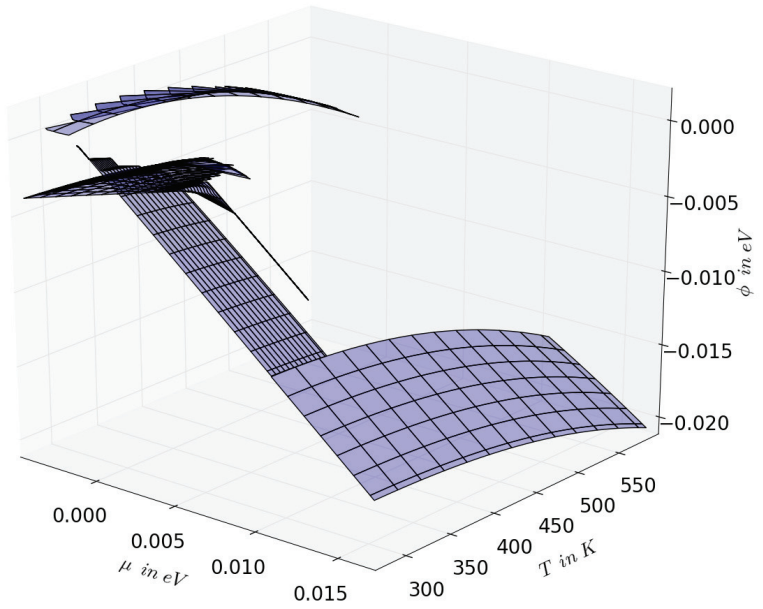


Abbildung 4.6: Gesamte Phasenfläche bei der Berechnung ausgehend von den einzelnen Grundzuständen.

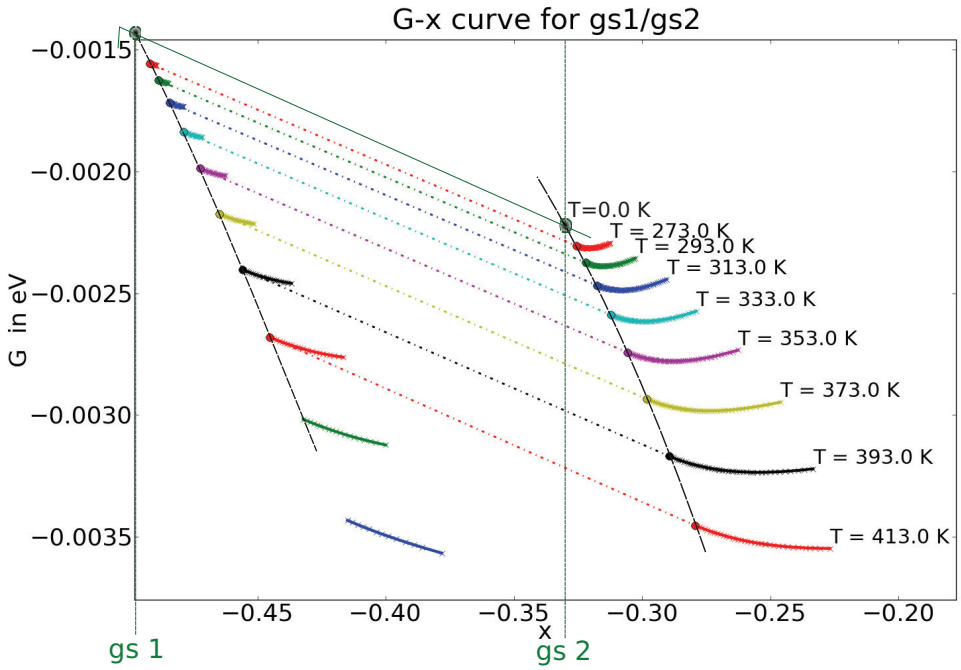


Abbildung 4.7: Freie Enthalpie über Konzentration (-1 entspricht 0% Hf, 1 entspricht 100% Hf) für die Grundzustände eins und zwei, bei verschiedenen Temperaturen.

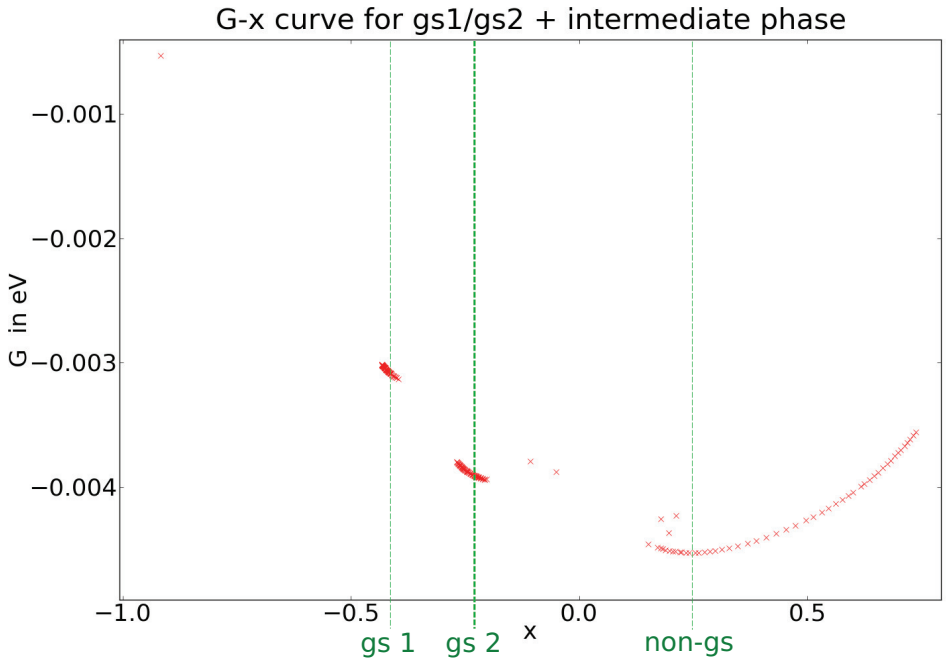


Abbildung 4.8: Freie Enthalpie über Konzentration (-1 entspricht 0% Hf, 1 entspricht 100% Hf) für die Grundzustände eins und zwei bei 293K. Es ist eine Phase zu erkennen, welche keinem der Grundzustände entspricht.

