Monte Carlo-Simulation von Langmuir-Monolagen

Dissertation zur
Erlangung des Grades
„Doktor der Naturwissenschaften“
am Fachbereich Physik der
Johannes-Gutenberg-Universität Mainz

vorgelegt von
Christoph Stadler

Mainz 1998

Erster Berichterstatter: *Prof. Dr. K. Binder*

Zweiter Berichterstatter: *P.D. Dr. F. Schmid*

Mündliche Prüfung: *Juli 1998*

Inhaltsverzeichnis

1 Einleitung **3**

2 Modell und Simulationsprinzip **7**
2.1 Modellbeschreibung 7
2.2 Wahl der Parameter 10
2.3 System-Hamiltonian 11
2.4 Wesentliche Observablen 13
2.5 Strukturelle Information 15
2.5.1 Radiale Paarverteilungsfunktion 15
2.5.2 Strukturfaktor 17

3 Simulationstechnik **23**
3.1 Notwendigkeit einer scherbaren Box 23
3.2 Kollektive Moves 25
3.2.1 Translations-Moves 25
3.2.2 Configurational Biased Monte Carlo 26
3.3 Beschleunigung der Simulation 31

4 Bestimmung der freien Energie **35**
4.1 Thermodynamische Integration 35
4.2 Harmonische Näherung 36
4.3 Teildiagonalisierung durch Ausnutzung der Periodizität 38
4.4 Berechnung des harmonischen Gibbs-Potentials 41
4.5 Bestimmung der Übergangstemperatur 44

5 Simulationsergebnisse **49**
5.1 Grundzustandsanalyse 49
5.2 System kleiner Köpfe 51
5.2.1 Verhalten bei niedrigem Druck 52
5.2.2 Verhalten bei hohem Druck 56
5.2.3 Phasendiagramm 64
5.3 System großer Köpfe 68
5.3.1 Modulation 68

5.3.2 Simulationsdaten 74
5.3.3 Hysterese 78
5.3.4 Auftreten der Gasphase 81
5.3.5 Phasendiagramm 87
5.3.6 Abhängigkeiten von der Kettenlänge 91
5.3.7 Finite-Size-Effekte 94
5.4 Sehr große Köpfe 98

6 Zusammenfassung und Ausblick 101

A Hamiltonian für das $N\Pi T$-Ensemble 105

B Berechnung der Kopplungskonstanten 109

C Thermische Expansion und harmonische Näherung 111

D Programm-Listing 113

Stichwortverzeichnis 175

Kapitel 1

Einleitung

Bei Langmuir-Monolagen handelt es sich um mono-molekulare Schichten wasserunlöslicher Moleküle an der Grenzfläche von Wasser und Luft. Derartige Schichten werden von *amphiphilen* Molekülen gebildet, die eine hydrophile Kopfgruppe und einen (oder mehrere) hydrophobe Schwänze besitzen. Dieser Zwittercharakter bewirkt eine Bindung der Moleküle an die Wasseroberfläche, ohne daß sie sich im Wasser lösen bzw. auf ihm aggregieren könnten.
Einkettige Moleküle[1] , die diese amphiphile Eigenschaft aufweisen sind beispielsweise Fettsäuren A_n-COOH, längerkettige Alkohole A_n-OH oder Ester A_n-COO-CH_3, wobei A_n eine Alkylgruppe mit 10-20 Kohlenstoffatomen repräsentiert.

Von Interesse sind Langmuir-Monolagen zum einen, da man an ihnen thermodynamische Eigenschaften eines (quasi-)zweidimensionalen Systems untersuchen kann, ohne störenden Einflüssen durch die Gitterstruktur eines festen Substrats ausgesetzt zu sein. Zum anderen stellen sie einen Zwischenschritt zum Studium amphiphiler Doppelschichten dar, die der wesentliche Bestandteil biologischer Membranen sind. Beim Aufziehen der Monolage auf einen festen Träger mittels der sog. Langmuir-Blodgett-Technik [12] bilden sich stabile Mono- und Multilayer mit definierter Struktur und Zusammensetzung. Deren mögliche Anwendungen liegen z.B. auf dem Gebiet der Biosensoren, der nichtlinearen Optik oder Microlithographie [8].
Die „klassische“ Untersuchungsmethode amphiphiler Monolagen ist die Messung von Druck-Flächen-Isothermen, wie sie auch bereits vom Namensgeber *I. Langmuir* durchgeführt wurden [43][44].
Modernere Verfahren, die in Zusammenhang mit Langmuir-Monolagen Verwendung finden, sind u.a. Röntgenstreuung [34][5], Fluoreszenzmikroskopie (PFM) [48][50] oder Brewsterwinkel-Mikroskopie (BAM) [30][31].

Der gemeinsame Befund dieser experimentellen Untersuchungen ergibt ein sehr komplexes Phasendiagramm, welches für eine größere Zahl amphiphiler Substanzen ei-

[1] Da sich die durchgeführten Simulationen auf einkettige Moleküle beschränkten, seien mehrkettige Amphiphile, wie z.B. Phospholipide, hier außer acht gelassen.

ne ähnliche Topologie aufweist[2], so daß man von einem „generalisierten“ Phasendiagramm für Langmuir-Monoschichten sprechen kann:

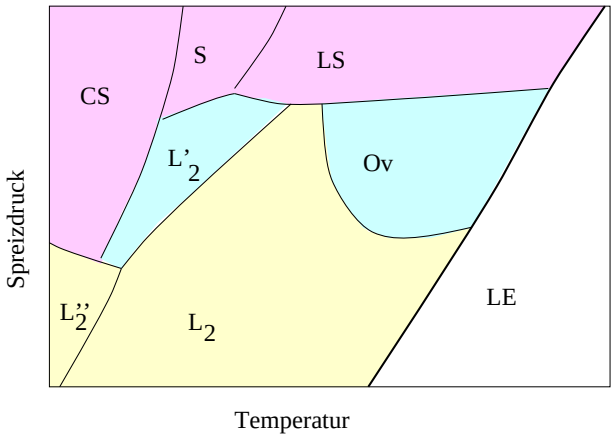


Abbildung 1.1:
charakteristisches Phasendiagramm
für Monolagen von Fettsäuren auf Wasser
nach [20]

Die Auftragung in diesem Diagramm erfolgt als Spreizdruck Π gegen Temperatur T. Der Spreizdruck ist das 2-dimensionale Analogon des Drucks p:
$p := \frac{F}{A} \mapsto \Pi := \frac{F}{l}$

Die Phasen der Monoschichten können durch die vier Ordnungsparameter der smektischen Phasen beschrieben werden [39]:

PO : Positionsordnung der Kopfgruppen in der Grenzfläche
BO : Bondordung der Kopfgruppen in der Grenzfläche
TO : Neigungsrichtung der Ketten bzgl. der Bonds des Kopfgruppengitters (*tilt order*)
HO : Anordnung der Ketten, falls deren azimutale Symmetrie gebrochen ist (*herringbone order*)

Die im char. Phasendiagramm eingetragenen Phasen werden durch diese Ordnungsparameter wie folgt unterschieden[3]:

Monolagen-Phase	PO	HO	BO	TO	T-B	smektische Phase
LE	s	s	s	s		A
LS	s	s	l	s		BH
S	s	l	l	s		E
L_2'	s	s	l	l	NNN	F
L_2	s	s	l	l	NN	I
L_2''	s	l	l	l	NN	K
Ov	s	s	l	l	NNN	L
CS	l	l	l	-		

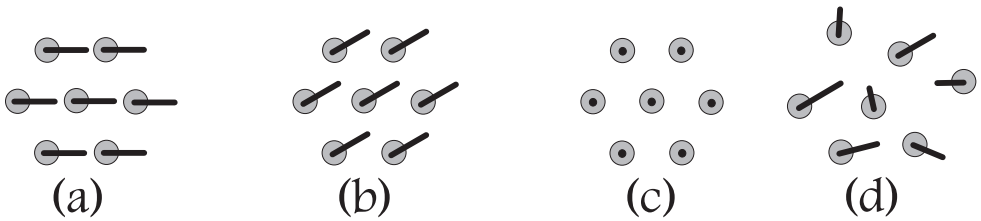
[2] Z.B. verschiebt bei Fettsäuren eine um ein C-Atom längere Alkylgruppe die Temperaturskala um ca. 8 K nach oben, ohne die Existenzgebiete der Phasen wesentlich zu verändern [44][35].

[3] „s“ und „l“ bedeuten hierbei kurz- bzw. langreichweitig, d.h. exponentiellen bzw. algebraischen Zerfall.

Eine Darstellung der Landau-Theorie dieser Phasen findet man z.B. bei Kaganer [33].

Von den vier Ordnungsparametern werden im weiteren die Positionsordnung PO und die Herringbone-Ordnung HO außer acht gelassen, da zum einen eine langreichweitige Positionsordnung mit der geringen Größe des simulierten Systems nicht festgestellt werden kann und zum anderen das verwendete Modell der Ketten keine azimutale Asymmetrie aufweist und daher eine Herringbone-Ordnung nicht auftreten kann.
Die Phasen mit langreichweitiger Bondordnung werden unter dem Begriff „liquid condensed“ (LC) zusammengefaßt und zeigen ein hexagonales, ggf. auch rechteckig-zentriertes Gitter der Kopfgruppen in der Grenzfläche.
Dagegen entspricht die Anordnung der Köpfe in der „liquid expanded“ (LE) Phase einer zweidimensionalen Flüssigkeit.
Im Gegensatz zu Monolagen auf kristallinen Substraten können, wenn wie hier Wasser die Rolle der Subphase übernimmt, keine Kommensurabilitäts-Effekte zwischen den Ketten und dem Substratgitter entstehen.

Langreichweitige Neigungsordnung TO bedeutet, daß die Ketten kollektiv gegen die Flächennormale geneigt sind, wobei eine Korrelation zwischen den Bonds des Kopfgruppengitters und der Richtung der kollektiven Neigung besteht. Es treten dabei zwei Fälle auf – Neigung in Richtung nächster Nachbarn (NN) und – jeweils dazwischen – Neigung zu übernächsten Nachbarn (NNN).



[Draufsicht]: kollektive Neigung (a) zu nächsten, (b) zu übernächsten Nachbarn;
(c) kollektiv ungeneigte Ketten; (d) kurzreichweitige Bondordnung, unkorrelierte Neigung;
(a)(b): TO langreichweitig, (c)(d) TO kurzreichweitig

Kurzreichweitige Neigungsordnung kann zwei Ursachen haben, entweder das Fehlen der Bondordnung, wie in der LE-Phase, in der die Ketten nahezu unabhängig voneinander zufällig geneigt sind, oder den Fall, daß die Ketten kollektiv senkrecht auf der Grenzfläche stehen.
Daß hier im Zusammenhang mit Flüssigkeiten von *Richtungen* nächster und übernächster Nachbarn gesprochen werden kann, liegt daran, daß es sich bei den flüssig-kondensierten Phasen um *hexatische* Phasen [35][57] handelt – wenn man von der CS-Phase absieht, die zusätzlich langreichweitige Positionsordnung besitzt. In den hexatischen Phasen ist die Positionsordnung der Köpfe zwar kurzreichweitig, es bestehen aber langreichweitige Korrelationen in den Bondrichtungen des Kopfgitters, wobei „Bond“ die Verbindung zwischen einer Kopfgruppe und einem ihrer nächsten Nachbarn bezeichnet. Dadurch sind die Richtungen unter denen nächste Nachbarn zu finden

sind über viele Gitterabstände hinweg einheitlich[4].

Betrachtet man nur die Ordnungsparameter der kollektiven Neigung (TO) und der Bondordnung (BO) des Kopfgruppengitters, kann man noch vier Phasen unterscheiden, die in Abbildung 1.1 farblich voneinander abgesetzt sind:

Modell-Phase	BO	TO	
LC-NNN	l	l	kollektive Neigung zu übernächsten Nachbarn *(Liquid Condensed – Next Nearest Neighbors)*
LC-NN	l	l	kollektive Neigung zu nächsten Nachbarn *(Liquid Condensed – Nearest Neighbors)*
LC-U	l	s	Ketten kollektiv senkrecht auf Grenzfläche *(Liquid Condensed – Untilted)*
LE	s	s	Kopfgruppen ungeordnet, zufällige Neigung *(Liquid Expanded)*

Dieses werden die Phasen sein, die mit den durchgeführten Simulationen beobachtet werden können.
In den Experimenten findet man eine weitere Phase, die sich im bisherigen Bild nicht von der flüssig-expandierten LE-Phase unterscheidet, jedoch durch einen Phasenübergang 1. Ordnung davon getrennt ist. Diese „gasartige“ G-Phase kennzeichnet eine deutlich geringere Dichte als LE und entspricht gut der Vorstellung eines zweidimensionalen Gases, in der die Abstände zwischen den Molekülen so groß sind, daß diese untereinander kaum noch wechselwirken.
Da sie erst bei höheren Temperaturen existiert, ist sie in Abbildung 1.1 nicht verzeichnet.

[4] *Helm et al.* [29] fanden experimentell Bond-Korrelationslängen von $\mathcal{O}(10^{-5}\text{m})$ bei gleichzeitiger kurzreichweitiger Positionsordnung von nur $\mathcal{O}(10^{-9}\text{m})$.

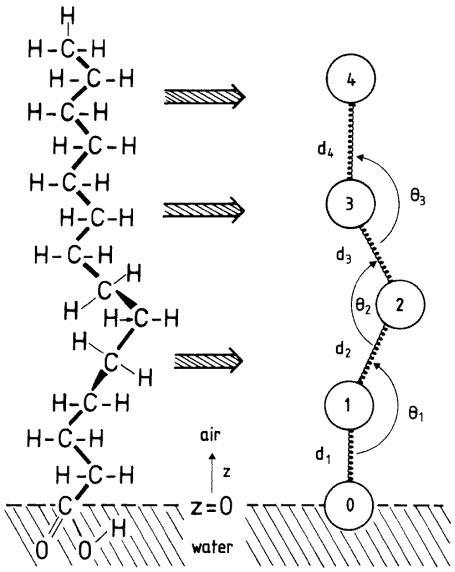
Kapitel 2

Modell und Simulationsprinzip

2.1 Modellbeschreibung

Das Modell, an dem die vorliegenden Simulationen durchgeführt wurden, und wie es bereits *F.M. Haas* [25] und *H.O. Lange* [42] eingesetzt haben, sollte zum einen einfach genug sein, um den nötigen Rechenaufwand bewältigen zu können, zum anderen detailliert genug, um wesentliche Eigenschaften realer Langmuir-Monoschichten wiederzugeben.

Hierzu wurde ein vergröbertes *Kontinuumsmodell* der amphiphilen Moleküle verwendet. In den Alkanketten, welche die hydrophoben Schwänze der Amphiphile bilden, wurden jeweils 2-3 CH_2-Gruppen zu einem *effektiven* Monomer zusammengefaßt. Dies stellt gegenüber dem *united atom*-Modell [7][28] eine weitere Vergröberung dar, da darin jede CH_2- bzw. CH_3-Gruppe als eigenes Teilchen behandelt wird, wodurch die Struktur der Alkankette erhalten bleibt.



Der Zusammenhalt der Kette und ihre Steifigkeit resultieren aus den Bondlängen- und Bondwinkelpotentialen die man zwischen diesen effektiven Monomeren wirken läßt.

Auch der hydrophile Kopf, der in einem realen Molekül z.B. eine Fettsäuregruppe COOH wäre, wird durch ein effektives (Kopf-)Monomer repräsentiert.

Die attraktive Wechselwirkung zwischen Wasser und hydrophilem Kopf bzw. die repulsive zwischen Wasser und dem hydrophoben Schwanz führt zu einer Bindung des Amphiphils an die Wasser-Luft-Grenzfläche. Im Modell ist diese Grenzfläche die $z = 0$-Ebene. Die Bindung erfolgt dadurch, daß sich die Kopfgruppen in dieser Ebene zwar frei bewegen dürfen, sie aber nicht verlassen können, während die Schwanzmo-

nomere auf den (Luft-)Halbraum oberhalb der Grenzfläche beschränkt sind.

Obige Skizze läßt erkennen, warum im betrachteten Modell keine *herringbone*-Ordnung der Ketten auftreten kann. Die abgebildete Alkankette liegt mit Ausnahme zweier *gauche*-Defekte im *all trans*-Grundzustand vor [9], wobei die C-Atome einen Zickzack bilden, d.h. die Kette hat einen anisotropen Querschnitt. Die Modellkette hingegen ist zylinder-symmetrisch, bis auf den Knick, der von den *gauche*-Defekten verursacht wird. Während es bei den anisotropen Molekülen die Möglichkeit gibt, die Rückgrat-Ebenen verschieden anzuordnen, was zur *herringbone*-Ordnung führt, ist diese Möglichkeit bei den isotropen Modellketten nicht gegeben. Experimentell entspricht diese Symmetrie den *Rotator*-Phasen, bei denen aufgrund der thermisch angeregten Bewegung die Ketten annähernd zylindrisch erscheinen.

Die intermolekularen Kräfte werden durch Lennard-Jones-Potentiale zwischen den Schwanzmonomeren, die nicht direkt durch einen Bond[1] verbunden sind berücksichtigt. Die Wechselwirkung der Kopfgruppen untereinander und mit den Schwanzmonomeren erfolgt über sogenannte *Soft Core*-Potentiale.
Bei dem **Bondlängenpotential** $V_{\mathrm{BL}}(d)$, das jeweils zwischen zwei benachbarten Monomeren einer Kette wirkt, handelt es sich um ein *FENE*-Potential[2], das um seine Gleichgewichtslänge d_0 herum harmonisch verläuft, die maximale Auslenkung aber bei $|d - d_0| \lesssim d_{\mathrm{BL}}$ dadurch begrenzt, daß es dort divergiert:

$$V_{\mathrm{BL}}(d) = -\frac{\epsilon_{\mathrm{BL}}}{2} d_{\mathrm{BL}}^2 \ln\left(1 - \left(\frac{d - d_0}{d_{\mathrm{BL}}}\right)^2\right) \quad \text{mit} \quad |d - d_0| < d_{\mathrm{BL}} \tag{2.1.1}$$

Das **Bondwinkelpotential** $V_{\mathrm{BW}}(\theta)$ zwischen je zwei benachbarten Bonds einer Kette hat die Form

$$V_{\mathrm{BW}}(\theta) = \epsilon_{\mathrm{BW}} \cdot (1 - \cos\theta)\,, \tag{2.1.2}$$

d.h. sein Minimum liegt bei $\theta = 0$, dem Fall der *gestreckten* Kette.

Um die Zahl der Wechselwirkungspartner jedes Monomers zu begrenzen, wurde das **Lennard-Jones-Potential** $V_{\mathrm{LJ}}(r)$ bei $r \geq r_{\mathrm{cut}}$ abgeschnitten und, um die Stetigkeit zu erhalten, dort auf Null verschoben:

$$\begin{aligned} V_{\mathrm{LJ}}(r) &= \epsilon_{\mathrm{LJ}} \cdot \left(\left(\frac{\sigma}{r}\right)^{12} - 2\left(\frac{\sigma}{r}\right)^{6}\right) - V_{\mathrm{cut}} && \text{für } r \leq r_{\mathrm{cut}} \text{ mit } r_{\mathrm{cut}} = 2\sigma \\ &= 0 && \text{für } r > r_{\mathrm{cut}} \end{aligned} \tag{2.1.3}$$

[1] Mit „Bond“ wird hier die Verbindung zweier aufeinander folgender Monomere einer Kette bezeichnet. Im Zusammenhang mit dem Kopfgruppengitter sind „Bonds“ die Verbindungen eines Kopfes mit seinen (6) nächsten Nachbarn.

[2] Finitely Extendible Nonlinear Elastic, vgl. [17]

Aus dem Lennard-Jones-Potential erhält man das repulsive **Soft Core-Potential** $V_{sc}(r)$ der Köpfe, indem man nur dessen abstoßenden Teil verwendet, d.h. schon bei σ_H abschneidet:

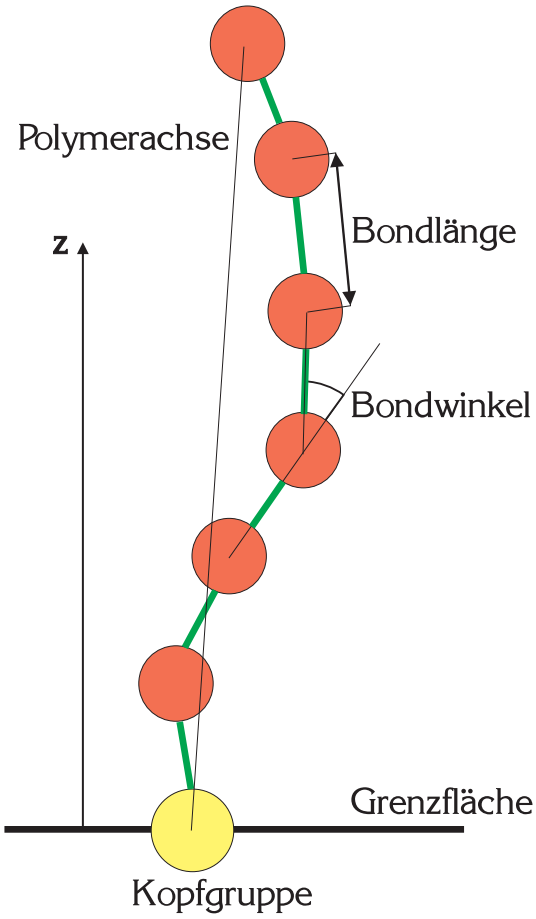
$$\begin{aligned} V_{SC}(r) &= \epsilon_{SC} \cdot \left(\left(\frac{\sigma_H}{r}\right)^{12} - 2\left(\frac{\sigma_H}{r}\right)^{6} + 1 \right) && \text{für } r \leq \sigma_H \\ &= 0 && \text{für } r > \sigma_H \end{aligned} \tag{2.1.4}$$

Den Größenunterschied zwischen der Kopfgruppe und den effektiven Schwanzmonomeren wird dadurch berücksichtigt, daß man $\sigma_H = 1.0...1.3\sigma$ setzt. Obiges Potential gilt für die Wechselwirkung zwischen zwei Kopfgruppen. Für das Potential zwischen einer Kopfgruppe und einem Schwanzmonomer ersetzt man σ_H durch $(\sigma+\sigma_H)/2$.