4.9 Monte Carlo Code

Um ein besseres Verständnis der Theorie zu erlangen wurde ein eigener Monte Carlo Code geschrieben. Als Programmiersprache wurde Python verwendet. Die Parameter der Clusterentwicklung wurden vom maps Code importiert.

4.9.1 Näherungen

Da das Ziel des Codes in der Veranschaulichung der prinzipiellen Abläufe einer Monte Carlo Simulation und deren Kopplung an die Clusterentwicklung liegt, wurden zur Begrenzung des Progammieraufwandes folgende Näherungen getroffen:

1. Beschränkung der Wechselwirkungsterme auf Paare nächster Nachbarn.
2. Beschränkung auf ein Polynom dritter Ordnung zum Fitten von Datenpunkten und Extrapolation zur Bestimmung von Phasenübergängen.
3. Keine automatische Kontrolle der Konvergenz der Energiemittelwerte.

4.9.2 Aufbau

Der Code ist aus mehreren Modulen aufgebaut. Zunächst wird ein dreidimensionales Gitter mit periodischen Randbedingungen generiert, dessen Konfiguration einem beliebigem Grundzustand entspricht. Dabei werden die Spinzustände $\sigma \in \{-1, 1\}$ durch ein vierdimensionales Array σ_{ijkl} definiert, wobei i dem Untergitter bzw. der Basis entspricht. j,k und l entsprechen hier den diskreten räumlichen Koordinaten $x, y, z \in \mathbb{N}$. Von dieser Grundzustandskonfiguration ausgehend, wird in jedem Markov - Schritt zufällig ein Gitterplatz ausgewählt. Ein Austausch des Atoms am ausgewählten Gitterplatz führt zu einer Änderung der Gesamtenergie des Gitters laut Gleichung 2.33. Ist die Differenz der Freien Energie kleiner null oder erfüllt Gleichung 2.34, findet ein Austausch der Spezies am Gitterplatz statt. Andernfalls wird die ursprüngliche Konfiguration beibehalten.

Ist eine definierte Anzahl an Monte Carlo Schritten erreicht, beginnt eine neue Markov-Kette mit $\mu + \Delta\mu$ als Parameter. Zu beachten ist, dass hier keine automatische Prüfung der Konvergenz der Energiemittelwerte erfolgt (siehe Kapitel). Daher ist eine ausreichend große Anzahl an Iterationsschritten zu wählen. Tritt ein Phasenübergang auf, findet eine Variation der Temperatur um ΔT statt. Der Vorgang der Thermodynamischen Integration wird in Abbildung 4.1 veranschaulicht.

Die thermodynamische Integration findet abhängig vom Gültigkeitsbereich der LTE (Low Temperature Expansion) von der Temperatur T_{LTE} aus statt. Der Wert

für das Großkanonische Potential ϕ_{LTE} bei dieser Temperatur, der den Startwert für die Integration bildet, wird aus den ATAT Rechnungen übernommen.

4.9.3 Konvergenz der Energiemittelwerte

Im folgenden wird das Konvergenzverhalten der Energiemittelwerte über die Markov-Kette betrachtet. Das Verhalten der Energie über die Anzahl der Markov-Schritte ist in Abbildung 4.9 für eine Temperatur von 900K dergestellt. Wie in

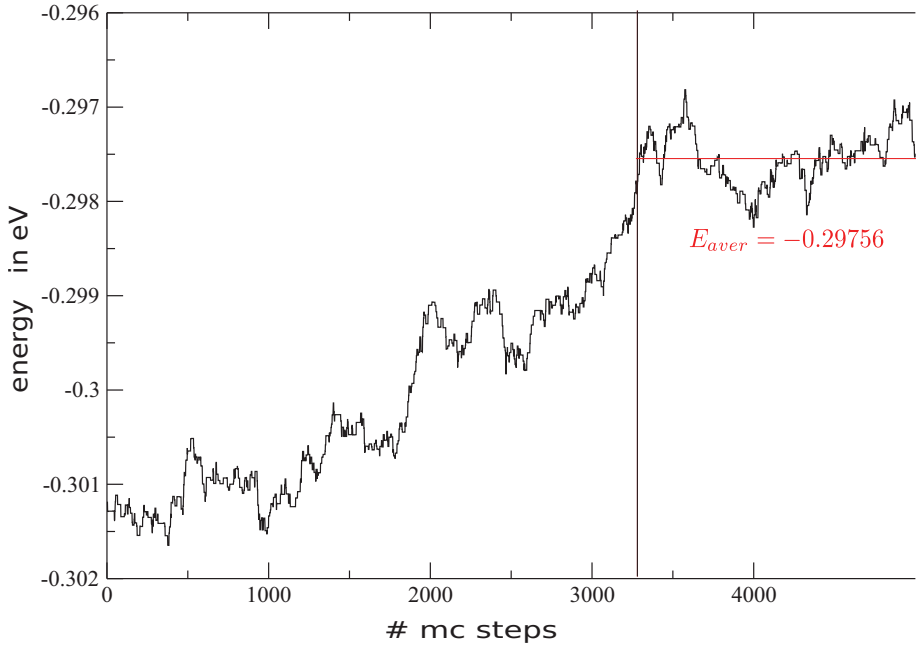


Abbildung 4.9: Konvergenz der mittleren Energie für Grundzustand 1 von TiAl (siehe Kapitel 3)

Abbildung 4.10 gut zu erkennen, sind Spin-Flips bei niedrigen Temperaturen sehr selten. Der Entropietherm der Freien Energie ist klein. Bei höheren Temperaturen wird der Beitrag der Entropie größer. Es ist jedoch eine größere Superzelle für die MC Simulation erforderlich um Konvergenz zu erreichen.

4.9.4 Korrelationen

Zur Analyse des Ordnungszustandes der aktuellen Konfiguration werden verschiedene Korrelationsfunktionen eingeführt.

Korrelationsfunktion

$$\langle \sigma_i; \sigma_j \rangle = \langle \sigma_i \sigma_j \rangle - \langle \sigma_i \rangle \langle \sigma_j \rangle \tag{4.12}$$

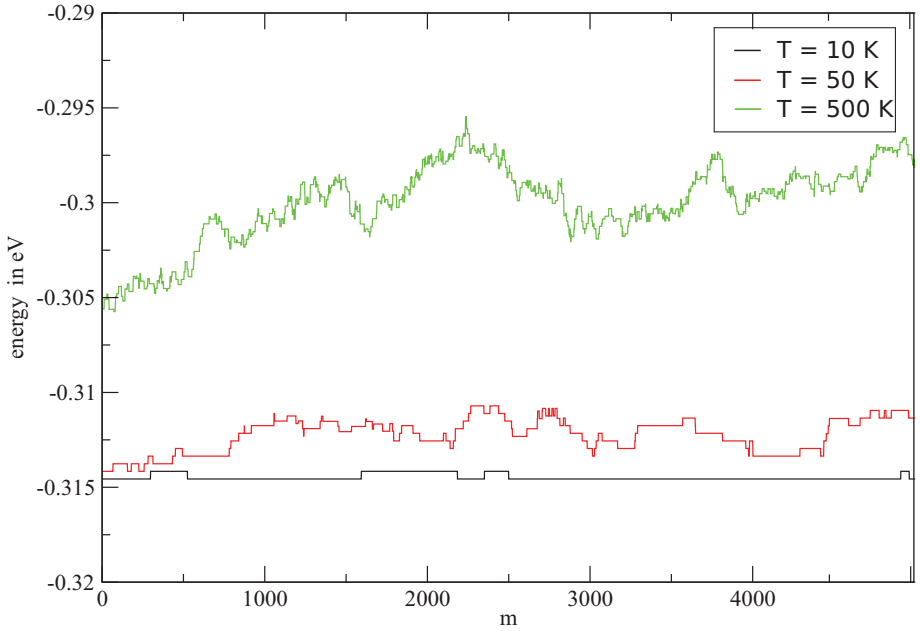


Abbildung 4.10: Energie über Anzahl der Markov-Schritte m bei verschiedenen Temperaturen.

Korrelationsfunktion zur Bestimmung der Fernordnung im Gitter

$$\langle \sigma^{(mc)}_{ijkl} \sigma^{(mc)}_{(i+n)jkl} \rangle \qquad n = 1 \ldots N/2 \tag{4.13}$$

Korrelationsfunktion bezüglich Grundzustand

Um zu beurteilen ob und in welchem Grundzustand sich das Gitter nach erfolgter Konvergenz des Monte Carlo Verfahrens befindet, wird die Korrelation der aktuellen Konfiguration mit den Grundzustandskonfigurationen betrachtet. Es ergibt sich eine Korrelationsmatrix G_m für den m-ten Markov-Schritt

$$\underline{G_m} = \begin{pmatrix} \langle \sigma^{(gs)_1}_{ijkl} \sigma^{(mc)_m}_{ijkl} \rangle \\ \langle \sigma^{(gs)_2}_{ijkl} \sigma^{(mc)_m}_{ijkl} \rangle \\ \vdots \\ \langle \sigma^{(gs)_{N_{gs}}}_{ijkl} \sigma^{(mc)_m}_{ijkl} \rangle \end{pmatrix} \qquad \langle \sigma^{(gs)_1}_{ijkl} \sigma^{(mc)_m}_{ijkl} \rangle \in [-1, 1] \tag{4.14}$$

Die Indizierung $\{(gs)_1 \ldots (gs)_{N_{gs}}\}$ steht hier für die Gesamtheit aller Grundzustände $\{1 \ldots N_{gs}\}$.

Liegt eine Komponente nahe bei eins, kann davon ausgegangen werden, dass sich die betrachtete Konfiguration im entsprechenden Grundzustand befindet. Allerdings ist dies relativ rechenintensiv mit $O(N^3)$ Rechenoperationen.

In Abbildung 4.11 ist der Verlauf der Grundzustands Korrelationsfunktion über die Schritte m der Markov-Kette für zwei unterschiedliche Temperaturen zu erkennen. Als Ausgangskonfiguration werden die Gitterplätze zufällig mit den Spezies Titan und Aluminium besetzt. Die Größe der Superzelle beträgt 5x5x5 Einheitszellen. Bei einer Temperatur von 10K ist eine schnelle Konvergenz in Richtung Grundzustandskonfiguration zu erkennen. Für eine Temperatur von 5000K stellt sich eine Zufallskonfiguration ein. Die Korrelationsfunktion strebt in diesem Fall gegen null.

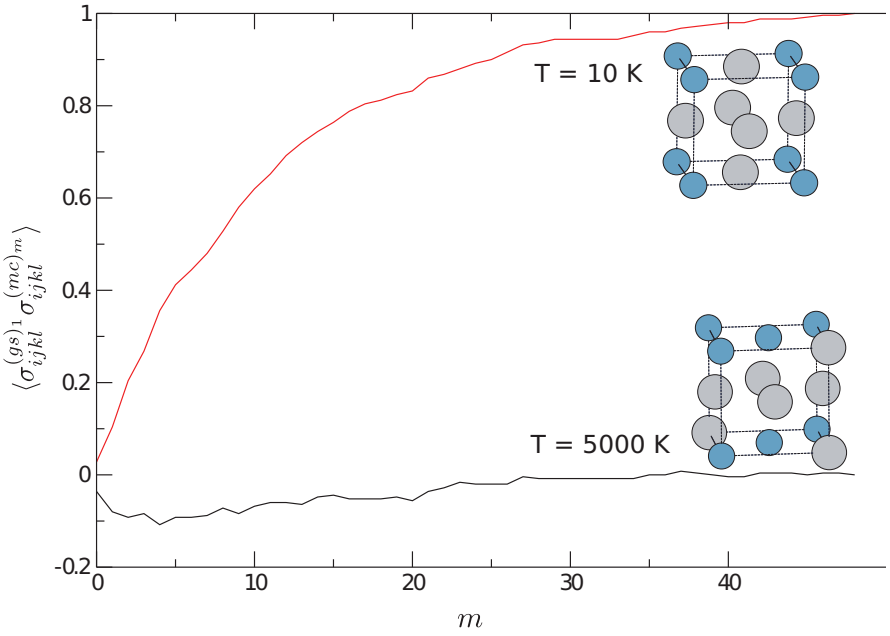


Abbildung 4.11: Korrelationsfunktion bezüglich eines Grundzustandes über die Markov-Kette