Ein Eindringen der Schwänze in den Wasserhalbraum $z < 0$ bzw. ein Verlassen der Grenzfläche durch die Köpfe wurde zwar im Simulationsprogramm explizit ausgeschlossen, könnte aber auch durch die zusätzlichen Potentiale

$$V_{\text{Kopf}}(z) = -\delta(z) \quad \text{und} \quad V_{\text{Schwanz}}(z) = V_\infty \cdot \Theta(-z) \tag{2.1.5}$$

verhindert werden.



2.2 Wahl der Parameter

Es erweist sich als günstig, die Bindungsenergie und den Gleichgewichtsabstand des Lennard-Jones-Potentials als Einheiten zu wählen:

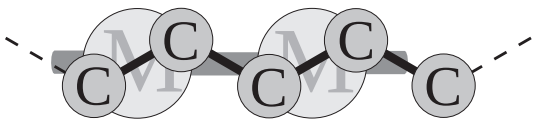
$$[\text{Energie}] = E_0 := \epsilon_{\text{LJ}} \quad [\text{Länge}] = l_0 := \sigma \qquad (\text{LJ-Einheiten})$$

Mit $k_B := 1$ sind dann auch Einheiten für Temperatur und Spreizdruck definiert:

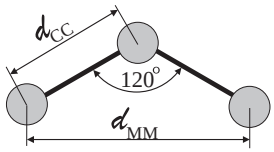
$$[\text{Temperatur}] = T_0 := E_0/k_B \quad [\text{Spreizdruck}] = \Pi_0 = E_0/l_0^2$$

Da in den Simulationen nicht beabsichtigt wurde, Daten für ein bestimmtes amphiphiles Molekül zu reproduzieren, was schon daran zu erkennen ist, daß ein effektives Monomer „2-3“ CH_2-Gruppen repräsentiert, ist eine *genaue* Umrechnung der verwendeten Einheiten in experimentell zugängliche Meßgrößen weder möglich noch sinnvoll. Natürlich möchte man trotzdem einen vagen Vergleich mit realen Zahlenwerten anstellen können.

Vereinfacht man wie in [9] eine Alkankette im *all-trans*-Zustand zu einem Zickzack aus Kohlenstoffatomen, die durch Federn und ein Torsionspotential verbunden sind, und setzt zwischen den C-Atomen verschiedener Ketten ein LJ-Potential an, so ist ein realistischer Wert für dessen Bindungsenergie $\epsilon_0 = 500\,\text{J/mol}$. Faßt man je zwei C-Atome zu einem Monomer zusammen, enthält das LJ-Potential zwischen zwei Monomeren die Wechselwirkungen von $2 \times 2 = 4$ C-Atomen und man erhält $E_0 := 4 \times 500\text{J}/N_A$.



Nimmt man für die LJ-Potentiale zwischen den C-Atomen und den Monomeren den selben Gleichgewichtsabstand σ an, findet man $\sigma := 3.79\,\text{Å}$. Damit kann man die Gleichgewichtslänge d_0 des Bondlängenpotentials bestimmen:
Die Länge einer C–C–Bindung beträgt $d_{CC} = 1.53\,\text{Å}$. Geht man von einem Knickwinkel von 120° entlang der Kette aus, erhält man für den Abstand d_{MM} zweier Monomere $d_{MM} = d_{CC}\sqrt{2(1-\cos 120°)}$. Dies ergibt eine Gleichgewichtslänge des Bondlängenpotentials von $d_0 = d_{MM} = 2.65\text{Å} \approx 0.7\sigma$.



Für die Temperatur und den Spreizdruck bedeutet das:

$$T_0 = \frac{2000\text{J/mol}}{6.022\cdot 10^{23}\cdot 1.38\cdot 10^{-23}\text{J/K}} = 241\,\text{K} \qquad \Pi_0 = \frac{2000\text{J}}{6.022\cdot 10^{23}\cdot (3.79\cdot 10^{-10}\text{m})^2} = 23\,\text{mN/m}$$

Experimente mit Langmuir-Monolagen finden typischerweise im Parameterbereich $\Pi = 0...70\text{mN/m}$ und $\text{T} = 0°\text{C}...100°\text{C}$ statt. Der Temperatur sind Grenzen gesetzt,

da das Wasser als Substrat im flüssigen Zustand vorliegen muß; der Druck wird nach oben durch die Oberflächenspannung des Wassers bzw. die Zerstörung der monomolekularen Anordnung der Ketten [66] eingeschränkt.
Diesem Parameterbereich entspricht in LJ-Einheiten Π = 0...3.0 und T = 1.13...1.55.

Die übliche Länge der simulierten, monodispersen Ketten beträgt 7 Monomere, was wegen der Interpretation als effektive Monomere Alkanketten mit 14–21 Kohlenstoffatomen entspricht.
In den angegebenen Potentialen sind 8 Parameter festzulegen:
- Bondlängenpotential : Stärke ϵ_{BL}, Gleichgewichtslänge d_0, max. Auslenkung d_{BL}
- Bondwinkelpotential : Stärke ϵ_{BW}
- Lennard-Jones-Potential : Stärke ϵ_{LJ}, Gleichgewichtsabstand σ
- Soft Core-Potential : Stärke ϵ_{SC}, Reichweite σ_H

Das Einheitensystem innerhalb der Simulation ist frei wählbar und wurde mit $\epsilon_{\mathrm{LJ}} := 1$, $\sigma := 1$ definiert.
Der Gleichgewichtsabstand σ kann als Durchmesser eines Monomers verstanden werden. Da man erwartet, daß die hydrophilen Kopfgruppen aufgrund der Hydratation etwas größer sind als der Durchmesser der Alkankette, und die abstoßende Wirkung des Soft Core-Potentials bei σ_H beginnt, verwendet man $\sigma_H := 1.0\sigma...1.5\sigma$ mit $\epsilon_{\mathrm{SC}} := \epsilon_{\mathrm{LJ}} = 1$.

Geht man davon aus, daß die Moleküle relativ steife Stäbe sind, d.h. daß die *intra*molekularen Potentiale wesentlich steifer sind, als die *inter*molekularen, muß man $\epsilon_{\mathrm{BW}}, \epsilon_{\mathrm{BL}} \gg \epsilon_{\mathrm{LJ}}$ beachten.
Mit $\epsilon_{\mathrm{BL}} := 100$ ist dies erfüllt, denn die Bondlängen bleiben auch bei hohen Temperaturen stets nahe ihrer Gleichgewichtslänge, so daß die Wahl der maximalen Auslenkung $d_{\mathrm{BL}} := 0.2$ nur von geringem Einfluß ist.
Für das Bondwinkelpotential kann man eine Abschätzung für ϵ_{BW} durch den Vergleich des mittleren Bondwinkels $\langle \cos\theta \rangle$ bei einer bestimmten Temperatur mit dem Erwartungswert des Winkels im Rigby-Roe-Modell [9][58] gewinnen. Man erhält dabei $\epsilon_{\mathrm{BW}} \approx 4.7$
Um mit älteren Simulationen von F.M. Haas [25] konsistent zu bleiben, wurde weiterhin mit $\epsilon_{\mathrm{BW}} := 10$ gearbeitet.
Im folgenden bezeichne n die Zahl der Ketten, l deren Länge, d.h. die Zahl effektiver Monomere einschließlich Kopfgruppe und $N = n \cdot l$ die Gesamtzahl der Monomere.

2.3 System-Hamiltonian

An dem beschriebenen Modell wurden Monte Carlo-Simulationen im $N\Pi T$-Ensemble durchgeführt, d.h. die Teilchenzahl N, der Spreizdruck Π und die Temperatur T waren jeweils konstant.

Um die Zahl der simulierten Moleküle wegen des Rechenaufwands gering zu halten, aber trotzdem eine ausgedehnte Monolage zu simulieren, wurden innerhalb der Grenzfläche[3] periodische Randbedingungen angewandt, wodurch die Abbilder der Simulationsbox die Ebene vollständig bedecken. Untersuchungen einer einzelnen, freistehenden Gruppe von Amphiphilen ohne periodische Randbedingungen findet man z.B. in den Molekulardynamik-Simulationen von *Callaway et al.* [13].

Die Bewegung der Ketten erfolgte nach einem elementaren „particle displacement"-Algorithmus. Dabei wird zufällig[4] ein Monomer ausgewählt, für dieses wird eine zufällige Verschiebung vorgeschlagen und mit einem Metropoliskriterium [47] über die Akzeptanz oder Ablehnung der neuen Teilchenposition entschieden.

Neben diesen *Monomer-Moves*, bei denen sich gegenüber dem vorherigen Zustand nur ein einziges Teilchen bewegt, sind im $N\Pi T$-Ensemble, im Gegensatz zum NAT-Ensemble, auch Moves[5] nötig, welche die Abmessungen der Simulationsbox verändern (siehe auch Abschnitt 3.2.1).
Für den größten Teil der Simulationen wurde eine *monokline* Box verwendet, d.h. sie hat die Form eines Prismas mit einem Parallelogramm als Grundfläche. Dieses Parallelogramm wird durch die Seitenlängen L_x, L_y sowie den Scherwinkel α beschrieben und hat eine Fläche von $A = L_x L_y \sin\alpha$.
Es wurden daher drei verschiedene „Flächenmoves"[6] implementiert, je einer für L_x, L_y und α, was dem entsprechend zu einer Streckung/Stauchung der Box in einer der beiden Richtungen oder zu einer Scherung der ganzen Box führt.
Ein *Monte Carlo-Schritt* MCS setzt sich dann aus N Monomer-Moves und je einem Volumen-Move für L_x, L_y und α zusammen[7].
Da sich bei einem Volumen-Move die Abstände aller Teilchenpaare in der Box ändern, sind sämtliche Wechselwirkungen neu zu berechnen, wodurch sich der Rechenaufwand gegenüber NAT etwa verdoppelt.

Der **effektive Hamiltonian** H_{eff}, der den $N\Pi T$-Simulationen zugrunde liegt, lautet

$$H_{eff} = V(\underline{r}) + \Pi A - N k_B T \ln A \quad \text{mit der Teilchenkonfiguration } \underline{r} \tag{2.3.6}$$

und ergibt sich aus dem NAT-Hamiltionan $H_{NAT} = V(\underline{r})$ durch eine Laplace-Transformation (Anhang A). Das Gesamtpotential $V(\underline{r})$ beinhaltet die Beiträge der Bondlängen-, Bondwinkel-, Lennard-Jones- und Soft Core-Potentiale.

[3] Da nur eine Monolage betrachtet wird, gelten senkrecht zur Grenzfläche (d.h. in z-Richtung) keine periodischen Randbedingungen.

[4] Die Pseudo-Zufallszahlen wurden mit dem R250-Zufallsgenerator [38][69] erzeugt.

[5] Ein deutscher Ausdruck wäre „Bewegungsversuche".

[6] Im weiteren wird hierfür die übliche Bezeichnung „Volumen-Move" verwendet.

[7] Ob Volumen-Moves so häufig sein müssen bzw. ob man sie zu *einem* Move zusammenfassen könnte, der L_x, L_y und α zugleich ändert, wurde nicht untersucht.

2.4 Wesentliche Observablen

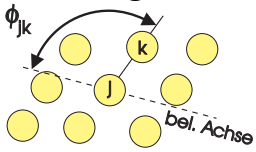
Dem von außen vorgegebenen, skalaren Druck Π^{ext} kann man den **Tensor des inneren Drucks** $\underline{\Pi}^{int}$ gegenüberstellen, der nach dem Virialtheorem [3] berechnet wird:

$$A \cdot \underline{\Pi}^{int} = A \cdot \begin{pmatrix} \Pi_{xx} & \Pi_{xy} \\ \Pi_{xy} & \Pi_{yy} \end{pmatrix} = \langle \sum_{i=1}^{N} \vec{r}_i \otimes \vec{F}_i \rangle + N k_B T \cdot \underline{1} \tag{2.4.7}$$

$\{\vec{r}_i\}$ sind darin die Koordinaten der N Monomere, $\{\vec{F}_i\}$ die darauf wirkenden Kräfte und $\underline{1}$ eine 2×2-Einheitsmatrix. Eine Ableitung dieser Formel und ihre Anwendung auf das hier verwendete Modell findet man in der Dissertation von F.M. Haas [25].
Wenn das simulierte System im Gleichgewicht ist, müssen sich der innere und äußere Druck kompensieren, also die Bedingung

$$\Pi^{ext} \cdot \underline{1} \stackrel{!}{=} \langle \underline{\Pi}^{int} \rangle \tag{2.4.8}$$

erfüllt sein. Im weiteren ist mit Π immer der von außen vorgegebene Druck Π^{ext} gemeint.
Der **hexagonale Ordnungsparameter** Ψ_6, der die Bondordnung wiedergibt, ist als



$$\Psi_6 := \left\langle \left| \frac{1}{6n} \sum_{j=1}^{n} \sum_{k=1}^{6} \exp(i6\phi_{jk}) \right|^2 \right\rangle \tag{2.4.9}$$

definiert, worin ϕ_{jk} der Winkel zwischen der Verbindungslinie der Köpfe j und k und einer beliebigen Referenzachse ist. Weil die exakte Bestimmung der nächsten Nachbarn für jede Kopfgruppe eine zeitaufwendige Voronoi-Konstruktion erfordern würde, wird hier immer über die sechs Köpfe mit dem geringsten Abstand summiert.

Da die Ketten in den LC-Phasen nahezu gestreckt vorliegen, lassen sich wesentliche Merkmale durch die Betrachtung ihrer End-zu-End-Vektoren wiedergeben. Sofern von diesen sog. „Moleküldirektoren“ $\vec{D}$ nur die Richtung, nicht aber die Länge benötigt wird, kann man diese auch als Hauptträgheitsachse zum kleinsten Trägheitsmoment der Kette definieren, wie es in einigen MD-Simulationen [36] praktiziert wird. In den vorliegenden Untersuchungen wurde zwischen beiden Definitionen kein relevanter Unterschied festgestellt.

Als Maß der **Neigungskorrelation** benachbarter Ketten dient die Größe

$$K_{NN} := \left\langle \frac{1}{6n} \sum_{j=1}^{n} \sum_{k=1}^{6} \frac{3}{2} \cos^2 \theta_{jk} - \frac{1}{2} \right\rangle . \tag{2.4.10}$$

θ_{jk} ist hierin der Winkel zwischen zwei benachbarten Moleküldirektoren. Die Nachbarschaftseigenschaft wird wie bei der Größe Ψ_6 gehandhabt.

Um die **Neigung der Ketten** zu quantifizieren betrachtet man primär den mittleren Winkel zwischen den Moleküldirektoren und der Flächennormalen.

$$\langle \cos\theta \rangle \;:=\; \left\langle \left[\frac{\vec{D} \cdot \vec{e}_z}{|\vec{D}|} \right] \right\rangle \tag{2.4.11}$$

Mit [...] sei die **Mittelung** über die Simulationsbox, d.h. über die Konfiguration ausgedrückt, und mit $\langle ... \rangle$ das thermodynamische Zeit- bzw. Ensemble-Mittel.
Diese Größe unterscheidet jedoch nicht zwischen kollektiver und unkorrelierter Neigung, weswegen man zusätzlich die Observable

$$R_{xy} \;:=\; \sqrt{\langle\, [x]^2 + [y]^2 \,\rangle} \qquad \text{mit } \vec{D} = (x, y, z)\,, \tag{2.4.12}$$

auswertet, die der Länge des in die Grenzfläche projizierten, mittleren Moleküldirektors entspricht. Im Falle unkorrelierter Neigung, wie in der LE-Phase, hat zwar $\langle \theta \rangle$ einen relativ großen Wert, R_{xy} verschwindet aber nahezu, da die Neigungsrichtungen der Ketten zufällig verteilt sind und sich über die Box weitgehend ausmitteln.
In kollektiv geneigten Phasen ist R_{xy} von Null verschieden, aber nicht auf Eins normiert. Dies erreicht man mit der Definition

$$O_{xy} \;:=\; \sqrt{\frac{\langle [x]^2 + [y]^2 \rangle}{\langle [x^2 + y^2] \rangle}} \;=\; \frac{R_{xy}}{\sqrt{\langle [x^2 + y^2] \rangle}}\,. \tag{2.4.13}$$

Zur Untersuchung von *finite size*-Effekten leitet man aus R_{xy} die **Ordnungsparameter-Kumulante 4. Ordnung**

$$g_4 \;:=\; \frac{\langle ([x]^2 + [y]^2)^2 \rangle}{\langle [x]^2 + [y]^2 \rangle^2} \tag{2.4.14}$$

ab, die in kollektiv geneigten Phasen der Wert $g_4 = 1$ hat und in ungeneigten oder unkorreliert geneigten Phasen auf $g_4 \approx 2$ ansteigt.

2.5 Strukturelle Information

2.5.1 Radiale Paarverteilungsfunktion

Eine im Zusammenhang mit statistischen Untersuchungen von Schmelzübergängen häufig benutzte Observable [56] ist die Paarverteilungsfunktion

$$g(\vec{r}) := \frac{V}{N}\frac{1}{N}\sum_{i\neq j}\left\langle \delta\left(\vec{r}-(\vec{R}_i-\vec{R}_j)\right)\right\rangle . \tag{2.5.15}$$

Da hier ein zweidimensionales System mit fluktuierender Fläche betrachtet wird, verwendet man

$$g(\vec{r}) = \frac{1}{n^2}\sum_{i\neq j}\left\langle A\cdot\delta(\vec{r}-\vec{R}_{ij})\right\rangle \tag{2.5.16}$$

bzw. die radiale Paarverteilungsfunktion[8] als Mittelwert über den Azimutwinkel φ:

$$g(r) := \frac{1}{2\pi}\int_0^{2\pi} g(r,\varphi)\mathrm{d}\varphi \tag{2.5.17}$$

Für den zweidimensionalen Abstand zwischen den Ketten wurden drei verschiedene Arten von Paaren betrachtet:

- die Kopfgruppen
- die Schwerpunkte der Ketten (deren Projektion in die Ebene)
- die Durchstoßpunkte mit der z=2–Ebene parallel zur Grenzfläche

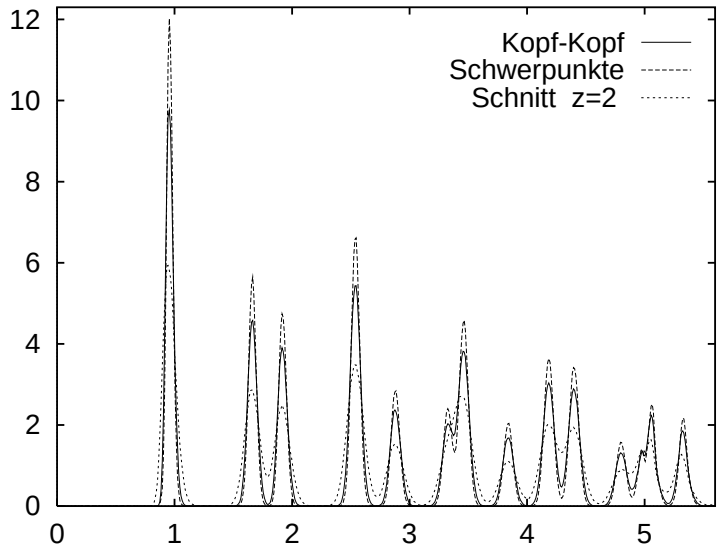


Abbildung 2.1: radiale Paarverteilungsfunktionen $g(r)$ für LC-U-Phase bei $\sigma_H = 1.1\sigma,\ \Pi = 100,\ T = 1$

[8] Aufgrund der periodischen Randbedingungen treten für Abstände, die im Bereich der Größe der Simulationsbox liegen, zusätzliche starke Korrelationen zwischen den Teilchen auf, weswegen man nur für Entfernungen r unterhalb dieser Grenze sinnvolle Werte erhält.