Kapitel 5

Zusammenfassung und Diskussion der Ergebnisse

5.1 Qualität der Ergebnisse

Im Vorfeld wurden eine Überprüfung der Cluster Expansion und der DFT Rechnungen durchgeführt. Es kann also davon ausgegangen werden, dass die Parameter der Cluster-Expansion hinreichend genau auskonvergiert sind. Ein weiterer für die Genauigkeit der thermodynamischen Daten ausschlaggebender Beitrag kommt von der Zellengröße der Monte-Carlo-Simulation. Wie die Konvergenztests zeigen kann diese für eine Zellengröße von 50x50x50 Einheitszellen als hinreichend konvergiert betrachtet werden. Die Konvergenz bezüglich der Parameter Differenz im chemischen Potential $\Delta\mu$ und Temperatur ΔT der thermodynamischen Integration ist ebenfalls gegeben.

Auch wenn die MC Simulationen hinsichtlich des Großkanonischen Potentials und der mittleren Konzentration auskonvergiert vorliegen, kann eine Änderung der Schrittweite der Thermodynamischen Integration großen Einfluss auf die Detektion von Phasenübergängen aufweisen. Vor allem wenn die freie Enthalpie der beiden Phasen nicht stark variiert oder eine der Phasen große Streuung in den Daten aufweist. Dies kann unter anderem durch Limitationen der verwendeten Cross Validation Score (CVS) Methode bezüglich der Energie auftreten. Möglicherweise könnte eine zusätzliche Untersuchung der Änderung der Wärmekapazität in der Nähe von Phasengrenzen zuverlässigere Werte liefern. Diese ist proportional zur Ableitung der Energie nach der Temperatur und weist damit Singularitäten höherer Ordnung als die Energie auf.

Es ist zu bemerken, dass die Streuung der Daten für nicht-Grundzustandskonfigurationen die Interpretation der Daten erschwert. Ob eine physikalische Phase auftritt, kann nur durch weiterführende Untersuchungen hinsichtlich der Korrelationsfunktion bezüglich der angenommenen Ordnungsphasen beurteilt werden.

Weiters besteht die Möglichkeit, dass die Genauigkeit der Simulation nur für die Grundzustandsphasen ausreichend ist und ein Erhöhen der Konvergenzparameter für nicht-Grundzustandsfunktionen notwendig wäre. Diesbezüglich wurden allerdings keine genaueren Untersuchungen durchgeführt.

Wie durch andere theoretische Untersuchungen bekannt [31], liefert der phononische Anteil der Entropie in den meisten Systemen einen wesentlichen Beitrag zur freien Energie. Dieser wurde in vorliegender Arbeit nicht behandelt. Es ist laut [31] davon auszugehen, dass dieser eine Verschiebung der Phasengrenzen zu niedrigeren Temperaturen zur Folge hat.

5.2 Ausblick

Ein interessanter Punkt wäre sicherlich die Untersuchung der in Kapitel 2.7 beschriebenen Intergitter-Korrelationsfunktion, um zu beurteilen ob es sich um eine geordnete Phase handelt. Dazu wäre jedoch eine Analyse des gesamten Spingitters erforderlich. Die Korrelationsfunktionen lassen sich jedoch zur Zeit nicht aus den vom emc2 Code ausgegebenen Daten rekonstruieren. Weiters könnte die ungeordnete Phase beziehungsweise die HTE als Ausgangspunkt für eine Thermodynamische Integration gewählt werden, um möglicherweise weitere Phasen aufzufinden.

Literaturverzeichnis

[1] J. Van Humbeeck, *Advanced Engineering Materials* **3**, 837 (2001).

[2] P. Potapov, *et al.*, *Materials Letters* **32**, 247 (1997).

[3] A. van de Walle, G. Ceder, *Journal of Phase Equilibria* **23**, 348 (2002).

[4] J. Spitaler, Cluster expansion data for the ni-ti-hf system (2012). (personal communication).

[5] P. Blaha, K. Schwarz, G. K. H. Madsen, D. Kvasnicka, J. Luitz, WIEN2k, An Augmented Plane Wave + Local Orbital Program for Calculating Crystal Properties, *Tech. rep.*, Vienna University of Technology, Vienna (2001).

[6] A. van de Walle, M. Asta, G. G. Ceder, *Calphad* **26**, 539 (2002).

[7] M. Chakraborty, J. Spitaler, P. Puschnig, C. Ambrosch-Draxl, *Comp. Phys. Comm.* **181**, 913 (2010).

[8] K. Otsuka, X. Ren, *Prog. Mater. Sci.* **50**, 511 (2005).

[9] M. P. D. Stöckel, *Feinwerktechnik & Messtechnik* **95**, 332 (1987).

[10] P. Nash, P. Nash, A. International, *Phase diagrams of binary nickel alloys*, Monograph series on alloy phase diagrams (ASM International, 1991).

[11] M. Zarinejad, Y. Liu, T. J. White, *Intermetallics* **16**, 876 (2008).

[12] W. Kohn, *Reprint from Highlights of Condensed- Matter Theory, Soc. Italiana di Fisica* **Course LXXXIX**, 4 (1985).

[13] M. Born, R. Oppenheimer, *Ann. Physik* **84**, 457 (1927).

[14] W. Kohn, L. J. Sham, *Phys. Rev.* **137**, A1697 (1965).

[15] P. Hohenberg, W. Kohn, *Phys. Rev.* **136**, B864 (1964).

[16] K. Burke, E. K. U. Gross, *Density Functionals: Theory and Applications* (Springer, Berlin, 1998), chap. A Guided Tour of Time-Dependent Density Functional Theory, pp. 116–146.

[17] J. M. Sanchez, *Phys. Rev. B* **48**, 14013 (1993).

[18] J. M. Sanchez, *Phys. Rev. B* **81**, 224202 (2010).

[19] R. Peierls, *Mathematical Proceedings of the Cambridge Philosophical Society* **32**, 477 (1936).

[20] G. Wedler, *Lehrbuch der Physikalischen Chemie* (WILEY-VHC Verlag GmbH Co. KGaA, Weinheim, 2004).

[21] R. Clausius, *Annalen der Physik und Chemie* **XCIII** (1884).

[22] H. Haug, *Statistische Physik* (Springer Verlag, 2.Auflage 2006, XVI).

[23] D. E. Knuth, *The Art of Computer Programming*, Four volumes (Addison-Wesley, 1968). Seven volumes planned (this is a cross-referenced set of BOOKs).

[24] W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling, *Numerical Recipies in Fortran 77* (Cambridge Univ. Press, 2005).

[25] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, E. Teller, *Journal of Chemical Physics* **21**, 1087 (1953).

[26] A. van de Walle, M. Asta, *Modelling and Simulation in Materials Science and Engineering* **10**, 521 (2002).

[27] C. D. et al., Exciting homepage; http://exciting-code.org/.

[28] J. K. Dewhurst, S. Sharma, C. Ambrosch-Draxl, *The EXCITING Code Manual, Version 0.9.224* (2008).

[29] M. Stone, *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **36**, 111 (1974).

[30] G. Bärwolff, *Höhere Mathematik für Naturwissenschaftler und Ingenieure* (Spektrum Akademischer Verlag, 1.Auflage, 2004).

[31] A. van de Walle, G. Ceder, *Rev. Mod. Phys.* **74**, 11 (2002).

Anhang A: Monte Carlo Code

```
from pylab import *
from scipy import *
import latt
import c_energy
import corr
import copy
import matplotlib.pyplot as plt

Nitt = 1000  # total number of Monte Carlo steps

N = 10          # linear dimension of the lattice,
   lattice-size= N x N
warm = 1      # Number of warmup steps
measure=1      # How often to take a measurement

def sample(Nitt, crystal, T, mu):
        "Monte Carlo sampling"
        gscrystal = copy.deepcopy(crystal)

        f = open('energy'+str(T)+'_'+str(mu), 'w')
        f1 = open('x'+str(T)+'_'+str(mu), 'w')
        Mn = 0
        #Ene = CEnergy(latt)  # Starting energy
        gsEne = c_energy.get_energy(crystal, 'ordered')
        Ene = gsEne
        print Ene
        for l in range(len(crystal)):
                for i in range(N):
                        for j in range(N):
                                Mn += sum(crystal[l][i][j
                                   ])
```

```
Naver=0        # Measurements
Eaver=0.0
Maver=0.0

#N2 = N*N
for itt in range(Nitt):
        ene = c_energy.get_energy(crystal, '
           ordered')

        l = int(rand()*len(crystal))
        i = int(rand()*len(crystal[0]))
        j = int(rand()*len(crystal[0][0]))
        k = int(rand()*len(crystal[0][0][0]))

        #S = crystal[l][i][j][k]          #spin of
           lattice point

        ncrystal = copy.deepcopy(crystal)

        ncrystal[l][i][j][k] = ncrystal[l][i][j][
           k]*(-1)

        #dE = c_energy.get_energy(crystal, '
           single', [l,i,j,k], 1) - c_energy.
           get_energy(crystal, 'single', [l,i,j,k
           ], -1)# - crystal[l][i][j][k] * mu

        nene = c_energy.get_energy(ncrystal, '
           ordered')
        dE = - ene + nene - crystal[l][i][j][k] *
            mu
        #print '2', crystal
        P = exp(-abs(dE)/(8.61733*10.**(-5.)*T))
        #print P, dE
        if P>rand() or dE<0: # flip the spin
                crystal[l][i][j][k] = -crystal[l
                   ][i][j][k]
                Ene = nene
                #print 'new',nene
                Mn = crystal[l][i][j][k]
        else: Ene = ene

        if itt>warm and itt%measure==0:
```

```
                        Naver += 1
                        Eaver += Ene
                        Maver += Mn
                        print Eaver/Naver, Maver/Naver
                        f.write(str(Eaver/Naver) + '\n')
                        f1.write(str(Maver/Naver) + '\n')
                if itt%10==0:
                        print corr.corr(gscrystal,
                           crystal)

        return Eaver/Naver, Maver/Naver

if __name__ == '__main__':
        #latt = c_energy.get_energy()

        #Define groundstate:
        la1 = latt.gen_latt(N,1)
        la2 = latt.gen_latt(N,-1)
        la3 = latt.gen_latt(N,1)
        la4 = latt.gen_latt(N,-1)
        crystal = [la1, la2, la3, la4]


        #wT = linspace(4,0.5,100)
        wT = [1.,10.,100.,1000.]
        for T in wT:


                menergy, mx = sample(Nitt, crystal, T,
                   0.00)
```

```
import latt

def get_energy(crystal, *args): #args[0]...mode, args
   [1]....position (if mode = 'single'), args[2].... -1
   for spinflip
        eci0 = -0.249416
        eci1 = 0.029914
        eci2 = 0.032573

        phi0 = 1.
        phi1 = 0.
        phi2 = 0.
```

```
multipl0 = 1.
multipl1 = 1.
multipl2 = 6.



if args[0] == 'rand': mod = 'rand'
elif args[0] == 'single': mod = 'single'
else: mod = 'ordered'

if mod == 'ordered':
        size = len(crystal[0])
        for l in range(len(crystal)):
                for i in range(size):
                        for j in range(size):
                                for k in range(
                                   size):
                                        phi1 =
                                           phi1 +

                                           crystal
                                           [l][i
                                           ][j][k
                                           ]
                                        phi2 =
                                           phi2 +

                                           get_nn
                                           (
                                           crystal
                                           , [l,i
                                           ,j,k],
                                           size
                                           ,1)

        phi2 = phi2/(4.*size**3.)
        phi1 = phi1/(4.*size**3.)
        phi0 = phi0/(4.*size**3.)
        #print phi0*(4.*size**3.), phi1*(4.*size
           **3.), phi2*(4.*size**3.)
        energy = multipl0*phi0*eci0 + multipl1*
```

```
                    phi1*eci1 + multipl2*phi2*eci2

        elif mod == 'single': #modify!
                size = len(crystal[0])
                pos = args[1]   #get energy contribution
                    of nearest neighbors
                phi1 = crystal[pos[0]][pos[1]][pos[2]][
                    pos[3]]*args[2]
                phi2 = get_nn(crystal, [pos[0],pos[1],pos
                    [2],pos[3]],size, args[2])
                phi0 =  phi0/(4.*size**3.)
                phi1 =  phi1/(4.*size**3.)

                phi2 = phi2/(4.*size**3.) #added: is this
                     right?#

                energy = multipl0*phi0*eci0 + multipl1*
                    phi1*eci1 + multipl2*phi2*eci2

        return energy

def get_nn(crystal, pos, size, *args):  #pos = [l,i,j,k]
    l...lattice number i,j,k...xyz positions, *args....
    spinflip
        nn = []
        l = pos[0]
        i = pos[1]
        j = pos[2]
        k = pos[3]

        var = 0
        nbase = len(crystal)
        for la in range(nbase):

                if la == l: continue
                p = [1.,1.,1.]
                p[var] = p[var]*0.

                for m in range(2):
                        for n in range(2):

                                if p[0] == 0: nn.append(
                                    crystal[la%nbase][i%
```

```
                                        size][(j+n)%size][(k+m
                                        )%size])
                                    elif p[1] == 0: nn.append
                                        (crystal[la%nbase][(i+
                                        n)%size][j%size][(k+m)
                                        %size])
                                    elif p[2] == 0: nn.append
                                        (crystal[la%nbase][i%
                                        size][(j+n)%size][(k+m
                                        )%size])
                    var+=1

            return float(sum([x * crystal[l][i][j][k]*args[0]
                for x in nn]))/len(nn)

#get_energy('')
```