Obige Abbildung 2.1 zeigt den Verlauf der radialen Paarverteilungsfunktion eines Systems in der ungeneigten LC-Phase. Die Peaks stimmen in ihrer Lage und relativen Höhe genau mit denen eines hexagonalen Gitters überein, da durch die fehlende Neigung das Gitter nicht verzerrt wird. Zwischen den drei Kurven besteht kein qualitativer Unterschied, man erkennt deutliche Strukturierung bis in die 14. Nachbarschaftsschale. Darüber hinaus würden die periodischen Randbedingungen das Bild verfälschen.

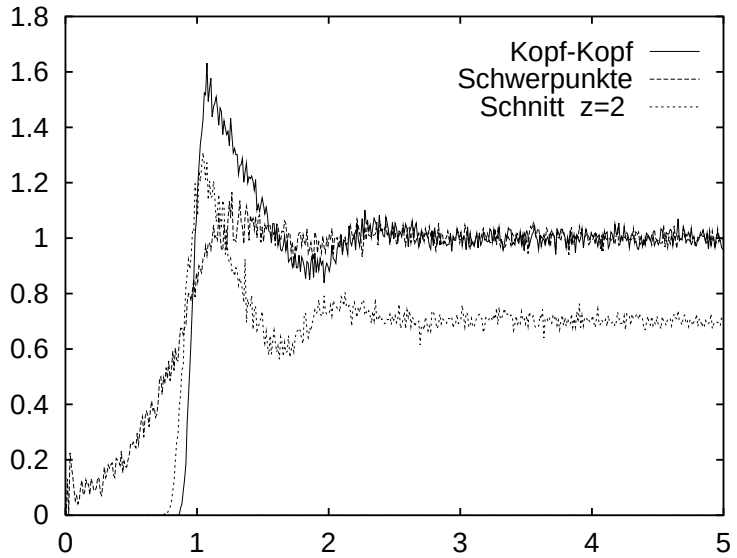


Abbildung 2.2: radiale Paarverteilungsfunktionen $g(r)$ für LE-Phase bei $\sigma_H = 1.1\sigma$, $\Pi = 1$, $T = 2$

In der LE-Phase ergibt sich der für Flüssigkeiten typische Verlauf. Der steile Abfall der Kurve der Kopfgruppen für $r \lesssim 1.1$ resultiert aus dem steilen Ansteig des Soft Core-Potentials für $r \lesssim \sigma_H$. Bei stark gekrümmten Ketten können ihre Schwerpunkte auch nahezu übereinander liegen. Die Funktion $g(r)$ verschwindet daher für die Schwerpunkte auch bei $r \gtrsim 0$ nicht völlig. Da die Schnittebene relativ hoch gelegt wurde um eine Aussage über die Kettenenden zu gewinnen, sind in dieser stark ungeordneten Phase einige Ketten soweit geneigt, daß sie gänzlich unter dieser Ebene bleiben und somit keinen Schnittpunkt liefern. Die entsprechende Kurve erreicht daher nicht ganz den erwarteten Wert $g(r) \approx 1$.

2.5.2 Strukturfaktor

Untersucht man Langmuir-Monoschichten mit Streuverfahren, z.B. Röntgenstreuung [5], erhält man die strukturelle Information in Form eines Strukturfaktors

$$S(\vec{q}) := \frac{1}{N_\infty}\left|\sum_{j=1}^{N_\infty} e^{i\vec{q}\cdot\vec{r}_j}\right|^2 . \tag{2.5.18}$$

Da die Monomerkoordinaten $\{\vec{r}_j\}$ periodischen Randbedingungen genügen, hat die unendlich ausgedehnte Monolage eine periodische Struktur deren Elementarzelle gerade die Simulationsbox ist[9].

$$\begin{array}{lll} \text{Basisvektoren} & : \vec{a}_x = \binom{L_x}{0}, & \vec{a}_y = \binom{L_y\cos\alpha}{L_y\sin\alpha} \\ \text{reziproke Basis} & : \vec{b}_x = \frac{2\pi}{L_x}\binom{1}{-\mathrm{ctg}\alpha}, & \vec{b}_y = \frac{2\pi}{L_y}\binom{0}{1/\sin\alpha} \end{array} \tag{2.5.19}$$

Wegen der bekannten Beziehung [6]

$$\sum_{\vec{R}} e^{i\vec{k}\cdot\vec{R}} = V_C \sum_{\vec{K}} \delta(\vec{k}-\vec{K}) \tag{2.5.20}$$

erhält man für $S(\vec{q})$ nur dann von Null verschiedene Werte, wenn die $\vec{q}$-Komponente in der Ebene der Grenzfläche ein ganzzahliges Vielfaches der reziproken Basisvektoren $\vec{b}_x, \vec{b}_y$ ist. Für die q_z-Komponente gilt keine derartige Beziehung, da in z-Richtung nur eine einzelne Molekülschicht betrachtet wird.
Die Basen im Orts- und im reziproken Raum hängen von L_x, L_y und α ab, welche im Verlauf der $N\Pi T$-Simulation variiert werden. Somit hat jede erzeugte Konfiguration ihr eigenes (reziprokes) Gitter, auch wenn diese häufig nur wenig voneinander abweichen.
Um dies bei der Mittelwert-Bildung zu berücksichtigen, wurde ein feinmaschiges Netz über die q_x-q_y-Ebene gelegt und die Beiträge in den einzelnen Feldern ähnlich einem Histogramm über die verschiedenen Konfigurationen aufsummiert. Da experimentell von einer Aparatur endlicher Auflösung auszugehen ist, und die Dauer der Messung ebenfalls zu einer Zeitmittelung führt, scheint die beschriebene Methode mit der experimentellen Situation gut verträglich zu sein.
Betrachtet man für den Impulübertrag $\vec{q}$ nur reziproke Gittervektoren, muß man nur noch über die N Teilchen in der Simulationsbox summieren, anstatt über alle N_∞ Teilchen der ausgedehnten Monolage. Bei dieser Summation wird nicht zwischen Kopfgruppen und Schwanzmonomeren unterschieden. Im Bild der Streutheorie geht man also von jeweils gleichen Formfaktoren für beide Teilchensorten aus.

[9] Diese Periodizität ist simulationsbedingt und hängt nicht mit einer etwaigen Periodizität der Kettenanordnung zusammen.

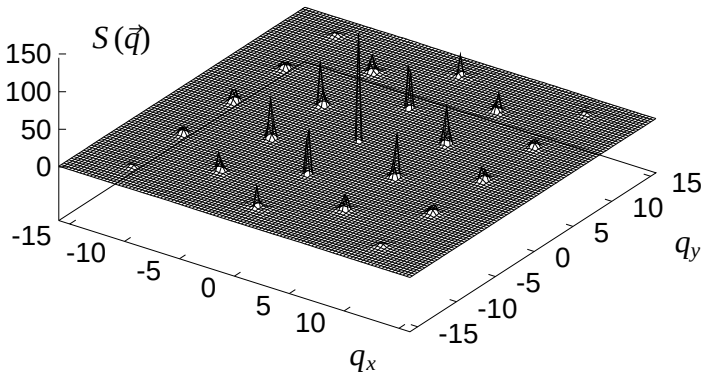


Abbildung 2.3
Strukturfaktor $S(\vec{q})$ in der xy-Ebene für die ungeneigte **LC-U**-Phase.
$\Pi = 50, \quad T = 2.0$

Der Strukturfaktor der LC-U-Phase zeigt deutliche Bragg-Peaks, deren Abstand der Gitterkonstanten des Kopfgruppengitters entspricht.
Wegen der relativ hohen Temperatur fallen die Peaks höherer Ordnung rasch ab.

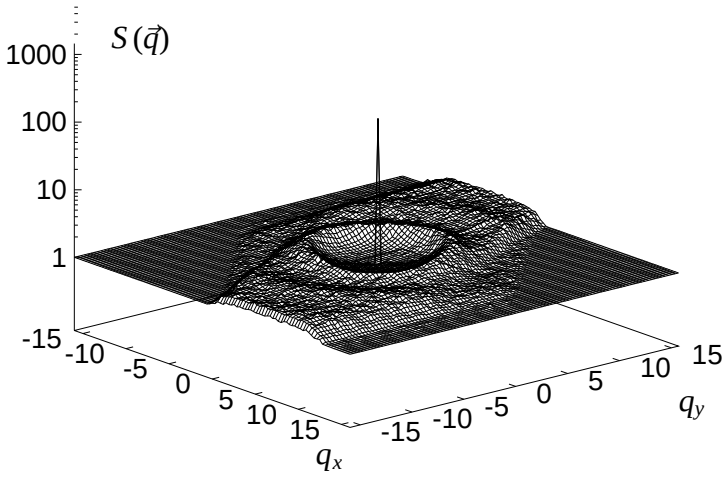


Abbildung 2.4
Strukturfaktor $S(\vec{q})$ in der xy-Ebene für die ungeordnete **LE**-Phase.
$\Pi = 10, \quad T = 2.5$

Die LE-Phase stellt sich im Strukturfaktor als Flüssigkeit dar.
Der Abfall an den Rändern rührt daher, daß die $\vec{q}$-Vektoren nur aus einem begrenzten Ausschschnitt des reziproken Gitters genommen wurden.

Bei kollektiv geneigten Phasen beobachtet man, daß die Bragg-Peaks, die bei ungeneigten Phasen in der $q_z = 0$-Ebene liegen, aus dieser herauswandern.
Die neue Ebene der Peaks ist genau um den Neigungswinkel ϑ der Ketten gegen die q_z-Achse verkippt. In der $q_z = 0$-Ebene findet man nur noch jene Strukturen, die auf der Schnittgeraden beider Ebenen liegen.
Ein Bild dieser Verhältnisse liefern Schnitte durch den $\vec{q}$-Raum in der q_x-q_z- bzw. q_y-q_z-Ebene. Hierbei ist zu beachten, daß das System als Monolage in der z-Richtung zwar nicht periodisch ist, die Bonds der Ketten aber annähernd gleich lang sind. Es liegt daher eine Anordnung von l äquidistanten Monomerschichten mit jeweils gleicher, hexagonaler Struktur vor. Dies gibt Anlaß zu Beugungsmaxima auch für q_z-Komponenten, die nicht mit der kollektiven Neigung erklärt werden können. Diese „Bondlängen-Peaks“ treten in Abständen $\Delta q_z \approx \frac{2\pi}{d_0 \cos\vartheta}$, d.h. ganzzahligen Vielfachen des reziproken Schichtabstands, von den „eigentlichen“ Maxima auf. Bei der Bestimmung des kollektiven Neigungswinkels ϑ aus dem Strukturfaktor ist daher eine Ver-

wechslung mit diesen „Bondlängen-Peaks“ zu vermeiden.

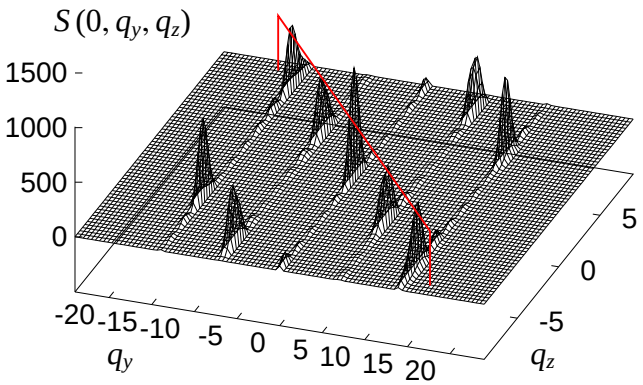


Abbildung 2.5:
Strukturfaktor $S(\vec{q})$ in der yz-Ebene für die geneigte **LC-NNN**-Phase.
$\Pi = 50, \quad T = 0.1$

Die Peaks 1. und 2. Ordnung sind aus der $q_z = 0$-Ebene verschoben, da die Ketten in Richtung der y-Achse geneigt sind.
Bei gleichem q_y findet man in Abständen $\Delta q_z \approx \frac{2\pi}{d_0 \cos\vartheta}$ Bondlängen-Peaks.

Die Breite dieser theoretischen Peaks, die durch keine experimentelle Auflösung begrenzt ist, wird in xy-Richtung durch die Fluktuation der Boxgeometrie und die individuellen Abweichung der Ketten von einer streng periodischen Anordnung vorgegeben, in z-Richtung jedoch durch die geringe Zahl l der Monomerlagen. Dies erklärt die Form der Reflexe im yz-Schnitt.

Während die eigentlichen Bragg-Peaks von der Anordnung der Ketten herrühren und somit eine reale Entsprechung haben, sind die Bondlängen-Reflexe eine Folge des Modells. In einer wirklichen Alkankette erfolgt die Streuung an den C- und H-Atomen und nicht an effektiven Monomeren. Insofern sind bei einem Vergleich mit gemessenen Strukturfaktoren die Bondlängen-Peaks außer acht zu lassen.

Für geneigte Phasen erweist es sich als praktisch, den Strukturfaktor $S(\vec{q})$ über die z-Komponente des Impulsübertrags zu integrieren,

$$S_{z-int}(\vec{q}_{\|}) := \int_{-\infty}^{\infty} \frac{1}{N} \left| \sum_{j=1}^{N} e^{i\vec{q}_{\|}\cdot\vec{r}_j} e^{i\vec{q}_z\cdot\vec{r}_j} \right|^2 \mathrm{d}q_z \,, \tag{2.5.21}$$

was experimentell dem Verzicht auf eine azimutale Auflösung entspricht. Man erhält damit Strukturen, die unabhängig von Neigung und Bondlängen-Peaks in der xy-Ebene liegen. Die Interpretation von Röntgenstrukturdaten geneigter Phasen ist z.B. bei *Kaganer* [34][35] beschrieben:

Da der vom einfallenden Strahl erfaßte Bereich der Monolage i.allg. so groß ist, daß er viele unabhängig orientierte Domänen der Phasen enthält, kann man dem Streusignal als *Pulver-Mittel* nur noch Information über $q_{\|} = \sqrt{q_x^2 + q_y^2}$ und q_z entnehmen.
Bildet man entsprechende Mittelwerte über die hier gezeigten Strukturfaktoren, kann man zwar $S(q_{\|}, q_z)$ direkt darstellen, ohne auf Schnitte durch den $\vec{q}$-Raum ausweichen zu müssen, die Darstellung verliert jedoch viel an Übersichtlichkeit.
Analog den experimentellen Abbildungen enthalten die folgenden azimutal-gemittelten Strukturfaktoren nur die Reflexe 0. und 1. Ordnung.

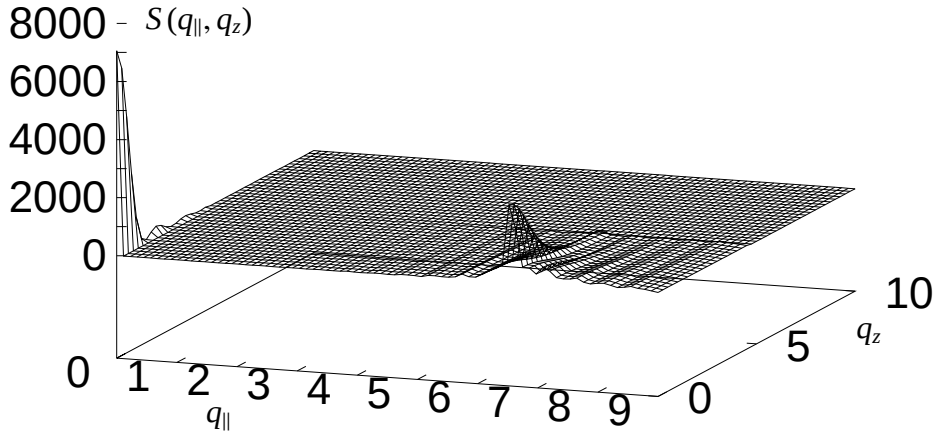


Abbildung 2.6: Strukturfaktor $S(q_{\|}, q_z)$ der ungeneigten LC-U-Phase; $T = 2.5, \Pi = 50$

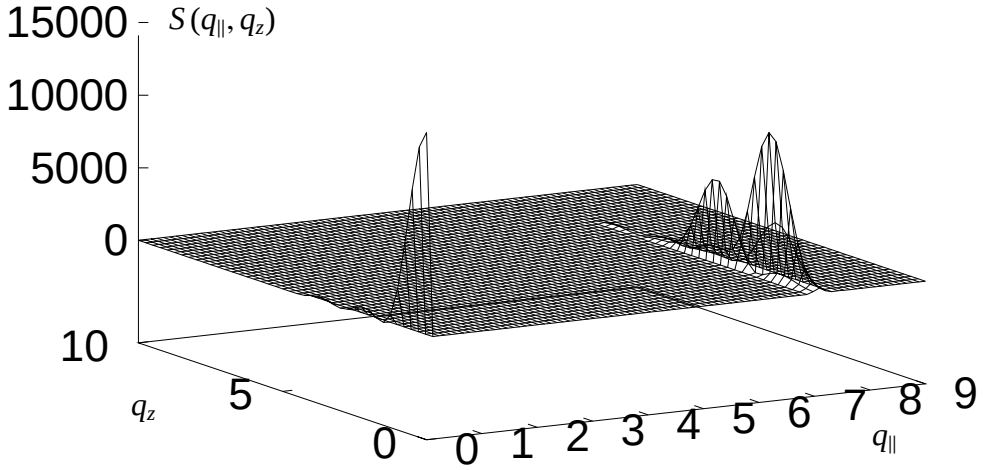


Abbildung 2.7: Strukturfaktor $S(q_{\|}, q_z)$ der NNN-geneigten LC-Phase; $T = 0.1, \Pi = 50$

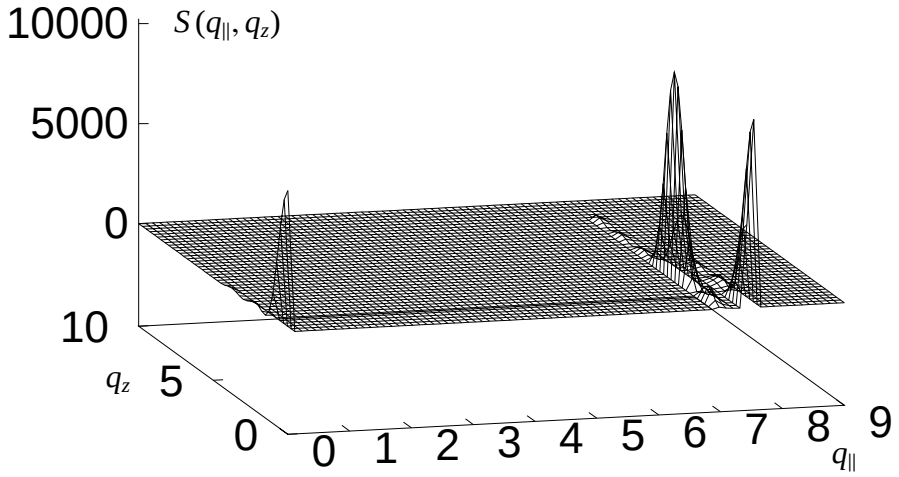


Abbildung 2.8: Strukturfaktor $S(q_{\|}, q_z)$ der NN-geneigten LC-Phase; $T = 0.1, \Pi = 1$

Die Bondlängen-Peaks wurden nach folgender Methode ausgeblendet:

Die Bragg-Peaks liegen im $\vec{q}$-Raum auf einer Ebene, die der Bedingung $\vec{q} \cdot \vec{D} \stackrel{!}{=} 0$ genügt, wobei $\vec{D}$ den mittleren Moleküldirektor bezeichne. Die Ebene steht also senkrecht auf der Längsachse der Ketten.
Die Bondlängen-Peaks treten in Abständen von $\Delta q_z \approx \frac{2\pi}{d_0 \cos\vartheta}$ davon auf.
Man erhält daher noch eine gute Wiedergabe, wenn man nur solche $\vec{q}$-Vektoren verwendet, die in q_z-Richtung um weniger als $\Delta q_z/2$ von der Bragg-Ebene entfernt sind.