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import *

def gen_latt(size, spin):

        latt = []
        for i in range(size):
                latt.append([])
                for j in range(size):
                        latt[i].append([])
                        for k in range(size):
                                if spin == rand: s = sign
                                    (2*rand()-1)
                                elif type(spin) == list:
                                    'lists not handled jet
                                    '
                                else:
                                        s = spin
                                        latt[i][j].append
                                            (s)
        return latt
```

```
def corr(gs_crystal, crystal):
        corr = 0
        size = len(crystal[0])
```

```
        for l in range(len(crystal)):
                for i in range(size):
                        for j in range(size):
                                for k in range(size):
                                        corr +=
                                           gs_crystal[l][
                                           i][j][k] *
                                           crystal[l][i][
                                           j][k]

        return corr/(float(len(crystal))*float(size)**3.)
```

Anhang B: Skripts zur Automatisierung und Auswertung

```
import os
import sys
import matplotlib.pyplot as plt
import subprocess
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from lxml import etree

class MC_out():
    def __init__(self):
        filename = sys.argv[1]#output filename: e.g.'
            gs3_1.out'
        #proc1 = subprocess.Popen(['find out.out'] ,
            shell=True)
        #proc1.communicate()
        cell = [10] #define cell sizes here
        files = [''] #define output file names here
        mu = []
        phi = []
        E_spin = []
        x = []
        T_i = []
        mu_i = []
        phi_i = []
        x_i = []
        C_i = []
        j=0

        root = etree.Element('phase', name=filename)
```

```
for file in files:
    er = etree.SubElement(root, 'cells', er=file
       )
    times = open(file+'times.out')
    t = times.read()
    print t

    pot = []
    data = self.read(file+filename)

    i=0
    T_i.append([])
    mu_i.append([])
    phi_i.append([])
    x_i.append([])
    C_i.append([])

    for T in data:
        xdata = etree.SubElement(er, 'data')
        T_i[j].append(T[0]) #indicate T position
           in phase diagram
        xdata.set('T',str(T[0]))
        mu_i[j].append(T[1])#indicate mu position
            in phase diagram
        xdata.set('mu',str(T[1]))
        phi_i[j].append(T[4])
        xdata.set('phi',str(T[4]))
        x_i[j].append(T[3])
        xdata.set('x',str(T[3]))
        C_i[j].append(T[5])
        xdata.set('C',str(T[5]))
        if i == len(data)-1:
            mu.append(T[1])
            E_spin.append(T[2])
            phi.append(T[4])
            x.append(T[3])
        #elif i == 2: E_spin.append(T[2])
        #elif i == 4: phi.append(T[4])

        i+=1
    j+=1
```

```
        self.fit3d(root, '64')

        with open('data.xml','w') as f:
            f.write(etree.tostring(root, pretty_print=
               True))

        fig1 = plt.figure()
        #plt.plot(cell,E_spin)
        ax = fig1.gca(projection='3d')
        for n in range(len(T_i)):
            ax.plot(mu_i[n], T_i[n],phi_i[n],'o',ms=n+3,
               mfc='None')

        #ax.plot_surface()

        ax.set_xlabel(r'mu')
        ax.set_ylabel(r'T')
        ax.set_zlabel(r'phi')
        fig = plt.figure()
        ax1 = fig.add_subplot(211)
        ax1.plot(cell,x)
        ax2 = fig.add_subplot(212)
        ax2.plot(cell,phi)



        plt.show()

    def read(self, file):
        dataT = []
        f = open(file)

        lines = f.readlines()
        m=0
        for line in lines:

            if len(line) == 1: continue

            n=0
            dataT.append([])
            values = line.split('\t')
            for value in values:
```

```
                if value != '\n': dataT[m].append(float(
                    value))
                n+=1
            m+=1
            #print dataT
        return dataT

    def fit3d(self, tree, er):
        Temp = []
        Mu = []
        fdataT = []
        fdataMu = []

        T_constMu = []
        phi_constMu = []
        constMu = []

        Mu_constT = []
        phi_constT = []
        constT = []

        for T in tree.xpath("//cells[@er = 'er%s/']/data/
            @T"%er):
            if T not in Temp: Temp.append(T)

        for mu in tree.xpath("//cells[@er = 'er%s/']/data
            /@mu"%er):
            if mu not in Mu: Mu.append(mu)

            fp = lambda val: [ float(x) for x in val ]#
                array string values to foating point

        for T in Temp:
            fdataT.append(np.polyfit(fp(tree.xpath("//
                cells[@er = 'er%(er)s/']/data[@T='%(T)s']/
                @mu"%{'er':er,'T':T})),fp(tree.xpath("//
                cells[@er = 'er%(er)s/']/data[@T='%(T)s']/
                @phi"%{'er':er,'T':T})),3))
        for mu in Mu:
            fdataMu.append(np.polyfit(fp(tree.xpath("//
                cells[@er = 'er%(er)s/']/data[@mu='%(mu)s
                ']/@T"%{'er':er,'mu':mu})),fp(tree.xpath("
                //cells[@er = 'er%(er)s/']/data[@mu='%(mu)
```