Im Vergleich mit den gemessenen Strukturfaktoren bei *Kaganer* [34] ergibt sich gute qualitative Übereinstimmung:

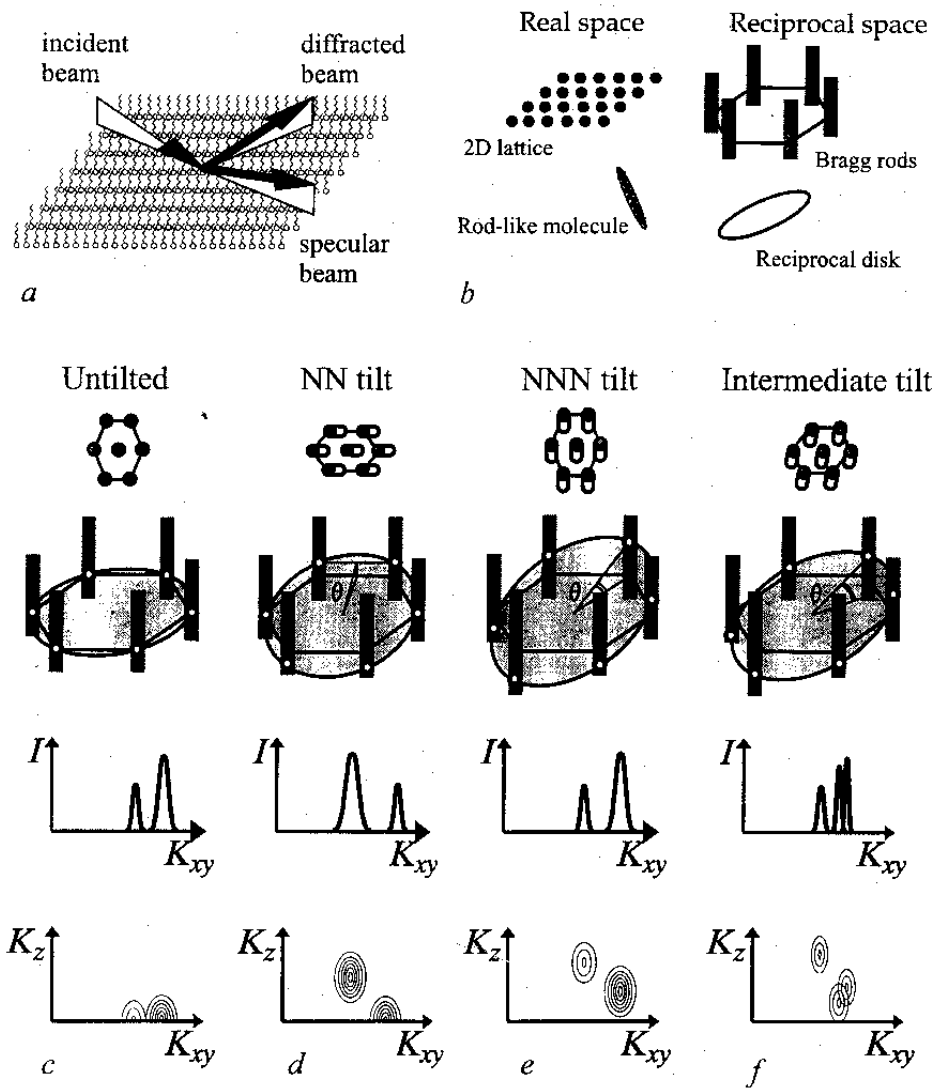


Abbildung 2.9: Schema der Röntgenbeugung bei streifendem Einfall:

(a) Entstehung des Beugungsbildes einer Monolage stabförmiger Moleküle

(b) Sichtweise im realen und im reziproken Raum

Charakteristische Beugungsbilder einer Monolage in (c) ungeneigter,

(d) NN-geneigter, (e) NNN-geneigter und (f) in Zwischenrichtung geneigter Phase

Die mittlere Reihe dieser Reproduktion zeigt das Kippen der Ebene der Bragg-Reflexe aufgrund der Neigung der Ketten, die unterste Reihe entspricht den umseitig abgebildeten Strukturfaktoren $S(\vec{q}_{\parallel}, q_z)$ im Pulvermittel.
Auf die rechte Spalte, in welcher der Fall einer Neigung in eine Zwischenrichtung diskutiert wird, wurde hier nicht eingegangen.

Kapitel 3

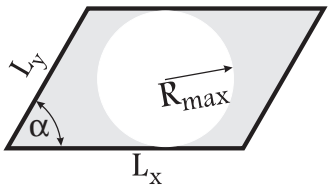
Simulationstechnik

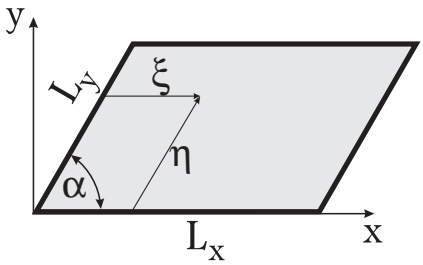
Die verwendeten Simulationsprogramme (siehe Anhang D) wurden aus dem Code von F.M. Haas [25] aufgebaut. Die beiden wesentlichen Erweiterungen sind zum einen die Zulassung verschiedener Potentiale zwischen den Köpfen (SoftCore) und den übrigen Monomeren (Lennard-Jones), zum anderen der Übergang von einer Simulationsbox mit rechteckiger Grundfläche zu einer scherbaren Box mit parallelogrammförmiger Grundfläche.

3.1 Notwendigkeit einer scherbaren Box

Die in dieser Arbeit präsentierten Simulationen wurden größtenteils mit einer Box durchgeführt, die nicht nur in den Seitenlängen L_x, L_y variabel ist, sondern auch im Scherwinkel α [1]. Während die *Minimum Image Convention* [4] bei einer rechteckigen Box trivial auszuwerten ist, gestaltet sich dies bei einer scherbaren Box deutlich komplizierter [3].

Mit einem schiefwinkligen, skalierten Koordinatensystem $(\xi, \eta) \in [0;1]^2$ statt des orthogonalen, absoluten (x, y) kann man den Aufwand auf ein praktikables Maß reduzieren, zumindest für Abstände d_{ij} die kürzer sind als der Radius R_{max} der größten Kugel, die der Box einbeschrieben werden kann.





$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} L_x & L_y \cos\alpha \\ 0 & L_y \sin\alpha \end{pmatrix} \begin{pmatrix} \xi \\ \eta \end{pmatrix} \tag{3.1.1}$$

[1] Dies entspricht dem 2-dimensionalen Fall der *Parrinello-Rahman*-Geometrie, in der die Simulationsbox ein beliebiges Parallelepiped ist [54][55].

Die *Minimum Image Convention* läßt sich nun wie im rechtwinkligen Fall auswerten:

$$\begin{aligned} &\Delta\xi_{ij} := (\xi_i - \xi_j) - \texttt{NINT}(\xi_i - \xi_j) \qquad \Delta\eta_{ij} := (\eta_i - \eta_j) - \texttt{NINT}(\eta_i - \eta_j) \\ &\texttt{NINT} : \text{Rundung auf } \textit{nearest integer} \\ &d_{ij}^2 := (L_x \cdot \Delta\xi_{ij})^2 + (L_y \cdot \Delta\eta_{ij})^2 + 2 \cdot L_x L_y \cos\alpha \cdot \Delta\xi_{ij} \cdot \Delta\eta_{ij} + (z_i - z_j)^2 \end{aligned} \tag{3.1.2}$$

Abstände die real größer sind als R_{max} werden nach dieser Formel zwar evtl. falsch angegeben, stets jedoch größer als R_{max}. Falls also R_{max} größer ist als der Abschneideradius r_{cut} des Lennard-Jones-Potentials, bleibt dessen Auswertung fehlerfrei. Da alle anderen Potentiale kürzere Reichweiten haben gilt dies für sie ebenso.
Daß dieser Aufwand notwendig ist zeigt sich an zwei Problemen, die bei Simulationen mit rechteckiger Box auftreten.

- Bei der Messung des Spreizdruck-Tensors $\underline{\Pi}^{int}$ erhält man nicht-verschwindende Scherkomponenten Π_{xy}.
- Die jeweils 6-fache Entartung der NN- bzw. NNN-Zustände der LC-Phase wird nicht korrekt reproduziert.

Dies sei anhand folgender Tabellen illustriert. Hierbei wurde, jeweils für rechteckige und scherbare Box, das System mit Ausrichtung φ_0 auf die sechs verschiedenen NN- und NNN-Positionen aufgesetzt und äquilibriert.

	φ_0		φ_{fin}	$\theta[°]$	Π^{int}	Π_{xy}	U/N	A/n
∠	0°	↦	0°	26.7	0.91	0.02	-5.388	0.891
□	0°	↦	0°	26.6	1.10	0.06	-5.388	0.890
∠	60°	↦	60°	26.6	1.00	0.02	-5.387	0.890
□	60°	↦	NNN	24.6	0.97	0.09	-5.377	0.878
∠	120°	↦	120°	26.6	1.00	0.00	-5.388	0.890
□	120°	↦	NNN	24.6	1.02	0.04	-5.377	0.878
∠	180°	↦	180°	26.7	0.99	-0.01	-5.388	0.891
□	180°	↦	180°	26.6	1.07	0.04	-5.377	0.890
∠	240°	↦	240°	26.6	1.05	-0.03	-5.387	0.890
□	240°	↦	NNN	24.6	1.01	-0.06	-5.377	0.878
∠	300°	↦	300°	26.6	1.03	-0.01	-5.387	0.890
□	300°	↦	NNN	24.6	0.99	-0.05	-5.377	0.878

NN-Richtungen bei Π=1.0 und T=0.1 für $\sigma_H = 1.1\sigma$:

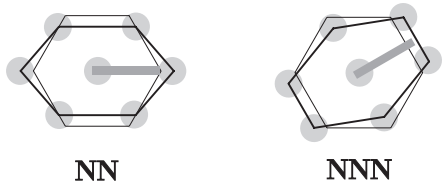
Während bei rechteckiger Box (□) das System in den Richtungen 0° und 180° verharrt und gut mit der scherbaren Box (∠) übereinstimmt, sind die anderen NN-Richtungen nicht stabil und gehen in NNN-Zustände über.
Dieses ist für die gewählten Werte von Π und T aber die instabile Phase.
Nur bei 0° und 180° liegt die Neigungsrichtung parallel zu den Kanten des Quaders.

	φ_0		φ_{fin}	$\theta[°]$	Π^{int}	Π_{xy}	U/N	A/n
∠	30°	↦	30°	20.6	49.97	0.01	-5.228	0.819
□	30°	↦	22°	20.3	50.03	**5.99**	-5.218	0.818
∠	90°	↦	90°	20.6	49.97	-0.05	-5.227	0.819
□	90°	↦	90°	20.6	50.06	0.09	-5.228	0.818
∠	150°	↦	150°	20.6	49.90	0.05	-5.228	0.819
□	150°	↦	159°	20.3	50.03	**-5.80**	-5.218	0.818
∠	210°	↦	210°	20.6	50.06	0.04	-5.228	0.818
□	210°	↦	202°	20.3	50.09	**5.94**	-5.218	0.818
∠	270°	↦	270°	20.6	50.06	-0.01	-5.228	0.818
□	270°	↦	270°	20.6	50.02	0.07	-5.227	0.818
∠	330°	↦	330°	20.6	49.90	-0.05	-5.229	0.819
□	330°	↦	330°	20.3	49.90	**-5.77**	-5.219	0.818

NNN-Richtungen bei Π=50.0 und T=0.1 für $\sigma_H = 1.1\sigma$:

Für die Neigungsrichtungen, die parallel zu den Quaderkanten liegen, d.h. 90° und 270°, entsprechen sich die Daten aus rechteckiger und scherbarer Simulation.
Die anderen Richtungen drehen sich in eine Zwischenrichtung, wobei sie große Scherdrücke aufbauen.

Da sich bei kollektiver Neigung der Ketten das ursprünglich hexagonale Kopfgruppengitter in Richtung der Neigung verzerrt, entsteht so ein *rechtwinklig-zentriertes* Gitter. Um die Neigung nicht ungewollt zu beeinflussen ist es notwendig, der Simulation bezüglich des Gitters größte Freiheit zu geben.



Dies ist mit einer scherbaren Box gewährleistet, während eine rechteckige Boxgeometrie nur solche rechteckig-zentrierten Gitter zuläßt, deren Achsen parallel zu den Boxkanten liegen.
Nahezu alle Ergebnisse in dieser Arbeit wurden daher mit scherbarer Geometrie gewonnen.

3.2 Kollektive Moves

Zur Beschleunigung der Simulation wurden neben einer optimierten Implementierung des konventionellen Algorithmus' auch alternative, kollektive Moves untersucht.

3.2.1 Translations-Moves

Die intramolekularen Wechselwirkungen, d.h. Bondlängen- und Bondwinkelpotentiale, sind sehr viel steifer als die intermolekulare Lennard-Jones-Wechselwirkung. Daher könnten Moves, welche die inneren Freiheitsgrade der Ketten unverändert lassen, eine relativ höhere Akzeptanz finden und die Simulation dadurch beschleunigen.
Die vorhandenen Monomer- und Volumen-Moves haben beide die Eigenschaft, sowohl die inter- als auch die intramolekularen Wechselwirkungen zu beeinflussen.
Moves, die Ketten jeweils als ganzes verschieben, ohne die innere Konfiguration zu verändern, würden diesen Nachteil umgehen. Solche *„Translations-Moves“* wurden sowohl als Einzelketten-Moves, als auch als alternative Volumen-Moves erprobt.

Bei einem Einzelketten-Move wird eine der n Ketten ausgewählt und dafür eine zufällige Verschiebung in der xy-Ebene vorgeschlagen. Über die Annahme dieser neuen Position wird mit einem Metropolis-Kriterium entschieden.

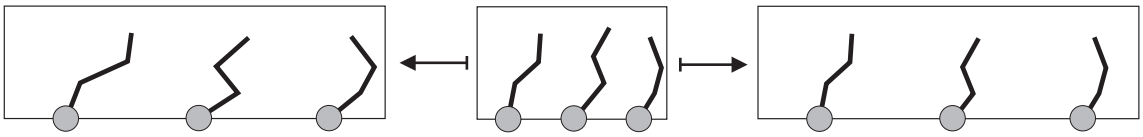
Die bisherigen Volumen-Moves entsprechen der Berechnung der $N\Pi T$-Zustandssumme aus Anhang A insofern, als bei der Streckung oder Stauchung der Simulationsbox alle Monomerkoordinaten $\{\vec{r}_i\}$ mit dem gleichen Faktor reskaliert werden[2].

[2] Damit widersprechen sie der Empfehlung von *Frenkel* und *Smit* [22], welche die im folgenden beschriebene Methode vorschlagen.

Im Falle einer Änderung der Boxlänge $L_x \mapsto L'_x$ also

$$\begin{pmatrix} r_{ix} \\ r_{iy} \\ r_{iz} \end{pmatrix} \mapsto \begin{pmatrix} r_{ix} \cdot L'_x/L_x \\ r_{iy} \\ r_{iz} \end{pmatrix} \quad \text{nun mit } 0 \leq r_{ix} < L'_x \ \ \forall i \tag{3.2.3}$$

Hierbei ändern sich zwar nicht die skalierten Relativkoordinaten $\{\vec{s}_i\}$, wohl aber die innermolekularen Abstände.



Reskalierung aller Koordinaten Verschiebung starrer Ketten

Im Unterschied dazu werden bei den neuen Volumen-Moves die Ketten als rigide Gebilde behandelt. Die Koordinaten der Kopfgruppen erfahren die übliche homogene Reskalierung und geben damit die Verschiebungsvektoren für alle anderen Monomere vor.

Dies stellt auch insoweit eine Abweichung vom bisherigen Vorgehen dar, als die Anzahl der unabhängig transformierten Koordinaten eine andere ist. Es werden nicht mehr $N = n \cdot l$ Monomerkoordinaten skaliert, sondern nur noch n Ketten, deren innere Freiheitsgrade unverändert bleiben.
Die Teilchenzahl, die im effektiven Hamiltonian im Term $-Nk_BT \ln A$ auftritt, ist nun nicht mehr die Zahl der Monomere N, sondern die Anzahl der Ketten n:

$$H_{eff}^{transl} := V(\underline{s}, L_x, L_y, \alpha) + \Pi A - nk_BT \ln A \tag{3.2.4}$$

Leider gelang es weder mit den Einzelketten- noch mit den alternativen Volumen-Moves die Geschwindigkeit der Simulation zu erhöhen.
Einerseits sind in den kondensierten Phasen hoher Dichte die Möglichkeiten zur Verschiebung der Ketten stark eingeschränkt, andererseits wird eine wesentlich schnellere Verarbeitung nicht erzielt, da zwar eine Neuberechnung der intramolekularen Potentiale entfällt, deren Anzahl aber verglichen mit den Lennard-Jones- und Soft Core-Wechselwirkungen relativ gering ist.

3.2.2 Configurational Biased Monte Carlo

Bewegt man wie bisher nur unabhängig einzelne Monomere einer Kette, sind großräumige Umlagerungen von der zufälligen Abfolge der akzeptierten Moves abhängig. Ein ganzes Molekül neu zu bilden, wie im *Configurational Biased Monte Carlo*-Verfahren [65][21], entspricht einem weiten Schritt im Konfigurationsraum und könnte daher große Vorteile bringen.

Die Vorstellung, die der Methode zugrunde liegt, besteht darin, eine lange Polymerkette in einer Schmelze an einer Stelle abzuschneiden und den entfernten Teil bis zur alten Länge neu wachsen zu lassen.
D. Frenkel et.al. [21] beschreiben eine Variante des CBMC-Algorithmus, die sich auf Kontinuumsmodelle von Ketten mit starken intramolekularen Wechselwirkungen anwenden läßt:

Erfolgte die Erzeugung der neuen Bonds „naiv“ zufällig, würde mit sehr großer Wahrscheinlichkeit eine Konfiguration vorgeschlagen, in der das neue Kettenstück sich selbst, den alten Kettenrumpf oder eine andere Kette durchdringt. Eine vorhandene Lennard-Jones- oder *excluded volume*-Wechselwirkung verbietet dies durch eine hohe „Energiestrafe“. Der Vorschlag wird daher verschwindendes Boltzmanngewicht haben und nur sehr wenige der neuen Kettenteile würden im MC-Prozeß akzeptiert.
Bei allen „intelligenteren“ Methoden, das neue Kettenstück bevorzugt energetisch günstig zu generieren, ist darauf zu achten, daß die *detailed balance* nicht verletzt wird, d.h. daß für zwei Zustände X und Y die Rate $K(X \mapsto Y)$ des Übergangs von X nach Y genauso groß ist, wie die des entgegengesetzten Übergangs

$$K(X \mapsto Y) \stackrel{!}{=} K(Y \mapsto X)\,. \tag{3.2.5}$$

Die Rate $K(X \mapsto Y)$ läßt sich in drei Faktoren zerlegen:

$$K(X \mapsto Y) \;=\; N_X \cdot P_Y \cdot A(X \mapsto Y) \tag{3.2.6}$$

Sie ist das Produkt der Wahrscheinlichkeit N_X den Zustand X vorliegen zu haben; Y als neuen Zustand vorzuschlagen ($\rightarrow\ P_Y$) falls man sich in X befindet; und der Akzeptanzwahrscheinlichkeit $A(X \mapsto Y)$ für diesen Vorschlag.
Geht man davon aus, daß die Zustände einer Boltzmannverteilung $N_X \sim \exp(-\beta U(X))$ gehorchen, sieht man, daß das Metropoliskriterium [47]

$$A(X \mapsto Y) \;=\; \min\left(1\,,\ \frac{P_X/\exp(-\beta U(X))}{P_Y/\exp(-\beta U(Y))}\right) \tag{3.2.7}$$

die Forderung des detaillierten Gleichgewichts erfüllt.
In „einfachen“ MC-Algorithmen würden die Zustände mit gleicher Wahrscheinlichkeit vorgeschlagen. Wegen $P_X = P_Y$ hängt die Akzeptanz dann nur noch vom Energieunterschied $\Delta U(X \mapsto Y)$ ab.
Bei dem hier untersuchten Verfahren geht es darum, für ein Polymer X der Länge l durch Abschneiden und Neu-wachsen eines Endes eine neue Konfiguration Y zu gewinnen.
Betrachtet man diesen Vorgang vorübergehend auf einem **Gittermodell**, gibt es für einen neu anzufügenden Bond eine endliche Zahl[3] k von Möglichkeiten.

[3] typischerweise $k = O(10)$

Die *Rosenbluth-Methode* [59] ein Polymer neu wachsen zu lassen sieht vor, aus diesen k Bonds, die an den i–1 langen Rumpf angefügt werden können, einen $1 \leq j \leq k$ gemäß seines Boltzmanngewichts $\exp(-\beta u_j^{(i)})$ auszuwählen. $u_j^{(i)}$ ist dabei die innere Energie der Bondposition j und enthält die Wechselwirkung mit den anderen Molekülen, sowie mit den $i-1$ Rumpfmonomeren, nicht aber die abgetrennten Monomere $i+1 \ldots l$.
Die Wahrscheinlichkeit der entstandenen Konfiguration beträgt

$$P_Y = \prod_{i=2}^{l} \frac{\exp(-\beta u_j^{(i)}(Y))}{Z_i(Y)} \qquad Z_i(Y) := \sum_{j=1}^{k} \exp(-\beta u_j^{(i)}(Y)) \tag{3.2.8}$$

Der zugehörige *Rosenbluth-Faktor* ist als

$$W_Y := \exp(-\beta u^{(1)}) \prod_{i=2}^{l} \frac{Z_i(Y)}{k} \tag{3.2.9}$$

definiert. Das Anfangsmonomer ($i = 1$) wird nicht verändert, weswegen $u^{(1)}(Y) = u^{(1)}(X)$ gilt.
Da die Wechselwirkung mit den noch abgetrennten Monomeren in die $u_j^{(i)}(Y)$ nicht eingeht, ergibt sich die Gesamtenergie der Kette Y zu $U(Y) = \sum_{i=1}^{l} u^{(i)}(Y)$. Hieraus folgt

$$P_Y / \exp(-\beta U(Y)) = \frac{1}{k^{l-1} W_Y} \qquad P_X / \exp(-\beta U(X)) = \frac{1}{k^{l-1} W_X}\,. \tag{3.2.10}$$

Wenn die Ketten mit ihrem korrekten Boltzmanngewicht in die Simulation eingehen sollen, müssen sie nach (3.2.7) mit einer Wahrscheinlichkeit

$$A(X \mapsto Y) = \min\left(1\,,\ \frac{W_Y}{W_X}\right), \tag{3.2.11}$$

akzeptiert werden, d.h. entsprechend ihrer Rosenbluth-Faktoren.
Das bis hierher beschriebene Verfahren ist in zwei Punkten zu modifizieren, will man es auf das untersuchte **Kontinuumsmodell** für Langmuir-Monoschichten anwenden:

- für einen neuen Bond stehen nicht k Möglichkeiten zu Verfügung, sondern ein (überabzählbares) Kontinuum;
- die innere Energie setzt sich einerseits aus Beiträgen der steifen Bondlängen- und Bondwinkelpotentiale zusammen, andererseits aus den weicheren Lennard-Jones- und Soft Core-Potentialen.
  Während erstere nur von den beiden Bondnachbarn abhängen, ist die Zahl der Wechselwirkungspartner bei letzteren sehr groß.