```
            s']/@phi"%{'er':er,'mu':mu})),3))

        i=0
        for polynom in fdataMu:
            T_constMu.append(np.linspace
               (-0.004537,0.010688,num=100))
            phi_constMu.append(np.polynomial.polynomial.
               polyval(T_constMu[i],polynom))
            constMu.append(np.ones(100)* float(Mu[i]))
            i+=1
        i=0
        for polynom in fdataT:
            Mu_constT.append(np.linspace(273,453,num=100)
               )
            phi_constT.append(np.polynomial.polynomial.
               polyval(Mu_constT[i],polynom))
            constT.append(np.ones(100)* float(Temp[i]))
            i+=1


        return T_constMu, phi_constMu, constMu, Mu_constT
           , phi_constT, constT
MC_out()
```

```
import os
import sys
import matplotlib.pyplot as plt
import subprocess
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from lxml import etree
import matplotlib

class MC_out():
    def __init__(self):

        matplotlib.rc('font', **{'size':20})

        filename = sys.argv[1]#output filename: e.g.'
           gs3_1.out'
        #proc1 = subprocess.Popen(['find out.out'] ,
           shell=True)
        #proc1.communicate()
```

```
colors = ['b','g','r','c','m']
fig1 = plt.figure()
#plt.plot(cell,E_spin)
ax = fig1.gca(projection='3d')

self.tree = etree.parse(filename)

#Tdata = map(float,self.tree.xpath("//cells[@er =
    'er0/']/data/@T"))
#mudata = map(float,self.tree.xpath("//cells[@er
   = 'er0/']/data/@mu"))
#phidata = map(float,self.tree.xpath("//cells[@er
    = 'er0/']/data/@phi"))
#ax.scatter(Tdata,phidata,mudata)

n=0
for elem in self.tree.xpath("//cells"):
    print '%s'%str(n)
    T_constMu, phi_constMu, constMu, Mu_constT,
       phi_constT, constT = self.fit3d(elem, '%s'
       %str(n))
    try:
        ax.plot_surface( constMu,T_constMu,
           phi_constMu, alpha=0.3,rstride=8,
           cstride=8)
    except:
        ax.plot_surface(np.array(constMu),np.
           array(T_constMu), np.array(phi_constMu
           ), alpha=0.7,rstride=8, cstride=8,
           color = colors[n])
    #ax.contour(T_constMu, phi_constMu, constMu,
       zdir='x', offset=np.amin(T_constMu)-50)
    #ax.contour(T_constMu, phi_constMu, constMu,
       zdir='y', offset=np.amax( phi_constMu)
       +0.001)
    #ax.contour(T_constMu, phi_constMu, constMu,
       zdir='z', offset=np.amin(constMu)-0.005)
    #ax.plot_surface(np.array(constT), np.array(
       phi_constT), np.array(Mu_constT), alpha
       =0.3)
    #ax.contour(constT, phi_constT, Mu_constT)
    n+=1
```

```
        #for n in range(len(T_i)):
            #ax.plot(mu_i[n], T_i[n], phi_i[n], 'o',ms=n+3,
                mfc='None')
        #print T_constMu, phi_constMu, constMu

        #T_constMu, phi_constMu, constMu, Mu_constT,
            phi_constT, constT = self.fit3d(elem, '64')
        #try:
        #    ax.plot_surface(T_constMu, phi_constMu,
            constMu, alpha=0.3)
        #except:
        #    ax.plot(T_constMu, phi_constMu, constMu,
            alpha=0.3)
        #ax.contour(T_constMu, phi_constMu, constMu, zdir
            ='z')
        #ax.plot_surface(Mu_constT, phi_constT, constT,
            alpha=0.3)

        #ax.plot_surface()

        ax.set_ylabel(r'$T \ \ in \ K$', fontsize=20)
        ax.set_zlabel(r'$\phi \ \ in\ eV$', fontsize=20)
        ax.set_xlabel(r'$\mu \ \ in\ eV$', fontsize=20)
        #fig = plt.figure()
        #ax1 = fig.add_subplot(211)
        #ax1.plot(cell,x)
        #ax2 = fig.add_subplot(212)
        #ax2.plot(cell,phi)



        plt.show()

    def read(self, file):
        dataT = []
        f = open(file)

        lines = f.readlines()
        m=0
        for line in lines:

            if len(line) == 1: continue
```

```
            n=0
            dataT.append([])
            values = line.split('\t')
            for value in values:
                if value != '\n': dataT[m].append(float(
                   value))
                n+=1
            m+=1
            #print dataT
        return dataT

    def fit3d(self, tree, er):
        Temp = []
        Mu = []
        fdataT = []
        fdataMu = []

        T_constMu = []
        phi_constMu = []
        constMu = []

        Mu_constT = []
        phi_constT = []
        constT = []


        for T in tree.xpath("//cells[@er = 'er%(er)s/']/
           data/@T"%{'er':er}):
            if T not in Temp: Temp.append(T)

        for mu in tree.xpath("//cells[@er = 'er%(er)s/']/
           data/@mu"%{'er':er}):
            if mu not in Mu: Mu.append(mu)

        fp = lambda val: [ float(x) for x in val ]#array
           string values to floating point
        #/cells[@er = 'er%(er)s/']
        for T in Temp:
            fdataT.append(np.polyfit(fp(tree.xpath("//
               cells[@er = 'er%(er)s/']/data[@T='%(T)s']/
               @mu"%{'er':er,'T':T})),fp(tree.xpath("//
               cells[@er = 'er%(er)s/']/data[@T='%(T)s']/
               @phi"%{'er':er,'T':T})),3))
```