Der bei *Allen* und *Tildesley* [3] dargestellten *Continuous Configurational Biased Monte Carlo*-Methode folgend, werden die Potentiale in zwei Klassen zusammengefaßt:

*intra*molekulares Potential v := Bondlängen- + Bondwinkelpotential
*inter*molekulares Potential V_{ext} := Lennard-Jones- und Soft Core-Wechselwirkung mit den anderen Ketten, sowie mit dem bestehenden Kettenrumpf, ohne dem (noch) abgetrennten Teil

Auf eine Ableitung der folgenden Schritte sei hier verzichtet, sondern nur das tatsächliche Vorgehen erklärt.
Als Ausgangspunkt sei für die neue Konfiguration der Kette der i.te Bond zu generieren.

Schritt 1: Erzeugung einer Schar von k Versuchsbonds gemäß ihres „inneren Boltzmanngewichts“

Für jeden Bond j sind $(r_j, \vartheta_j, \varphi_j)$ zu bestimmen, deren Verteilung durch das *intra*molekulare Potential festgelegt wird:

$$P(r,\vartheta,\varphi) = \frac{e^{-\beta v(r,\vartheta,\varphi)}}{Z} = \frac{e^{-\beta V_{\mathrm{BL}}(r)} \cdot e^{-\beta V_{\mathrm{BW}}(\vartheta)}}{2\pi \int_0^\infty e^{-\beta V_{\mathrm{BL}}(r)} r^2 \mathrm{d}r \int_0^\pi e^{-\beta V_{\mathrm{BW}}(\vartheta)} \sin\vartheta \mathrm{d}\vartheta} \tag{3.2.12}$$

ϑ_j ist hierbei der Knickwinkel der mit dem vorangegangenen Bond $i-1$ gebildet wird.

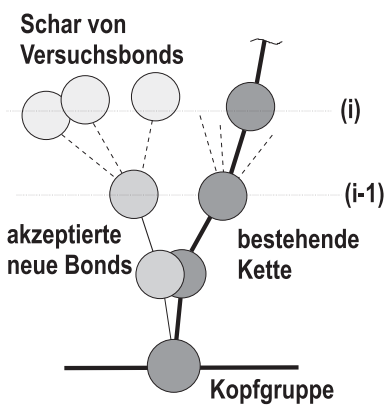
Schritt 2: Berechnung des „externen Rosenbluth-Gewichts“ der Schar

$$W_i^{trial} := \frac{1}{k} \sum_{j=1}^{k} e^{-\beta V_{ext}^{(i)trial}(j)} \tag{3.2.13}$$

trial deutet an, daß hier die unteren $i-1$ bereits erzeugten neuen Bonds eingesetzt werden, obwohl über die Annahme der neuen Konfiguration noch nicht endgültig entschieden wurde.

Schritt 3: Auswahl eines Bonds j^* aus der Schar

Die Wahl erfolgt entsprechend der „externen Boltzmanngewichte“ $\exp(-\beta V_{ext}^{(i)trial}(j))$ der Bonds. Auf diese Art wird schrittweise eine neue Konfiguration für die Kette aufgebaut, über deren Akzeptanz im Schritt 5 endgültig entschieden wird.



Schritt 4: „externes Rosenbluth-Gewicht“ der Schar auf der alten Konfiguration auswerten

Die alte Kette wird am $i-1$.ten Bond abgeschnitten und die Schar als i.te Bonds aufgesetzt. Da i.allg. der alte $i-1$.te Bond in eine andere Raumrichtung zeigt als der neue $i-1$.te Bond, die Gewichte der Bondwinkelpotentiale aber erhalten bleiben sollen, ist die ganze Schar entsprechend zu rotieren.

$$W_i^{old} := \frac{1}{k}\{e^{-\beta V_{ext}^{(i)old}(i_{old})} + \sum_{j=1;j\neq j^*}^{k} e^{-\beta V_{ext}^{(i)old}(j)}\} \tag{3.2.14}$$

Der i.te Bond der alten Konfiguration ersetzt den akzeptierten Bond j^* der neuen Versuchskonfiguration. *old* drückt aus, daß die unteren Monomere der alten Kette betrachtet werden.

Wiederholung der Schritte 1...4 bis volle Kettenlänge erreicht, d.h. bis $i = l$

Schritt 5: Akzeptanz-Entscheidung nach Gesamt-Rosenbluth-Gewicht

$$W_{ges} := \prod_{i=2}^{l} \frac{W_i^{trial}}{W_i^{old}} \qquad A(old \mapsto trial) = \min(1, W_{ges}) \tag{3.2.15}$$

Die Größe k der Schar erzeugter Versuchsbonds ist so zu wählen, daß einerseits eine ausreichend große Akzeptanzrate erreicht wird, andererseits keine Rechenzeit an unnötige Bonds vergeudet wird; sie wird im Bereich $k = O(10)$ liegen.
Im vollständigen Simulationsprogramm treten die CCBMC-Moves neben den „klassischen“ Monomer- und Volumen-Moves auf. Die Kette, welche von einem CCBMC-Move umgeformt werden soll, wird zufällig ausgewählt. Ebenso kann man die Zahl der neu zu erzeugenden Bonds frei wählen, d.h. die Stelle, an der die Kette abgeschnitten wird.

Eine wesentliche Beschleunigung der klassischen Simulation erreicht man durch Einführung einer *Verlet*-Nachbarschafts-Tabelle (siehe 3.3). Ihre Verwendung zusammen mit CCBMC-Moves erweist sich als problematisch.
Z.B. möchte man ausnutzen, daß alle Versuchsbonds einer Schar – sowohl auf der neuen, als auch auf der alten Konfiguration – dieselben Wechselwirkungspartner haben wie der tatsächlich bestehende Bond. Die Abstände der Versuchsbonds können insbesondere am Kettenende $i \lesssim l$ relativ groß werden, so daß der Radius der Verletkugel ebenfalls entsprechend groß werden muß, will man sicher sein, keine Wechselwirkungen zu übersehen. Je größer aber der Radius, umso weniger effizient ist die Verwendung einer Verlet-Tabelle.
Außerdem bewirkt die Akzeptanz eines CCBMC-Moves eine so starke Konfigurationsänderung, daß in der Regel sofort eine Aktualisierung der Verlet-Tabelle erfolgen muß, was relativ viel Rechenzeit kostet.

In den LC-Phasen liegen die Ketten in einer dichten, nahezu kristallinen Ordnung vor. Einzelne Ketten haben darin kaum Möglichkeiten sich individuell umzulagern. Wenn man die Schar der Versuchsbond nicht unverhältnismäßig vergrößern will, ist daher die Akzeptanz für CCBMC-Moves entsprechend gering $O(5\%)$.

Beide Gründe – die Schwierigkeiten mit der Verlet-Tabelle und die starke Ordnung der geneigten LC-Phasen – führen dazu, daß sich aus der Verwendung der CCBMC-Moves kein Vorteil erzielen ließ.
Da die Versuche mit CCBMC bereits in einem sehr frühen Stadium der vorliegenden Arbeit gemacht wurden, wurde darauf verzichtet, diesen komplexen Algorithmus beim Übergang von rechteckiger auf scherbare Box fortzuschreiben.

Es sollte jedoch bemerkt werden, daß alle Berechnungen auf Skalarrechnern erfolgt sind. Die Schar der Versuchsbonds müßte sich aber ausgesprochen gut zur Verarbeitung mit Vektorrechnern eignen, insbesondere die zeitintensive Auswertung der externen Rosenbluthgewichte (3.2.13 und 3.2.14).

3.3 Beschleunigung der Simulation

Auch ohne Verwendung kollektiver Moves gelang es, das Simulationsprogramm durch Einsparung überflüssiger Rechenoperationen um einen Faktor 8 zu beschleunigen.

Ein erster Faktor 2 konnte dadurch erzielt werden, daß alle 2-Teilchen-Wechselwirkungen im System ständig im Arbeitsspeicher des Rechners gehalten werden.
Bei einem vorgeschlagenen Move – gleichgültig ob für ein einzelnes Monomer oder für die ganze Box – müssen dann zur Bestimmung der Energiedifferenz nur noch die Wechselwirkungsenergien der neuen Konfiguration berechnet werden, nicht aber für die bestehende, da diese bereits zu einem früheren Zeitpunkt[4] einmal ausgewertet worden sind.
Leider erkauft man sich diesen Geschwindigkeitsvorteil mit einer drastischen Vergrößerung des Speicherbedarfs[5], wodurch man z.B. mit einem System von $n = 30^2$, $l = 7 \Rightarrow N = 6300$ Teilchen bereits an die Grenze der verfügbaren Rechner stößt.
Den größten Rechenzeitbedarf im untersuchten Modell hat die Auswertung der Lennard-Jones- und Soft Core-Potentiale. Bei der Optimierung kommt ihr daher große Beachtung zu. Um die Auswertung 6. ter und 12. ter Potenzen während der Simulation zu vermeiden, wurden diese Potentiale, einschließlich Abschneideradius und

[4]Durch Neuberechnung aller Energien und Vergleich mit den gespeicherten Werten läßt sich eine relativ breite Klasse möglicher Programmierfehler ausschließen.

[5]Man könnte die Wechselwirkungs-Energien auch analog der unten beschriebenen Verlet-Tabelle speichern, womit der Platzbedarf nur noch mit $c \cdot N^1$ anstatt mit N^2 stiege. Beim Zugriff wäre man dann auf die Verlet-Tabelle angewiesen und würde die Symmetrie $V(i, j) = V(j, i)$ verlieren.

Nullpunkts-Verschiebung, bezüglich des Abstandsquadrats tabellarisiert.
Hier ist darauf zu achten, das Tabellen-Raster fein genug zu wählen. Andernfalls tritt eine bleibende Abweichung zwischen dem externen Druck Π^{ext} und dem mittleren gemessenen Innendruck $\overline{\Pi}^{int}$ auf.

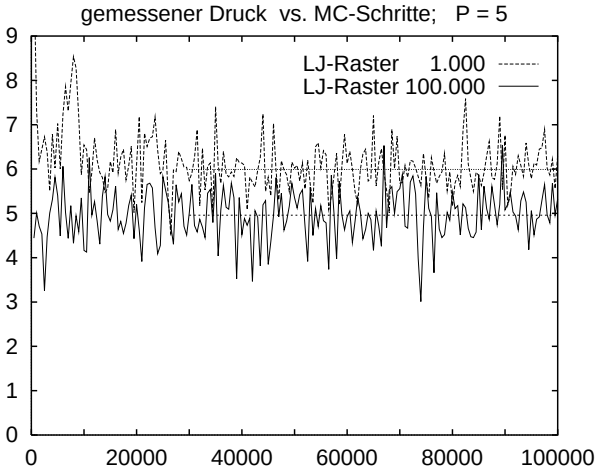
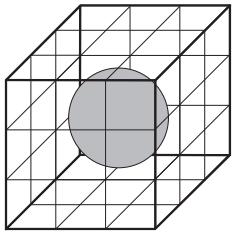


Abbildung 3.1: Innendruck Π^{int} aufgetragen gegen MC-Schritte für $\sigma_H = 1.2\sigma$, $\Pi^{ext} = 5$, $T = 0.1$ mit Lennard-Jones-Rastern von 1000 und 100000 Schritten.
Zusätzlich sind die Mittelwerte $\langle \Pi^{int} \rangle_{MCS}$ markiert.

Während bei 100000 Rasterstufen $\Pi^{int} \overset{!}{=} \Pi^{ext}$ gut erfüllt ist, ergibt sich bei 1000 Rasterstufen noch eine Abweichung von ca. 20%.

Die wesentliche Beschleunigung des Programms konnte dadurch erzielt werden, daß der vorhandene *Linked-Cell*-Algorithmus um eine *Verlet*-Tabelle erweitert wurde [4].
Ausgangspunkt beider Techniken ist die Tatsache, daß die Lennard-Jones- und Soft-Core-Potentiale bei einer endlichen, relativ geringen Reichweite $r_{cut} = 2\sigma$ abgeschnitten werden. Somit befinden sich alle tatsächlichen Wechselwirkungspartner $\{j\}$ eines Monomers i im Abstand $|\vec{r}_i - \vec{r}_j| \leq r_{cut}$. Teilt man die Simulationsbox in Quader der Kantenlänge[6] r_{cut}, so befindet sich jedes Monomer i eindeutig in einer dieser Zellen und die Monomere, mit denen es potentiell wechselwirken kann, liegen entweder in der selben Zelle, oder in einer der 26 unmittelbaren Nachbarzellen. Bei der Energieauswertung darf man sich auf diesen Bereich beschränken und vermeidet die Teilchenpaare, die zu weit voneinander entfernt sind, um einen Beitrag zu liefern.
Die praktische Implementierung dieser Zellen geschieht als verknüpfte Listen *linked cells*, d.h. wenn ein Teilchen die Zelle wechselt, wird es aus der einen Liste entfernt und an eine andere angehängt.
Nebenstehende Skizze illustriert, daß die Einsparung durch die Verwendung von *linked cells* bei weitem noch nicht optimal ist.



Der Raumbereich, in dem Wechselwirkungspartner gesucht werden, hat noch ein Volumen von ca. $(3r_{cut})^3$. Es würde jedoch genügen, um jedes Monomer eine Kugel mit Radius r_{cut} zu untersuchen. Unterstellt man eine etwa homogene Monomerdichte, könnte man die Zahl der Teilchenpaare damit um einen Faktor $\frac{(3r_{cut})^3}{4\pi r_{cut}^3/3} \approx 6.4$ verringern.

[6] Da die Zahl der Quader n_x, n_y, n_z ganzzahlig sein muß, werden sie i.allg. etwas größer sein: $L_x/n_x \gtrsim r_{cut}$.

Die Idee der *Verlet*-Tabelle besteht darin, für jedes Teilchen i die Indizes j seiner Nachbarteilchen einzutragen, die innerhalb einer Kugel mit Radius r_{Verlet} um i liegen.
Um nicht den gewonnenen Vorteil dadurch wieder einzubüßen, daß man die Verlet-Tabelle ständig, d.h. nach jedem akzeptierten Move, aktualisieren muß, wählt man ihren Radius etwas größer $r_{Verlet} := r_{cut} + r_{skin}$. Mit einer Wahl von $r_{skin} \approx 0.25 r_{cut}$ erreicht man, daß die Tabelle nur ca. alle 20 MonteCarlo-Schritte zu erneuern ist.
Da die erzielbare Geschwindigkeit empfindlich von Temperatur, Teilchendichte und r_{skin} [7] abhängt, ist es günstig, einen neuen Update erst zu veranlassen, wenn dies durch die Konfigurationsänderungen seit dem letzten Update nötig ist.
In Molekulardynamik-Simulationen werden in jedem Zeitschritt alle Kräfte zwischen den Teilchen neu berechnet. Wegen *actio=reactio* genügt es daher, für Teilchen i die Kräfte bezüglich der Partner $j > i$ auszuwerten. Entsprechend wird man in die Verlet-Tabelle nur Paare $j > i$ eintragen. In MonteCarlo-Simulationen im NVT-Ensemble sind nur Monomer-Moves nötig. Für diese müssen *alle* Wechselwirkungen des Teilchens i bewertet werden und die Tabelle enthält alle Paare mit $|\vec{r}_i - \vec{r}_j| \leq r_{Verlet}$.

MonteCarlo-Simulationen im $N\Pi T$-Ensemble stehen in gewisser Weise dazwischen, da es wegen der Monomer-Moves zwar nötig ist, alle Paare in der Tabelle zu haben, für die Volumen-Moves aber die Paare $j > i$ genügen, da alle Wechselwirkungen im System neu zu berechnen sind. Es ist deshalb vorteilhaft, die Verlet-Tabelle so zu organisieren, daß man die Paare nach $j > i$ bzw. $j < i$ unterscheidet.

Gegenüber dem reinen *linked cells*-Algorithmus ergab die *Verlet*-Tabelle eine Beschleunigung um etwa einen Faktor 4, was dem optimalen Wert von 6.4 schon recht nahe kommt. Hierbei erfolgt die Auswertung der Wechselwirkungen mit Hilfe der Verlet-Tabelle und deren Aktualisierung unter Benutzung der linked-cells-Struktur.

[7] r_{skin} wurde während der Simulation auf minimalen CPU-Zeit-Verbrauch geregelt.

Kapitel 4

Bestimmung der freien Energie

4.1 Thermodynamische Integration

Das thermodynamische Potential, das im $N\Pi T$-Ensemble über die Stabilität einer Phase gegenüber einer anderen bzw. über die Lage der Phasengrenzlinie entscheidet, ist die **Gibbs'sche freie Energie**

$$G(N,\Pi,T) = U - TS + \Pi A \qquad \Rightarrow \qquad \mathrm{d}G = A\mathrm{d}\Pi - S\,\mathrm{d}T \tag{4.1.1}$$

Da im Gegensatz z.B. zum Spreizdruck die Entropie während der Simulation nicht „gemessen" werden kann, scheitert sowohl die direkte Berechnung des Gibbs'schen Potentials, als auch die thermodynamische Integration $\int \mathrm{d}G$.
Stattdessen betrachtet man die Größe βG, denn es gilt

$$\begin{aligned} \mathrm{d}(\beta G) &= H\mathrm{d}\beta + \beta A\mathrm{d}\Pi \qquad \text{mit der Enthalpie } H = U + \Pi A \\ \Delta(\beta G) &= \beta_2 G_2 - \beta_1 G_1 = \int_\Gamma H\mathrm{d}\beta + \beta A\mathrm{d}\Pi \qquad (k_B = 1) \\ \Rightarrow \Delta(\beta G) &= \int_\Gamma \frac{A}{T}\mathrm{d}\Pi - \frac{H}{T^2}\mathrm{d}T \qquad \Gamma = \text{Weg im } \Pi - T\text{-Raum} \end{aligned} \tag{4.1.2}$$

Um mittels thermodynamischer Integration zu einer vollständig definierten Größe G zu gelangen, benötigt man einen Referenzzustand mit bekanntem G. Hierzu böte sich der Grundzustand $\lim T \to 0$ an. Leider ist dieser als Startwert für die Integration ungeeignet, da die Integranden bei $T = 0$ divergieren.
Als Ausweg wurde eine *harmonische Näherung* betrachtet, worin man sich auf kleine Auslenkungen aus dem Grundzustand beschränkt. Wie im folgenden Abschnitt gezeigt wird, gelingt es, den Hamiltonian des Systems in harmonischer Näherung auszuwerten. Diese ist bis zu Temperaturen von $T \lesssim 0.2$ recht genau erfüllt, so daß die Zustandssumme Z des Systems ebenfalls durch die harmonische Zustandssumme Z_{harm}

approximiert werden kann und man mit der Beziehung

$$G(\beta) \approx G_{harm}(\beta) = -\frac{1}{\beta} \ln Z_{harm}(\beta) = -\frac{1}{\beta} \ln \left(\underbrace{\int \ldots \int e^{-\beta H}}_{(4.4.37)} \right) \tag{4.1.3}$$

zu dem gewünschten Anfangswert der thermodynamischen Integration gelangt.

4.2 Harmonische Näherung

Der Zustand des simulierten Systems läßt sich durch die Angabe aller Monomerkoordinaten $\{\vec{r}_m | m = 1 \ldots n \cdot l\}$ und der Boxparameter $\{L_x, L_y, \alpha\}$ vollständig beschreiben.
Da sich die Monomere, mit Ausnahme der Köpfe, im 3-dimensionalen Raum bewegen, hat das System $F = (3(l-1)+2) \cdot n + 3$ Freiheitsgrade[1].
Jeder der f harmonischen[2] Freiheitsgrade trägt mit $k_B/2$ zur Wärmekapazität bei.

Daraus ergibt sich ein Kriterium für den Gültigkeitsbereich der Näherung:

$$H(N, \Pi, T) - H(N, \Pi, 0) \approx \frac{f}{2} k_B T \tag{4.2.4}$$

Ein Vergleich der simulierten Enthalpie zeigt gute Übereinstimmung mit dem erwarteten Verlauf $H(T) = H(0) + k_B T \cdot 2881/2$ bis mindestens $T = 0.2$.

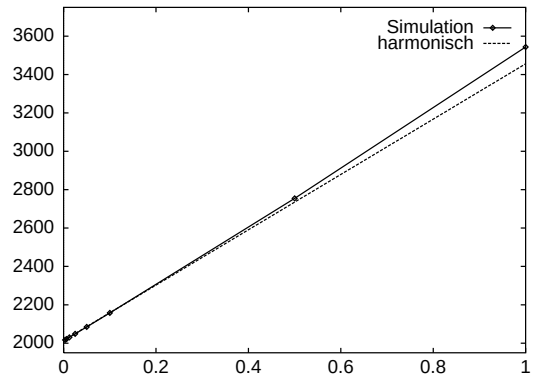


Abbildung 4.1: Enthalpie H vs. Temperatur T für $\sigma_H = 1.1\sigma$, $\Pi = 50$, LC-NNN-Phase

Faßt man die F Freiheitsgrade zu einem Vektor $\vec{U}$ zusammen, betrachtet man kleine Auslenkungen $\vec{u}$ aus dem Grundzustand $\vec{U}^0$:

$$\vec{u} := \vec{U} - \vec{U}^0 \qquad u_i \in \mathbb{R}, i = 1 \ldots F\,. \tag{4.2.5}$$

Für $T \mapsto 0$ geht der effektive Hamiltonian (2.3.6) in die Enthalpie über und man erhält

$$\begin{aligned} H(\{\vec{r}_m\}, L_x, L_y, \alpha) &= V(\{\vec{r}_m\}, L_x, L_y, \alpha) + \Pi A(L_x, L_y, \alpha) \\ \Rightarrow H(\vec{U}) &= V(\vec{U}) + \Pi A(\vec{U}). \end{aligned} \tag{4.2.6}$$

Entwickelt man diesen Ausdruck um den Grundzustand $\vec{U}^0$

$$H(\vec{U}) = H(\vec{U}^0) \;+\; \vec{u}\, \nabla_{\vec{u}} H|_{\vec{U}^0} \;+\; \tfrac{1}{2} \vec{u}^{\top} \left[\nabla_{\vec{u}} \otimes \nabla_{\vec{u}} H|_{\vec{U}^0} \right] \vec{u} \;+\; O(\vec{u}^3) \tag{4.2.7}$$

[1] Mit den üblichen Werten $n = 144$ und $l = 7$ ergibt sich $F = 2883$.
[2] Wegen zweier Translationsfreiheitsgrade gilt $f = F - 2$.

vernachlässigt Terme höherer Ordnung und nutzt, daß im Grundzustand der Gradient verschwinden muß, gelangt man zu

$$\begin{aligned} 2(H(\vec{U}) - H^0) &= \vec{u}^\top [\nabla_{\vec{u}} \otimes \nabla_{\vec{u}} H|_{\vec{U}^0}]\vec{u} \\ &= \sum_{ij=1}^{F} u_i \frac{\partial^2 H}{\partial u_i \partial u_j} u_j \quad =: \sum_{ij=1}^{F} u_i (\underline{H})_{ij} u_j \end{aligned} \tag{4.2.8}$$

mit der Kopplungsmatrix $\underline{H} \in \mathbb{R}^{F\times F}$, $\underline{H}^\top = \underline{H}$.
Zu deren Berechnung sind Ableitungen der Enthalpie der Typen

$$\frac{\partial^2 H}{\partial u_i \partial u_j} \quad : \quad \frac{\partial^2 H}{\partial x_m \partial z_n}, \quad \frac{\partial^2 H}{\partial y_m \partial \alpha}, \quad \frac{\partial^2 H}{\partial \alpha \partial L_x} \quad \text{usw.} \tag{4.2.9}$$

auszuwerten. Details dazu finden sich im Anhang B.
Um zu den Eigenmoden des Systems zu gelangen, wird diese Matrix diagonalisiert:

$$\underline{H}\,\vec{\epsilon}_s = \lambda_s \vec{\epsilon}_s \quad \text{mit } \vec{\epsilon}_s \in \mathbb{R}^F, \qquad \vec{\epsilon}_s \cdot \vec{\epsilon}_{s'} = \delta_{ss'} \qquad s, s' = 1...F \tag{4.2.10}$$

und $\lambda_s \in \mathbb{R}^+$ – falls man tatsächlich um den Grundzustand entwickelt hat. Die Eigenvektoren $\{\vec{\epsilon}_s\}$ bilden eine Orthonormalbasis des F-dimensionalen Zustandsraums der $\vec{u}$. Drückt man nun die Auslenkungen $\vec{u}$ in der Eigenbasis $\{\vec{\epsilon}_s\}$ aus

$$\vec{u} = \sum_{s=1}^{F} A_s \vec{\epsilon}_s \quad \in \mathbb{R}^F \qquad A_s = \vec{\epsilon}_s \cdot \vec{u} \quad \in \mathbb{R} \;\; \forall s = 1...F\,, \tag{4.2.11}$$

findet man für den harmonischen Hamiltonian

$$\begin{aligned} 2(H(\vec{u}) - H^0) &= \sum_{ss'} A_s A_{s'}\, \vec{\epsilon}_s^\top \underline{H}\, \vec{\epsilon}_{s'} = \sum_{ss'} A_s A_{s'}\, \lambda_s\, \delta_{s,s'} \\ H(\vec{u}) &= H^0 + \tfrac{1}{2} \sum_s A_s^2\, \lambda_s \end{aligned} \tag{4.2.12}$$

Unter den F Eigenmoden sind auch zwei Translationsmoden zu $\lambda = 0$ enthalten. Diese sind als solche leicht zu identifizieren und zu eliminieren, so daß man im folgenden nur $f = F - 2$ Eigenmoden verwendet.
Bei der bisher geschilderten Vorgehensweise muß eine reelle, symmetrische $F \times F$-Matrix diagonalisiert werden. Für die übliche Kettenzahl (144) und Kettenlänge (7) stößt man dabei mit einer 2883×2883-Matrix, was die Rechenzeit und den Speicherbedarf anbelangt, an die Grenze dessen, was auf den heutigen Maschinen als „gut handhabbar“ bezeichnet werden kann. Eine Verdoppelung der Kettenzahl würde den Speicherbedarf vervierfachen und die Rechenzeit verachtfachen[3], so daß ein deutlich schnellerer, weniger speicherintensiver Algorithmus dringend angezeigt wäre.
Für den NN-Grundzustand mit modulierter Neigung zu nächsten Nachbarn (vgl. Abschnitt 5.3.1) erscheint dies sehr aufwendig, für die unmodulierten Grundzustände sollte man sich aber deren strenge *Periodizität* zunutze machen.

[3] Der Zeitbedarf für die Diagonalisierung einer $N \times N$-Matrix wächst mit N^3 [51].

4.3 Teildiagonalisierung durch Ausnutzung der Periodizität

Die unmodulierten Grundzustände sind kettenperiodisch auf dem zweidimensionalen schiefwinkligen Gitter $\{\vec{R}\}$ der Kopfgruppen.
Abweichend von der Indizierung $i = 1...F$ aus dem vorangegangenen Abschnitt werden nun die Freiheitsgrade über die Zugehörigkeit zu einer bestimmten Kette $\vec{R}$ und über die Position und Richtung innerhalb der Kette $i = 1...(3l-1)$ angesprochen. Die $\vec{R} \in \mathcal{R}$ entsprechen den Gleichgewichtslagen der Köpfe in der Simulationsbox und es gilt

$$\mathcal{R} = \{\vec{R} | \vec{R} = n_x \vec{a}_x + n_y \vec{a}_y; 0 \leq n_x, n_y < \sqrt{n}\} \qquad \vec{a}_x = \tfrac{L_x}{\sqrt{n}}\binom{1}{0} \qquad \vec{a}_y = \tfrac{L_y}{\sqrt{n}}\binom{\cos\alpha}{\sin\alpha}. \quad (4.3.13)$$

Zum Gitter $\{\vec{R}\}$ existiert auch ein reziprokes Gitter $\{\vec{K}\}$ mit

$$\vec{a}_\alpha \cdot \vec{b}_\beta = 2\pi \cdot \delta_{\alpha,\beta} \quad \alpha, \beta \in \{x, y\} \qquad \vec{b}_x = 2\pi \tfrac{\sqrt{n}}{L_x}\begin{pmatrix} 1 \\ -\cot\alpha \end{pmatrix} \qquad \vec{b}_y = 2\pi \tfrac{\sqrt{n}}{L_y}\begin{pmatrix} 0 \\ 1/\sin\alpha \end{pmatrix}. \quad (4.3.14)$$

Die Boxparameter, die bisher die Auslenkungen $u_i, i = F-2...F$ bildeten, werden nun zur Unterscheidung mit dem Vektor $\vec{B} := (L_x - L_x^0, L_y - L_y^0, \alpha - \alpha^0)$ bezeichnet.
Der Hamiltonian in harmonischer Näherung (4.2.8) lautet jetzt

$$\begin{aligned} 2(H - H^0) = & \sum_{i,j=1}^{3l-1} \sum_{\vec{R},\vec{R}' \in \mathcal{R}} u_i(\vec{R}) \frac{\partial^2 H}{\partial u_i(\vec{R}) \partial u_j(\vec{R}')} u_j(\vec{R}') \; + \; \sum_{i,j=1}^{3} B_i \frac{\partial^2 H}{\partial B_i \partial B_j} B_j \\ & + \; 2 \cdot \sum_{i=1}^{3l-1} \sum_{\vec{R} \in \mathcal{R}} \sum_{j=1}^{3} u_i(\vec{R}) \frac{\partial^2 H}{\partial u_i(\vec{R}) \partial B_j} B_j \end{aligned} \quad (4.3.15)$$

In diesem Ausdruck treten Funktionen auf, die auf diskreten Punkten $\vec{R}$ definiert sind und den periodischen Randbedingungen $f(\vec{R} + \sqrt{n}\vec{R}') = f(\vec{R}) \;\; \forall \vec{R}, \vec{R}' \in \mathcal{R}$ gehorchen. Über deren Fouriertransformation weiß man [6]:

$$\begin{aligned} & f(\vec{R}) = \tfrac{1}{\sqrt{n}} \sum_{\vec{k} \in \mathcal{K}} \tilde{f}(\vec{k}) e^{i\vec{k}\cdot\vec{R}} \qquad \tilde{f}(\vec{k}) = \tfrac{1}{\sqrt{n}} \sum_{\vec{R} \in \mathcal{R}} f(\vec{R}) e^{-i\vec{k}\cdot\vec{R}} \\ & \text{mit} \quad \mathcal{K} = \{\vec{k} \,|\, \vec{k} = \tfrac{n_x}{\sqrt{n}}\vec{b}_x + \tfrac{n_y}{\sqrt{n}}\vec{b}_y; \; 0 \leq n_x, n_y < \sqrt{n}\} \end{aligned} \quad (4.3.16)$$

und es gelten die bekannten Beziehungen

$$\tfrac{1}{n} \sum_{\vec{k} \in \mathcal{K}} e^{i\vec{k}\cdot\vec{R}} = \delta_{\vec{R},\vec{0}} \qquad \tfrac{1}{n} \sum_{\vec{R} \in \mathcal{R}} e^{i\vec{k}\cdot\vec{R}} = \delta_{\vec{k},\vec{0}}\,. \quad (4.3.17)$$

Außerdem gilt

$$\begin{aligned} & \forall \vec{k} \in \mathcal{K} \;\text{ mit }\; -\vec{k} \notin \mathcal{K} \quad \exists \vec{k}' \in \mathcal{K} \;\text{ so daß }\; \sqrt{n}\vec{k}' - \vec{k} \in \mathcal{K} \\ & \forall \vec{R} \in \mathcal{R} \;\text{ mit }\; -\vec{R} \notin \mathcal{R} \quad \exists \vec{R}' \in \mathcal{R} \;\text{ so daß }\; \sqrt{n}\vec{R}' - \vec{R} \in \mathcal{R}\,. \end{aligned} \quad (4.3.18)$$

Falls also im folgenden Vektoren $-\vec{R} \notin \mathcal{R}$ oder $-\vec{k} \notin \mathcal{K}$ auftreten, sind jeweils deren Repräsentanten $\sqrt{n}\vec{R}' - \vec{R} \in \mathcal{R}$ bzw. $\sqrt{n}\vec{k}' - \vec{k} \in \mathcal{K}$ gemeint.
Für die Auslenkung der Monomere aus ihren Ruhelagen erhält man die Transformation

$$u_i(\vec{R}) = \tfrac{1}{\sqrt{n}} \sum_{\vec{k}\in\mathcal{K}} \tilde{u}_i(\vec{k}) e^{i\vec{k}\cdot\vec{R}} \qquad \tilde{u}_i(\vec{k}) = \tfrac{1}{\sqrt{n}} \sum_{\vec{R}\in\mathcal{R}} e^{-i\vec{k}\cdot\vec{R}} \qquad u_i(\vec{R}) \in \mathbb{R} \;\; \tilde{u}_i(\vec{k}) \in \mathbb{C} \tag{4.3.19}$$

und, unter Ausnutzung ihrer Periodizität

$$\frac{\partial^2 H}{\partial u_i(\vec{R}) \partial u_j(\vec{R}')} =: H_{ij}(\vec{R}, \vec{R}') = H_{ij}(\vec{0}, \vec{R}' - \vec{R}) \quad \forall i, j \;\; \forall \vec{R}, \vec{R}' \in \mathcal{R}\,, \tag{4.3.20}$$

für die Kopplungskonstanten

$$\begin{aligned} \tilde{H}_{ij}(\vec{k}, \vec{k}') &= \tfrac{1}{\sqrt{n}} \sum_{\vec{R}\in\mathcal{R}} \tfrac{1}{\sqrt{n}} \sum_{\vec{R}'\in\mathcal{R}} e^{-i\vec{k}\cdot\vec{R}} H_{ij}(\vec{R}, \vec{R}')\, e^{-i\vec{k}'\cdot\vec{R}'} \\ &= \sum_{\vec{R}''\in\mathcal{R}} H_{ij}(\vec{0}, \vec{R}'') \tfrac{1}{n} \sum_{\vec{R}\in\mathcal{R}} e^{-i(\vec{k}+\vec{k}')\cdot\vec{R}} \\ &= \delta_{-\vec{k},\vec{k}'} \tilde{H}_{ij}(\vec{k}) \qquad \text{mit} \quad \tilde{H}_{ij}(\vec{k}) := \sum_{\vec{R}\in\mathcal{R}} H_{ij}(\vec{0}, \vec{R}) e^{i\vec{k}\cdot\vec{R}} \\ H_{ij}(\vec{R}, \vec{R}') &= \tfrac{1}{n} \sum_{\vec{k}''\in\mathcal{K}} \tilde{H}_{ij}(\vec{k}'') e^{i\vec{k}''\cdot(\vec{R}-\vec{R}')}\,. \end{aligned} \tag{4.3.21}$$

Aus der Symmetrie $H_{ij}(\vec{R}, \vec{R}') = H_{ji}(\vec{R}', \vec{R})$ folgt die Hermitezität $\tilde{H}_{ij}(\vec{k}) = \tilde{H}^*_{ji}(\vec{k})$ der dynamischen Matrix $\underline{\tilde{H}}(\vec{k}) \in \mathbb{C}^{(3l-1)\times(3l-1)}$.
Die gemischten Ableitungen hängen wegen der Periodizität des Zustands, um den herum entwickelt wurde, nicht von $\vec{R}$ ab, und man findet

$$E_{ij} := \frac{\partial^2 H}{\partial u_i(\vec{R}) \partial B_j} \;\; \forall \vec{R} \quad \Rightarrow \quad \tilde{E}_{ij}(\vec{k}) = \sqrt{n}\, E_{ij}\, \delta_{\vec{k},\vec{0}} \tag{4.3.22}$$

Neben der Matrix $\underline{E} \in \mathbb{R}^{(3l-1)\times 3}$ definiert man noch $F_{ij} := \partial^2 H / \partial B_i \partial B_j \Rightarrow \underline{F} \in \mathbb{R}^{3\times 3}$.
Setzt man alle Fouriertransformationen in (4.3.15) ein, ergibt sich

$$\begin{aligned} 2(H - H^0) = & \sum_{\vec{k},\vec{k}',\vec{k}''} \sum_{\vec{R},\vec{R}'} \sum_{ij} \tfrac{1}{\sqrt{n}} \tilde{u}_i(\vec{k}) e^{i\vec{k}\cdot\vec{R}} \tfrac{1}{n} e^{i\vec{k}''\cdot(\vec{R}-\vec{R}')} \tilde{H}_{ij}(\vec{k}'') \tfrac{1}{\sqrt{n}} \tilde{u}_j(\vec{k}') e^{i\vec{k}'\cdot\vec{R}'} \\ & + \sum_{ij} B_i F_{ij} B_j \;\; + \;\; 2 \sum_{\vec{k}} \sum_{\vec{R}} \sum_{ij} \tfrac{1}{\sqrt{n}} \tilde{u}_i(\vec{k}) e^{i\vec{k}\cdot\vec{R}} E_{ij} B_j\,. \end{aligned} \tag{4.3.23}$$

Unter Ausnutzung von

$$\sum_{\vec{R}} e^{i(\vec{k}+\vec{k}'')\cdot\vec{R}} = n\delta_{-\vec{k},\vec{k}''} \qquad \sum_{\vec{R}'} e^{i(\vec{k}'-\vec{k}'')\cdot\vec{R}} = n\delta_{-\vec{k}',\vec{k}''} \qquad \sum_{\vec{R}} e^{i\vec{k}\cdot\vec{R}} = n\delta_{-\vec{k},\vec{0}} \tag{4.3.24}$$

vereinfacht sich dies zu

$$2(H-H^0) = \sum_{\vec{k}''}\sum_{ij} \tilde{u}_i(-\vec{k}'')\tilde{H}_{ij}(\vec{k}'')\tilde{u}_j(\vec{k}'') \;+\; 2\sqrt{n}\sum_{ij}\tilde{u}_i(\vec{0})E_{ij}B_j \;+\; \sum_{ij} B_i F_{ij} B_j\,. \tag{4.3.25}$$

Da die Monomerauslenkungen reellwertig sind, gilt

$$u_i(\vec{R}) = \tfrac{1}{\sqrt{n}}\sum_{\vec{k}} \tilde{u}_i(-\vec{k})e^{-i\vec{k}\cdot\vec{R}} \;=\; u_i^*(\vec{R}) = \tfrac{1}{\sqrt{n}}\sum_{\vec{k}} \tilde{u}_i^*(\vec{k})e^{-i\vec{k}\cdot\vec{R}} \;\Rightarrow\; \tilde{u}_i(-\vec{k}) = \tilde{u}_i^*(\vec{k})\,. \tag{4.3.26}$$

Die Terme B_j, E_{ij}, F_{ij} treten nicht im Zusammenhang mit $\tilde{u}_i(\vec{k})$ für $\vec{k} \neq 0$ auf.
Man kann daher folgende Umgruppierung vornehmen

$$2(H-H^0) = \begin{pmatrix} \vdots \\ \tilde{u}_i(\vec{0}) \\ \vdots \\ B_i \end{pmatrix}^\dagger \begin{pmatrix} \underline{\tilde{H}}(\vec{0}) & \sqrt{n}\underline{E} \\ \sqrt{n}\underline{E}^\top & \underline{F} \end{pmatrix} \begin{pmatrix} \vdots \\ \tilde{u}_j(\vec{0}) \\ \vdots \\ B_j \end{pmatrix} + \sum_{\vec{k}\neq\vec{0}} \begin{pmatrix} \vdots \\ \tilde{u}_i(\vec{k}) \\ \vdots \end{pmatrix}^\dagger \underline{\tilde{H}}(\vec{k}) \begin{pmatrix} \vdots \\ \tilde{u}_j(\vec{k}) \\ \vdots \end{pmatrix} \tag{4.3.27}$$

welche den Titel dieses Abschnitts „Teildiagonalisierung“ erklärt. Die Periodizität bewirkt, daß die Fouriertransformation die Matrix $\underline{H}$ in einzelne Blöcke zu den Ausbreitungsvektoren $\vec{k}$ zerlegt.
Anstatt eines $F\times F$-Eigenwertproblems hat man nun $n-1$ komplexe Eigenwertprobleme der Dimension $(3l-1)\times(3l-1)$ zu $\vec{k} \neq \vec{0}$ zu lösen und eines der Dimension $(3l+2)\times(3l+2)$ zu $\vec{k} = \vec{0}$. Die Einsparung an Rechenzeit ist beachtlich:

$$\frac{[n(3l-1)+3]^3}{[2(3l+2)]^3 + (n-1)[2(3l-1)]^3} \approx \frac{n^2}{8} \quad \approx 2600 \;\text{ für } n = 144,\; l = 7 \tag{4.3.28}$$