```
for mu in Mu:
    fdataMu.append(np.polyfit(fp(tree.xpath("//
       cells[@er = 'er%(er)s/']/data[@mu='%(mu)s
       ']/@T"%{'er':er,'mu':mu})),fp(tree.xpath("
       //cells[@er = 'er%(er)s/']/data[@mu='%(mu)
       s']/@phi"%{'er':er,'mu':mu})),3))

i=0
for polynom in fdataMu:
    thisT = fp(tree.xpath("//cells[@er = 'er%(er)
       s/']/data[@mu='%(mu)s']/@T"%{'er':er,'mu':
       Mu[i]}))
    T_constMu.append(np.linspace(min(thisT),max(
       thisT),num=100))
    #try:
    #    phi_constMu.append(np.polynomial.
       polynomial.polyval(polynom,T_constMu[i]))
    #except:
    phi_constMu.append(np.polyval(polynom,
       T_constMu[i]))
    constMu.append(np.ones(100)* float(Mu[i]))

    i+=1

i=0
for polynom in fdataT:
    thisMu = fp(tree.xpath("//cells[@er = 'er%(er
       )s/']/data[@T='%(T)s']/@mu"%{'er':er,'T':T
       }))
    Mu_constT.append(np.linspace(min(thisMu),max(
       thisMu),num=100))
    #try:
    #    phi_constT.append(np.polynomial.
       polynomial.polyval(Mu_constT[i],polynom))
    #except:
    phi_constT.append(np.polyval(polynom,
       Mu_constT[i]))
    constT.append(np.ones(100)* float(Temp[i]))
    i+=1


return T_constMu, phi_constMu, constMu, Mu_constT
   , phi_constT, constT
```

MC_out()

```
import matplotlib.pyplot as plt
import numpy as np
import sys
from lxml import etree

fi = sys.argv[1]

f = open(fi)

tree = etree.parse(f)
N = len(tree.xpath('//step'))
print N
for i in range(N):
        B = tree.xpath('//step[%s]/graph/@bulk_mod'%str(i
           +1))
        x = np.linspace(1,10,num = len(B))
        plt.plot(x,B, label = str(i+1))
plt.legend()
plt.show()
print B
```

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

class Plot_str():
    def __init__(self, coord, atoms, fname):
        self.fname = fname
        self.coord = coord
        self.atoms = atoms
        self.x = []
        self.y = []
        self.z = []
        self.spec = []
        if self.coord == []:
            self.read_coord()
            self.fromfile = True
        else:
            self.fromfile = False

        self.plot_latt()
```

```
def plot_latt(self):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    n = 1
    if self.fromfile:
        for n in range(len(self.x)):
            if self.spec[n] == 'Hf': col = 'g'; size
               = 50
            elif self.spec[n] == 'Ni': col = 'b';
               size = 200
            else: col = 'r'; size = 100
            ax.scatter([self.x[n]],[self.y[n]],[self.
               z[n]],s = size, c = col)
        #ax.text(point[0],point[1],point[2],str(n))
    else:
        for point in self.coord:
            print point
            ax.scatter([point[0]],[point[1]],[point
               [2]],s=200)
            ax.text(point[0],point[1],point[2],str(n)
               )
    plt.show()
    return

def read_coord(self):
    for line in file(self.fname):
        line = line.split()
        if len(line) == 4:
            print line
            self.x.append(float(line[0]))
            self.y.append(float(line[1]))
            self.z.append(float(line[2]))
            self.spec.append(line[3])
        else: continue

def plot_atoms(self):

    return
```

```
Plot_str([],[],'mcsnapshot.out')
```

```
#import numpy
import subprocess
import sys
import os
import shutil
try:
        from lxml import etree
except:
        import xml.etree.ElementTree as etree

class MC(object):

        def __init__(self):
                try:
                        from lxml import etree
                        self.lxml = True
                except:
                        import xml.etree.ElementTree as
                           etree
                        self.lxml = False
                self.infile = str(sys.argv[1])
                self.dir = str(sys.argv[0]).rstrip('
                   set_mcrun.py')
                self.calcdir = str(os.getcwd())
                print sys.argv[0]
                print self.dir

                self.dirs()
                if str(sys.argv[2]) == 'l':
                        self.loadl()
                else:
                        self.shellcommand()

                return

        def dirs(self):
                proc1 = subprocess.Popen(['xsltproc ' +
                   self.dir + 'mc_dirs.xsl %s'% self.
                   infile] , shell=True)
                proc1.communicate()
```

```
f = etree.parse(self.infile)
if self.lxml:

        self.paths = f.xpath('/experiment
           /set/@path')
        for path in self.paths:
                shutil.copy(self.calcdir
                   + '/lat.in',self.
                   calcdir + '/' + path +
                    '/lat.in')
                shutil.copy(self.calcdir
                   + '/clusters.out',self
                   .calcdir + '/' + path
                   + '/clusters.out')
                shutil.copy(self.calcdir
                   + '/eci.out',self.
                   calcdir + '/' + path +
                    '/eci.out')
                shutil.copy(self.calcdir
                   + '/gs_str.out',self.
                   calcdir + '/' + path +
                    '/gs_str.out')
                print os.getcwd()
else:

        elements = f.getroot().findall('
           set')
        for element in elements:
                shutil.copy(self.calcdir
                   + '/latt.in',self.
                   calcdir + '/' +
                   element.get('path') +
                   '/')
                shutil.copy(self.calcdir
                   + '/clusters.out',self
                   .calcdir + '/' +
                   element.get('path') +
                   '/')
                shutil.copy(self.calcdir
                   + '/eci.out',self.
                   calcdir + '/' +
```

```
                                        element.get('path') +
                                        '/')
                                    shutil.copy(self.calcdir
                                        + '/gs_str.out',self.
                                        calcdir + '/' +
                                        element.get('path') +
                                        '/')
                            print 'Done'

                return

        def loadl(self):
                proc1 = subprocess.Popen(['xsltproc ' +
                    self.dir + 'loadleveler.xsl %s'% self.
                    infile] , shell=True)
                proc1.communicate()
                return
        def shellcommand(self):
                proc1 = subprocess.Popen(['xsltproc ' +
                    self.dir + 'shelcommand.xsl %s'% self.
                    infile] , shell=True)
                proc1.communicate()
                return

if __name__=='__main__':
        MC()

#mcsetup = MC()
```