Die Diagonalisierung der hermiteschen Matrizen ergibt für $\vec{k} \neq \vec{0}$

$$\underline{\tilde{H}}(\vec{k},s)\vec{\epsilon}(\vec{k},s) = \lambda(\vec{k},s)\vec{\epsilon}(\vec{k},s) \qquad s = 1...3l-1 \quad \vec{\epsilon}(\vec{k},s) \in \mathbb{C}^{3l-1}, \lambda(\vec{k},s) \in \mathbb{R}^+$$
$$\lambda(-\vec{k},s) = \lambda(\vec{k},s), \quad \vec{\epsilon}(-\vec{k},s) = e^{i\varphi(\vec{k},s)}\vec{\epsilon}^*(\vec{k},s) \tag{4.3.29}$$

und für $\vec{k} = \vec{0}$

$$\begin{pmatrix} \underline{\tilde{H}}(\vec{0}) & \sqrt{n}\underline{E} \\ \sqrt{n}\underline{E}^\top & \underline{F} \end{pmatrix} \begin{pmatrix} \vec{\epsilon}(\vec{0},s) \\ \vec{b}(s) \end{pmatrix} = \lambda(\vec{0},s) \begin{pmatrix} \vec{\epsilon}(\vec{0},s) \\ \vec{b}(s) \end{pmatrix} \qquad \lambda(\vec{0},s) \in \mathbb{R}^+ \quad \begin{pmatrix} \vec{\epsilon}(\vec{0},s) \\ \vec{b}(s) \end{pmatrix} \in \mathbb{R}^{3l+2} \tag{4.3.30}$$

mit jeweils orthonormalen Eigenvektoren

$$\vec{\epsilon}^*(\vec{k},s)\cdot\vec{\epsilon}(\vec{k},s') = \delta_{s,s'} \;\; \forall s,s' = 1...3l-1 \qquad \vec{\epsilon}(\vec{0},s)\cdot\vec{\epsilon}(\vec{0},s') = \delta_{s,s'} \;\; \forall s,s' = 1...3l+2 \tag{4.3.31}$$

Man macht daher den Ansatz

$$\tilde{u}_i(\vec{k}) = \sum_s A_{\vec{k},s}\, \vec{\epsilon}(\vec{k}, s) \qquad A_{\vec{k},s} \in \mathbb{C} \tag{4.3.32}$$

was für die Auslenkungen zur Gleichung

$$u_i(\vec{R}) = \tfrac{1}{\sqrt{n}} \sum_{\vec{k}} \sum_s A_{\vec{k},s}\, \epsilon_i(\vec{k}, s)\, e^{i\vec{k}\cdot\vec{R}} \tag{4.3.33}$$

führt, welche reellwertig sein müssen:

$$u_i(\vec{R}) \in \mathbb{R} \;\; \forall \vec{R}, \forall i \quad \Rightarrow \quad A_{-\vec{k},s} \overset{!}{=} A^*_{\vec{k},s}\, e^{-i\varphi(\vec{k},s)} \tag{4.3.34}$$

Eingesetzt in den harmonischen Hamiltonian (4.3.27) ergibt sich

$$\begin{aligned} 2(H - H^0) &= \sum_{\vec{k}} \sum_{s,s'} A^*_{\vec{k},s}\; \vec{\epsilon}^{\,*}(\vec{k}, s) \underline{\tilde{H}}(\vec{k}) \vec{\epsilon}(\vec{k}, s')\; A_{\vec{k},s'} \\ (4.3.31) \Rightarrow \qquad &= \sum_{\vec{k}} \sum_{s,s'} A^*_{\vec{k},s} A_{\vec{k},s'}\, \lambda(\vec{k}, s')\, \delta_{s,s'} \\ H &= H^0 \;+\; \tfrac{1}{2} \sum_{\vec{k}} \sum_s |A_{\vec{k},s}|^2\, \lambda(\vec{k}, s)\,. \end{aligned} \tag{4.3.35}$$

Die Spektren $\{\lambda(\vec{k}, s)\}$ bzw. $\{\lambda_s\}$ die mit und ohne Teildiagonalisierung berechnet werden, stimmen bis auf 12 Dezimalstellen überein. Die beiden Translationsmoden werden ebenfalls reproduziert und entstehen bei $\vec{k} = \vec{0}$. Somit ergibt sich formal völlige Übereinstimmung mit 4.2.12.
Die Moden zu $\vec{k} = \vec{0}$ sind sinngemäß in obiger Darstellung eingeschlossen, wobei man für die Box-Freiheitsgrade $\vec{B}$

$$\vec{b}(\vec{0}, s) \in \mathbb{R}^3 \qquad B_i = \sum_{s=1}^{3l+2} A_{\vec{0},s}\, b_i(\vec{0}, s)\,. \tag{4.3.36}$$

erhält.

4.4 Berechnung des harmonischen Gibbs-Potentials

Die Motivation für die harmonische Näherung und Diagonalisierung des Hamiltonians war es, zu einer Darstellung zu gelangen, die es erlaubt, über die Zustandssumme das Gibbs'sche Potential zu bestimmen.
Hierzu wird man zuerst die Zustandssumme (siehe Anhang A) anstatt in den bisherigen Konfigurationsvariablen $\underline{r}, L_x, L_y, \alpha$, nun in den Amplituden $\{A_s\}$ der Eigenmoden ausdrücken. Der Übersichtlichkeit halber sei dies an der Darstellung (4.2.11), d.h. ohne Zerlegung nach Vektoren $\vec{k}$ gezeigt.

Für die Zustandssumme im $N\Pi T$-Ensemble hat man

$$Z_{N\Pi T} = \frac{1}{\sigma^{F-1}} \int_0^\infty \mathrm{d}L_x \int_0^\infty \mathrm{d}L_y \int_0^\pi \mathrm{d}\alpha \int_{\mathrm{Box}(L_x,L_y,\alpha)} \underline{\mathrm{d}r}\; e^{-\beta H(\underline{r},L_x,L_y,\alpha)} \,. \tag{4.4.37}$$

Diese „natürlichen“ Variablen wurden durch ihre Auslenkungen $\vec{u}$ aus der Gleichgewichtslage ersetzt. Da dies eine einfache Verschiebung der Koordinaten bedeutet, wird nur eine Änderung der jeweiligen Integrationsgrenzen bewirkt.
Hinter der harmonischen Näherung steht die Annahme, daß bei den betrachteten Temperaturen größere Auslenkung nicht auftreten, weil sie mit einer hohen Energie bestraft würden und entsprechend ein verschwindendes Boltzmann-Gewicht hätten. Wenn man also die Integrationsgrenzen auf $-\infty...\infty$ erweitert, nimmt man nur Zustände hinzu, deren Boltzmann-Gewicht nicht zur Zustandssumme beiträgt:

$$Z_{N\Pi T} = \frac{1}{\sigma^{F-1}} \int_{-\infty}^{\infty} \mathrm{d}u_1 ... \int_{-\infty}^{\infty} \mathrm{d}u_F \; e^{-\beta H(\vec{u})} \tag{4.4.38}$$

Die Transformation (4.2.11) ist orthonormal denn die Eigenvektoren $\{\vec{\epsilon}_i\}$ bilden eine VONB des F-dimensionalen Konfigurationsraums der $\vec{u}$. Es gilt somit

$$\vec{u} = \underline{T}\,\vec{A} \qquad \underline{T} \in \mathbb{R}^{F\times F} \qquad \text{mit} \quad \det \underline{T} = 1 \tag{4.4.39}$$

In den $\vec{A}$ stellt sich deshalb die Zustandssumme als

$$Z_{N\Pi T} = \frac{1}{\sigma^{F-1}} \int_{-\infty}^{\infty} \mathrm{d}A_1 ... \int_{-\infty}^{\infty} \mathrm{d}A_F \; e^{-\beta H(\vec{A})} \tag{4.4.40}$$

dar, wofür man nun die Unabhängigkeit der Moden (4.2.12) ausnutzen kann.
Dazu geht man von einem Hamiltonian der Form

$$H_{harm} = H^0 + \sum_{s=1}^{F} \lambda_s A_s^2 \, x_1 \tag{4.4.41}$$

aus, worin $x_1 \in \mathbb{R}\backslash\{0\}$ eine beliebige Konstante ist und erhält für die Zustandssumme

$$\begin{aligned} \sigma^{F-1} Z_{harm} &= \int_{-\infty}^{\infty} \mathrm{d}A_1 ... \int_{-\infty}^{\infty} \mathrm{d}A_F \;\; \exp(-\beta[H^0 + \sum_{s=1}^{F} \lambda_s A_s^2 \, x_1]) \\ &= e^{-\beta H^0} \prod_{s=1}^{F} \left[\int_{-\infty}^{\infty} e^{-\beta x_1 \lambda_s A^2} \mathrm{d}A \right] \end{aligned} \tag{4.4.42}$$

Hier ist zu berücksichtigen, daß zwei der F Moden Translationsmoden zu $\lambda_s = 0$ sind. Diese seien $s = 1$ und $s = 2$.

$$\begin{aligned} \sigma^{F-1} Z_{harm} &= e^{-\beta H^0} \, x_2 \prod_{s=3}^{F} \left[\int_{-\infty}^{\infty} e^{-\beta x_1 \lambda_s A^2} \mathrm{d}A \right] \qquad \text{mit} \quad x_2 := \left(\int_{-\infty}^{\infty} \mathrm{d}A \right)^2 \\ &= e^{-\beta H^0} \, x_2 \prod_{s=3}^{F} \sqrt{\frac{\pi}{\beta x_1 \lambda_s}} \qquad = e^{-\beta H^0} \, x_2 \left(\sqrt{\frac{\pi}{x_1}} \right)^{F-2} \prod_{s=3}^{F} (\beta \lambda_s)^{-1/2} \end{aligned} \tag{4.4.43}$$

Damit ergibt sich für das Gibbs-Potential

$$\begin{aligned} G(T) &= -\frac{1}{\beta}\,\ln Z_{harm}(\beta) \\ &= H^0 + \frac{k_B T}{2}\sum_{s=3}^{F}\ln(\beta\lambda_s) - x(\sigma, F, x_1, x_2)\cdot T \end{aligned} \tag{4.4.44}$$

Das Gibbs'sche Potential ist hier also nur bis auf $G' = G - xT$ festgelegt.
Es wird aber βG und nicht direkt G thermodynamisch integriert. Für βG gilt jedoch

$$\Delta(\beta G') = (\beta_2 G_2 - xT_2\beta_2) - (\beta_1 G_1 - xT_1\beta_1) = \beta_2 G_2 - \beta_1 G_1 = \Delta(\beta G) \tag{4.4.45}$$

Einerseits legt die thermodynamische Integration den Faktor x nicht fest, andererseits wir für jede Wahl von x der Ausgangswert $\beta_1 G_1'$ korrekt nach $\beta_2 G_2'$ entwickelt.
Wichtig ist also nur, daß jedem der erzeugten Anfangswerte $\beta_1 G_1'$ der Integration konsistent der gleiche Wert x zugrunde liegt !
Aus der Theorie der Festkörper ist bekannt, daß in harmonischer Näherung der Ausdehnungskoeffizient eines Kristalls für $T \mapsto 0$ verschwindet. Es ist daher ein interessanter Test für die hier durchgeführten Rechnungen, ob dieses Verhalten reproduziert werden kann. Siehe hierzu Anhang C.

Zur Auswertung des Gibbs'schen Potentials nach (4.4.44) ist die Kenntnis der einzelnen Eigenwerte nicht notwendig [49].
Es genügt vielmehr, den Term $\sum_{s=3}^{F}\ln\lambda_s$ bestimmen zu können:

$$(4.4.44) \Rightarrow \quad G(T) = H^0 - \frac{f}{2}k_B T\ln(k_B T) + \frac{k_B T}{2}\sum_{s=3}^{F}\ln\lambda_s \tag{4.4.46}$$

Dieser Term steht in Zusammenhang mit der Determinante der Kopplungsmatrix $\underline{H}$ aus (4.2.8):

$$\begin{aligned} \sum_{s=3}^{F}\ln\lambda_s &= \ln\left(\prod_{s=3}^{F}\lambda_s\right) = \ln\det\begin{pmatrix}\lambda_3 & & 0\\ & \ddots & \\ 0 & & \lambda_F\end{pmatrix} \\ &= \ln\det(\underline{T}^{\top}\underline{H}\,\underline{T}) = \ln\det(\underline{H}) \qquad \text{denn} \qquad \det\underline{T} = 1 \end{aligned} \tag{4.4.47}$$

Somit gilt

$$(4.4.44) \Rightarrow \quad G(T) = H^0 - \frac{f}{2}k_B T\ln(k_B T) + \frac{k_B T}{2}\ln\det(\underline{H})\,. \tag{4.4.48}$$

Bei der praktischen Verwendung dieser Darstellung tritt das Problem auf, daß man die beiden Translationsmoden mit $\lambda_1 = \lambda_2 = 0$ leicht eliminieren konnte, diese aber in der $F\times F$-Matrix $\underline{H}$ noch enthalten sind und $\det(\underline{H}) = 0$ bewirken.
Diese Problem läßt sich mit einer geeigneten Transformation so lösen, daß man nur

noch eine $f \times f$-Unterdeterminante auswerten muß. Hierzu ersetzt man die x- und y-Auslenkung eines, z.B. des ersten Monomers durch die xy-Koordinaten des Schwerpunkts. Nach Streichung der Zeilen und Spalten, die zu diesem Koordinatenpaar gehören, gelangt man zu einer positiv definiten Unter-Matrix.
Auch wenn die Berechnung einer Determinante nur ca. $1/20$ der Rechenzeit für die Diagonalisierung derselben Matrix benötigt, skaliert die Zeit ebenfalls mit N^3, so daß man die Teilchenzahl nur unwesentlich vergrößern kann, ohne an technische Grenzen zu stoßen.
Da typische Werte für $\sum_{s=3}^{F} \ln \lambda_s$ in der Größenordnung $O(2 \cdot 10^5)$ lagen, ergibt sich für die Determinante $O(10^{10000})$, was eine weitere Transformation der Matrix numerisch notwendig macht.

4.5 Bestimmung der Übergangstemperatur

Kennt man ihr Gibbs'sche Potential, ist man in der Lage zu entscheiden, welche von zwei Phasen, bei gegebenem Druck und Temperatur, die stabile ist.
Dies sei am Beispiel des Übergangs LC-NNN↔LC-NN für die Kopfgrößen $\sigma_H = 1.1\sigma$ und $\sigma_H = 1.2\sigma$ gezeigt.

Im Fall der kleineren Kopfgruppen handelt es sich dabei um den Übergang zwischen den konventionell NN- bzw. NNN-geneigten Phasen, bei den größeren Kopfgruppen ist die NN-Phase dagegen *moduliert* (vgl. Abschnitt 5.3.1). Aufgrund der damit verbundenen Überstruktur konnte die (restliche) Periodizität hierbei nicht ausgenutzt werden, was die volle Behandlung aller F Freiheitsgrade erforderlich machte.

Die Grenzlinie zwischen den jeweils zwei Phasen ist durch die Forderung

$$\Delta G(T,\Pi) := G_{\mathrm{NNN}}(T,\Pi) - G_{\mathrm{NN}}(T,\Pi) \stackrel{!}{=} 0 \tag{4.5.49}$$

festgelegt.
Ausgehend von Grundzuständen zum Wert $(T = 0, \Pi)$, die man nach harmonischer Näherung als Startpunkte der thermodynamischen Integration benutzt, ist man in der Lage, die Gibbs'sche freie Energie für beliebige Zustände zu berechnen und so die Phasengrenze zu lokalisieren.

Innerhalb eines einzelnen Phasengebietes ändert sich das Gibbs'sche Potential stetig[4] und man sollte, wie von einer Zustandsgröße zu erwarten, *einen* Wert für das Gibbs'sche Potential $G(T,\Pi)$ erhalten, gleichgültig auf welchem Weg man sich diesem Zustand nähert.
Es ist daher ein Test für das ganze Verfahren, ob dieses Verhalten auch reproduziert wird.

[4] Da dabei keine Umwandlungswärme (Enthalpie-Sprung) auftritt, kann auch über Phasenübergänge 2. Ordnung hinweg integriert werden.

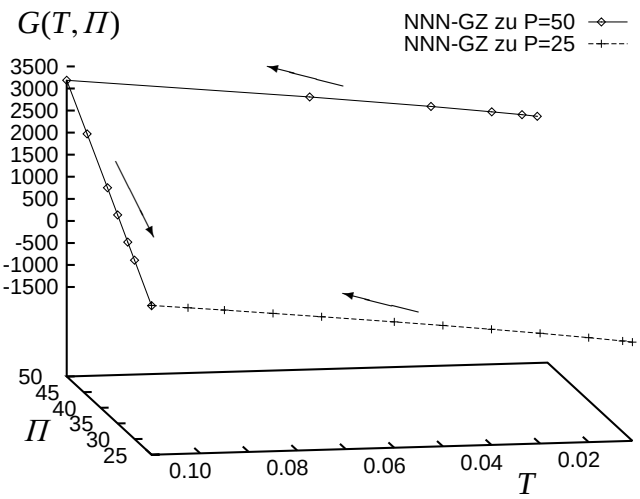


Abbildung 4.2: Integration von $G(\Pi, T)$ in der NNN-Phase

Um dies zu prüfen wurde für $\sigma_H = 1.2\sigma$ aus den NNN-Grundzuständen $\lim T \to 0$ zu $\Pi = 25$ und $\Pi = 50$ heraus zum Punkt $T = 0.1, \Pi = 25$ integriert.

Wie man sieht, treffen sich beide Kurven nahezu ($\delta G \approx 6$).

Daß die Integration vollständig in dem Gebiet abläuft, in dem eigentlich die modulierte NN-Phase stabil wäre (vgl. Abschnitt 5.3.2), stellt kein Problem dar, da sich während der ganzen Simulation keine Anzeichen eines Übergangs in die modulierte oder konventionelle NN-Phase zeigten.

Da man in der Formel (4.4.46) für das Gibbs-Potential in harmonischer Näherung $G(T)$ nur die Temperatur, nicht aber den Druck als Variablen zur Verfügung hat, muß die Integration an einem Punkt beginnen, der auf einer Isobaren mit dem Grundzustand liegt, dessen Hessematrix man diagonalisiert hat. Es verbleibt also zu klären, bei welcher Temperatur die Integration beginnen soll, bzw. bis zu welcher Temperatur (4.4.46) korrekte Werte liefert. Diese Entscheidung erweist sich als unkritisch.
Entlang einer Isobaren wird der Integrand $-H(T)/T^2$ (siehe 4.1.2) durch die Enthalpie bestimmt. Bei niedrigen Temperaturen ($T < 0.1$) reproduziert die Simulation hierfür in sehr guter Näherung den Verlauf, den die harmonische Näherung vorhersagt:

$$H(T) \approx H^0 + \frac{f}{2} k_B T \tag{4.5.50}$$

Integriert man diesen Ausdruck gemäß (4.1.2) erhält man exakte Übereinstimmung mit den Werten für $G(T)$, die man mittels (4.4.46) direkt aus der Zustandssumme berechnet. Innerhalb des Gültigkeitsbereichs der Näherung (4.5.50) kann daher der Integrationsbeginn beliebig gewählt werden.

Die Grundzustandsiterationen ergaben für den Übergang zwischen der NN- und NNN-Phase bei folgende Abschätzungen der Phasengrenze:

bei $T = 0$ für $\sigma_H = 1.1\sigma$: $\Pi_{\mathrm{NN}\leftrightarrow\mathrm{NNN}} \approx 9.17$ für $\sigma_H = 1.2\sigma$: $\Pi_{\mathrm{mod}\leftrightarrow\mathrm{NNN}} \approx 53.0$,

wobei bei höherem Druck jeweils die NNN-geneigte Phase vorliegt.
Um den Verlauf der Phasengrenze zu verfolgen wurde zunächst für $\sigma_H = 1.1\sigma$ die Übergangstemperatur auf der Isobaren $\Pi := 20$ und für $\sigma_H = 1.2\sigma$ auf der Isobaren $\Pi := 50$ aus dem Gibbs'schen Potential bestimmt.

Bei der Berechnung der Kopplungsmatrizen und deren Diagonalisierung kann man, außer bei der modulierten Konfiguration, deren Gitter-Periodizität ausnützen.

Unterstellt man, daß die Übergangstemperatur in dem Bereich liegt, in dem das System dem harmonischen Enthalpieverlauf (4.5.50) genügt, gelangt man durch Einsetzen von $G(T)$ (4.4.46) in (4.5.49) zu einem Schätzwert[5] der Übergangstemperatur:

$$T_{\mathrm{NN}\leftrightarrow\mathrm{NNN}} \approx -2.0\frac{H^0_{\mathrm{NN}} - H^0_{\mathrm{NNN}}}{\Lambda_{\mathrm{NN}} - \Lambda_{\mathrm{NNN}}} \qquad \text{mit} \quad \Lambda := \sum_{i=3}^{F} \ln \lambda_i \tag{4.5.51}$$

Mit den gefundenen Konfigurationen erhält man hieraus:

für $\sigma_H = 1.1\sigma,\ \Pi = 20:\ \ T_{\mathrm{NN}\leftrightarrow\mathrm{NNN}} \approx 0.116 \qquad \sigma_H = 1.2\sigma,\ \Pi = 50:\ \ T_{\mathrm{mod}\leftrightarrow\mathrm{NNN}} \approx 0.069$

Für die Integration wurden Simulationen verwendet, die beginnend mit den iterierten Grundzuständen, schrittweise aufgeheizt wurden. Die erhaltenen Daten stimmen aber weitgehend mit denen überein, die während der Abkühlung entstanden, die zur Gewinnung der Startwerte der Grundzustandsiteration nötig war.

Als Startpunkte der Integration wurden die Temperaturen $T = 0.0001$ für $\sigma_H = 1.1\sigma$ und $T = 0.001$ für $\sigma_H = 1.2\sigma$ festgelegt.

Man findet, daß sich die Gibbs'schen Potentiale beider Phasen bei den großen Köpfen an der 3., bei den kleinen Köpfen erst an der 4. Stelle unterscheiden und der Schnittpunkt daher recht empfindlich von den simulierten Enthalpien abhängt.

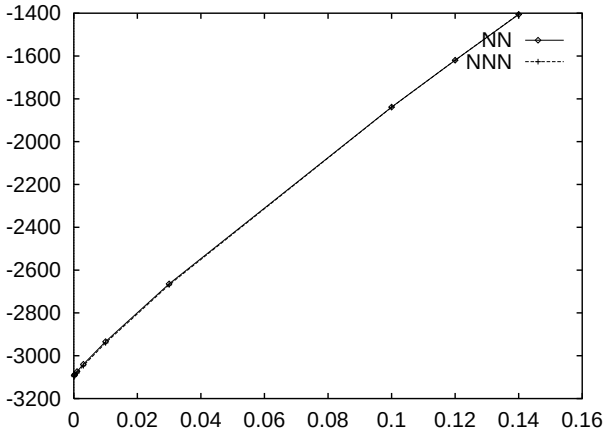


Abbildung 4.3: freie Energien $G(T, 20)$ vs. T beider Phasen für $\sigma_H = 1.1\sigma,\ \Pi = 20$

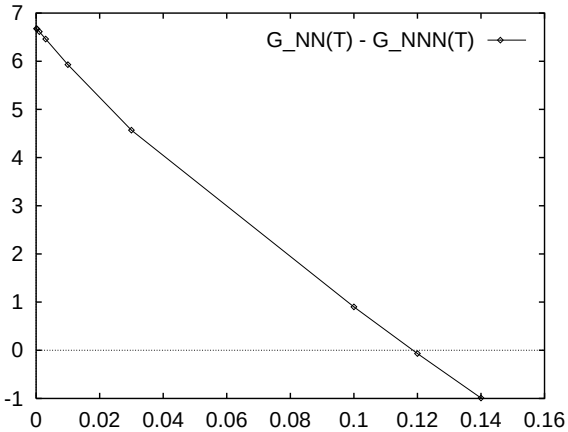


Abbildung 4.4: Differenz $\Delta G(T, 20)$ vs. T für $\sigma_H = 1.1\sigma,\ \Pi = 20$

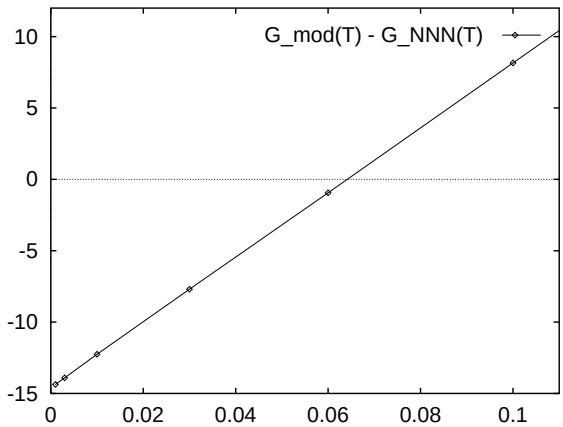


Abbildung 4.5: Differenz $\Delta G(T, 50)$ vs. T für $\sigma_H = 1.2\sigma,\ \Pi = 50$

[5] In diesen Schätzwert gehen ausschließlich Grundzustands-Größen ein.

Die Nulldurchgänge und damit die Übergangstemperaturen liegen bei

für $\sigma_H = 1.1\sigma,\ \ \Pi = 20:\ \ T_{\mathrm{NN}\leftrightarrow\mathrm{NNN}} \approx 0.119 \qquad \sigma_H = 1.2\sigma,\ \ \Pi = 50:\ \ T_{\mathrm{mod}\leftrightarrow\mathrm{NNN}} \approx 0.064$

in guter Übereinstimmung mit obigen Vorhersagen aus der harmonischen Näherung.

Kennt man einen Punkt (T_g, Π_g) auf der Grenzlinie, kann man aus der Forderung

$$\begin{aligned} G_{\mathrm{NN}}(T_g + \delta T, \Pi_g + \delta\Pi) &\stackrel{!}{=} G_{\mathrm{NNN}}(T_g + \delta T, \Pi_g + \delta\Pi) \\ (4.1.2) \Rightarrow \quad \frac{\delta\Pi}{\delta T} &= \frac{H_{\mathrm{NN}}(T_g, \Pi_g) - H_{\mathrm{NNN}}(T_g, \Pi_g)}{T_g \cdot (A_{\mathrm{NN}}(T_g, \Pi_g) - A_{\mathrm{NNN}}(T_g, \Pi_g))}\,, \end{aligned} \tag{4.5.52}$$

d.h. aus der Clausius-Clapeyron-Gleichung, auch die Steigung der Grenzlinie in der unmittelbaren Umgebung erkennen.
Da hierfür kleine Differenzen der simulierten Enthalpien und Flächen dividiert werden müssen ist die Genauigkeit der gewonnenen Steigungen fraglich. Zum Vergleich sei hier der Differenzenquotient angegeben, welcher der Sekantensteigung zwischen dem gefundenen Punkt (T_g, Π_g) und dem Übergang bei $T = 0$ (vgl. Abschnitte 5.2,5.3.2) entspricht:

$$\begin{array}{lcll} \text{für } \sigma_H = 1.1\sigma & & & \\ T_g = 0.119,\ \ \Pi_g = 20 & : & \dfrac{\delta\Pi}{\delta T} \approx 135 & \dfrac{\Delta\Pi}{\Delta T} = \dfrac{20 - 9.17}{0.119 - 0} = 91 \\ \text{für } \sigma_H = 1.2\sigma & & & \\ T_g = 0.064,\ \ \Pi_g = 50 & : & \dfrac{\delta\Pi}{\delta T} \approx -46 & \dfrac{\Delta\Pi}{\Delta T} = \dfrac{50 - 53.03}{0.064 - 0} = -47 \end{array}$$

Bei den großen Köpfen ergibt sich sehr gute Übereinstimmung und auch die Werte für die kleinen Köpfe widersprechen sich nicht notwendig, da die Phasengrenzlinie durchaus gekrümmt sein darf.

Im System der großen Köpfe wurden durch fortgesetzte Integration noch weitere Punkte auf der Phasengrenze bestimmt (siehe Phasendiagramm im Abschnitt 5.3.5).

Bei den kleinen Köpfen gelang dies nicht, da oberhalb von $T \gtrsim 0.15$ bzw. oberhalb von $\Pi \gtrsim 25$ die NN-Neigung instabil wurde und derart gestartete Simulationen selbständig in NNN-Neigung übergingen. Eine Verfolgung von $G_{NN}(T, \Pi)$ war somit unmöglich.

Im Zusammenhang mit dieser Instabilität mag die Erscheinung stehen, daß z.B. bei $\Pi = 30$ die Kopplungsmatrix des Grundzustands nicht mehr nur zwei (Translations)-Moden zum Eigenwert null besitzt, sondern eine dritte derartige Mode auftritt. Dieser Wert $\lambda_3 = 0$, der zum Ausbreitungsvektor $\vec{k} = \vec{0}$ gehört und in der entsprechenden NNN-Konfiguration nicht erscheint, bewirkt auch eine (zusätzliche) Divergenz der harmonischen Zustandssumme (4.4.42).
Untersucht man den zugehörigen Eigenvektor $\vec{\epsilon}(\vec{k} = \vec{0}, s = 3)$ aus Eigenwertgleichung (4.3.30), beschreiben dessen Komponenten die Auslenkungen der Monomere, die wegen $\vec{k} = \vec{0}$ alle Ketten gleich, d.h. ohne Phasendifferenz erfassen und keine Anregungen der Boxparameter L_x, L_y, α beinhalten.

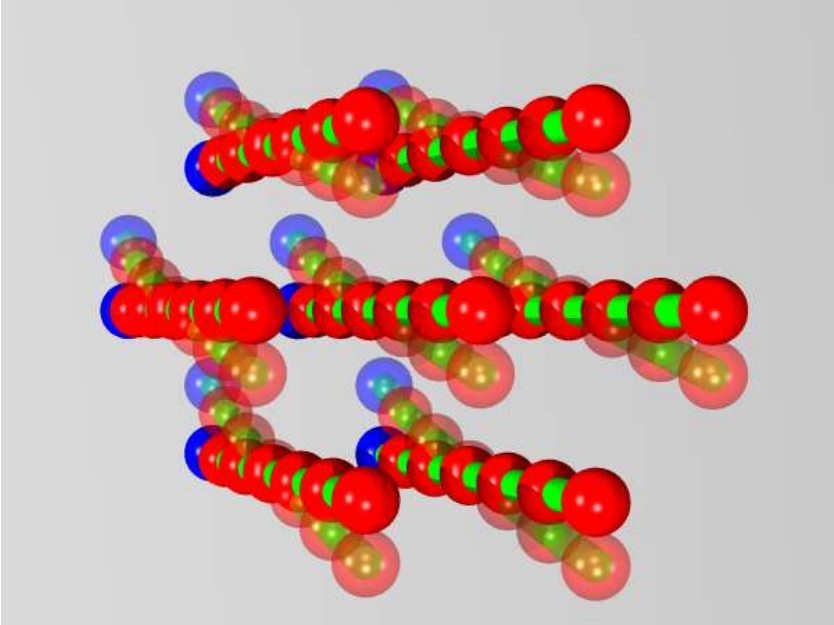


Abbildung 4.6: Draufsicht auf einen Ausschnitt der Simulationsbox; $\Pi = 30, \quad T = 0, \quad \sigma_H = 1.1\sigma$
Die bestehende NN-Grundzustandskonfiguration ist massiv dargestellt,
die gemäß $\vec{\epsilon}(\vec{k} = \vec{0}, s = 3)$ ausgelenkten Ketten transparent.
Zur besser Übersicht entspricht der Kugeldurchmesser nur 0.15σ bzw. $0.15\sigma_H$.

In der Draufsicht ist zu erkennen, daß die Auslenkungen, die allein in y-Richtung liegen, zu einer effektiven Drehung der mittleren Neigungsrichtung führen. Wegen $\lambda_3 = 0$ fehlt eine rücktreibende Kraft, weswegen dies nicht als Oszillation zu verstehen ist. Mit dem Auftreten dieser *Entkopplungs-Mode* sind also Neigungen, die von der Vorzugsrichtung NN abweichen, ohne „Energiestrafe“ möglich.

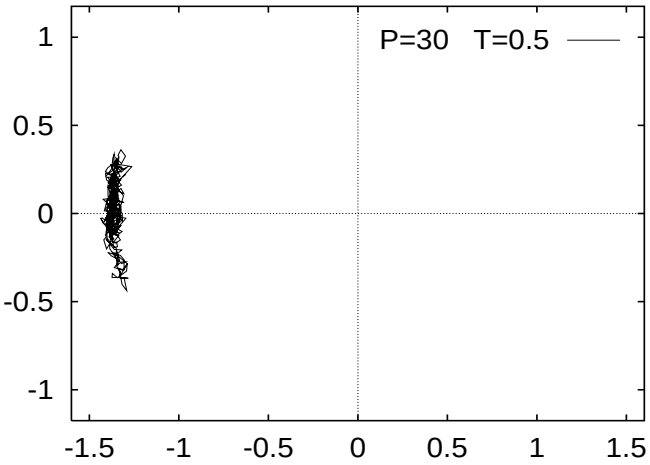


Abbildung 4.7: Trajektorie des Moleküldirektors $[\vec{D}]$
D_x, D_y über MCS für $\Pi = 30, \quad T = 0.5, \quad \sigma_H = 1.1\sigma$

Verfolgt man den mittleren Moleküldirektor $[\vec{D}]$ über die MC-Schritte, findet man bei $\Pi = 30, T = 0.5$ eine weiträumige „Bewegung“ um die NN-Richtung.
Hierbei ändert sich zwar die Neigungsrichtung, der Neigungswinkel ϑ gegen die Flächennormale bleibt dagegen annähernd konstant.

Es liegt nahe, diese Bewegung mit der Entkopplungs-Mode des Grundzustands in Verbindung zu bringen.

Kapitel 5

Simulationsergebnisse

Die hier vorgestellten Resultate reihen sich in die Simulationen von *F.M. Haas* [25] und *H.O. Lange* [42] ein, die weitgehend dasselbe Modell verwendet haben.
F.M. Haas untersuchte Systeme, in denen sich die Kopfgruppen bezüglich ihrer Wechselwirkungen nicht von den Schwanzmonomeren unterschieden sowohl im NAT- [26] als auch im $N\Pi T$-Ensemble [27]. *H.O. Lange* betrachtete Ketten, deren Köpfe über Lennard-Jones-Potentiale wechselwirken, und auch die hier verwendeten Soft Core-Köpfe jeweils im $N\Pi T$-Ensemble und analysierte die Abhängigkeit des Grundzustands von der Kopfgröße [60].

Dieser Abschnitt gliedert sich in drei Teile, in denen die Ergebnisse für drei verschiedene Kopfgrößen präsentiert werden.
Aufgrund früherer Grundzustands-Analysen wurde davon ausgegangen, daß bei Kopfgrößen $0.95\sigma < \sigma_H < 1.13\sigma$ im Grundzustand $\Pi \mapsto 0, T \mapsto 0$ Neigung zu übernächsten Nachbarn vorliegt, während bei größeren Köpfen bei $\Pi = 0$ NN-Neigung auftritt, die erst bei höheren Drücken in NNN übergeht.
Um beide Regimes zu untersuchen erfolgten Simulationen mit $\sigma_H := 1.10\sigma$ (*„kleine Köpfe“*) und $\sigma_H := 1.20\sigma$ (*„große Köpfe“*). Zusätzlich wurden noch Systeme mit deutlich größeren Kopfgruppen betrachtet, bei denen das Größenmißverhältnis so stark ist, daß die Schwänze keine geschlossene Monolage mehr bilden.

5.1 Grundzustandsanalyse

In die früheren Grundzustandsanalysen [60][42][61] gingen immer bestimmte Annahmen über die Konfiguration bei $T = 0$ ein. Beispielsweise wurden die Ketten als starre Stäbe behandelt, oder wenigstens die Bondlängen als uniform und die Bondwinkel als gestreckt angesehen. Im Gegensatz dazu kann die Simulation alle dem System zur Verfügung stehenden Freiheitsgrade ausnutzen.
Die hier beschriebene **Grundzustands-Iteration** erlaubt dies ebenfalls.

Da zur Auswertung der freien Energie (vgl. 4.4.46) in harmonischer Näherung eine positiv-definite Kopplungsmatrix vorliegen muß, ist es nötig, das System möglichst nahe an den Grundzustand $\lim T \mapsto 0$ zu bringen. Mit der normalen Simulation gelingt das nur bedingt, vermutlich da die zufällige Abfolge der particle-displacement-Moves kaum kollektive Konfigurationsänderungen zuläßt. Um den Grundzustand zu einem bestimmten Druck Π genauer zu erhalten, kann man ein einfaches Gradienten-Verfahren auf dem F-dimensionalen Konfigurationsraum (vgl. Abschnitt 4.2) iterieren. Leider scheitert das Newtonverfahren

$$\vec{U}_{n+1} := \vec{U}_n - \left(\nabla \otimes \nabla H|_{\vec{U}_n}\right)^{-1} \vec{\nabla} H|_{\vec{U}_n} \tag{5.1.1}$$

daran, daß in jedem Iterationsschritt die Hessematrix $\nabla \otimes \nabla H$ neu berechnet werden müßte. Wegen des großen damit verbundenen Rechenaufwands, wird man dies nur einmal – für den Endzustand – unternehmen.
Man erhält ein schnelles, wenn auch schlecht konvergierendes Verfahren, wenn man jeweils in Richtung des aktuellen Gradienten bis zum lokalen Minimum geht:

$$\begin{gathered} \vec{U}_{n+1} := \vec{U}_n - \lambda \cdot \vec{\nabla} H|_{\vec{U}_n} \qquad \text{mit} \quad H_n(\lambda) \leq H_n(\mu) \;\; \forall \mu \\ H_n(\mu) := H(\vec{U}_n - \mu \cdot \vec{\nabla} H|_{\vec{U}_n}) \end{gathered} \tag{5.1.2}$$

Die Berechnung eines Gradienten $\vec{\nabla} H$ benötigt nur wenige CPU-Sekunden und kann daher in jedem Iterationsschritt durchgeführt werden. Die Konvergenz des Verfahrens läßt sich optimieren, wenn man einen „Relaxationsfaktor“ $\lambda \mapsto \kappa\lambda$ mit $\kappa = O(1)$ einführt.

Nach dem 3. Hauptsatz stimmen bei $T = 0$ die Gibbs’sche freie Energie (4.1.1) und die Enthalpie überein [53]. Mit der Iteration ist es daher möglich, nicht nur Grundzustände für die harmonische Näherung zu präparieren, sondern aus den Grundzustandsenthalpien auch die Übergänge zwischen den verschiedenen Phasen bei $T = 0$ genau[1] zu bestimmen.

[1] Für die Grundzustandsenthalpien H_0 erhält man etwa 10 signifikante Stellen.

5.2 System kleiner Köpfe

Für Köpfe, die um 10% größer sind, als die Schwanzmonomere, d.h. $\sigma_H := 1.1\sigma$, erhält man aus der Grundzustands-Iteration[2] folgende Werte für die Enthalpie bzw. das Gibbs'sche Potential bei $T = 0$:

Π	H_{NN}/N	H_{NNN}/N
20	-3.06825577	-3.07488654
10	-4.28703993	-4.28755245
8	-4.53349914	-4.53276983

Interpoliert man zwischen $\Pi = 8$ und $\Pi = 10$, findet man unterhalb des Drucks $\Pi \approx 9.17$ Neigung zu nächsten Nachbarn, oberhalb davon, Neigung zu übernächsten Nachbarn.

Die vorliegenden Simulationsergebnisse wurden i.allg. dadurch gewonnen, daß eine Anfangskonfiguration zu niedriger Temperatur entlang einer Isobaren sukzessive aufgeheizt wurde. Bei jeder Temperatur erfolgte eine Simulation von (typischerweise) 200.000 MCS, wovon die ersten 70.000 MCS als Äquilibrierung betrachtet wurden und daher nicht in die „Messungen“ eingingen. Während des restlichen Laufs wurden alle 500 MCS die Observablen ausgewertet und Histogramme aufgenommen.
Bei dieser Länge der Läufe und der gewählten Temperaturschritte konnte keine nennenswerte Hysterese[3] gegenüber einer schrittweisen Abkühlung von hohen Temperaturen beobachtet werden.

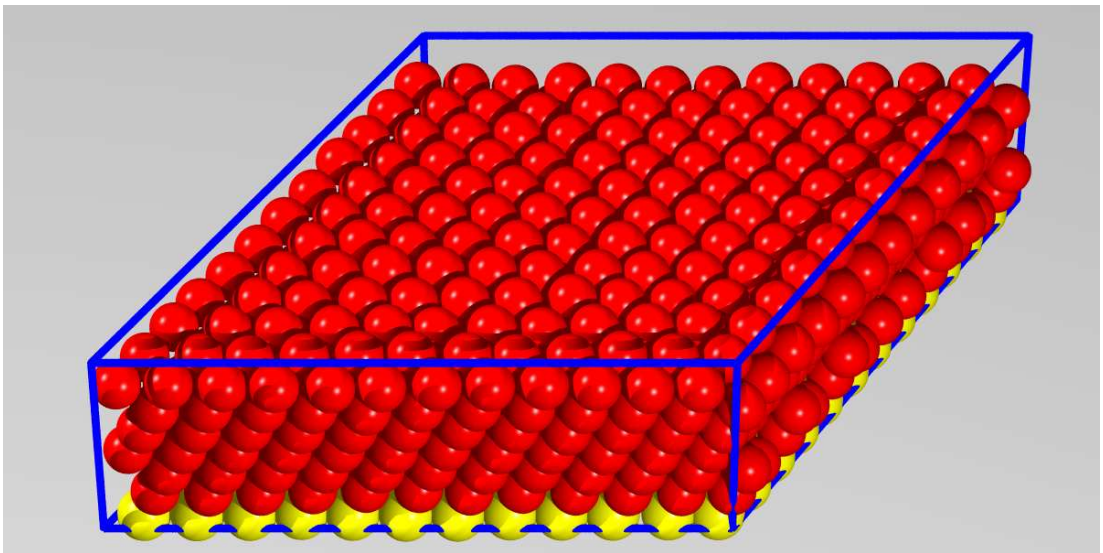


Abbildung 5.1: Snapshot einer Konfiguration bei $\Pi = 1$, $T = 0.1$ für $\sigma_H = 1.1\sigma$
Die eff. Monomere sind als Kugeln dargestellt, deren Durchmesser σ bzw. σ_H entspricht.
Weitere Snapshots finden sich im Abschnitt 5.3.5.

[2] Erzwingt man eine modulierte Phase als Anfangskonfiguration (vgl. 5.3.1), gelingt es der Iteration bei dieser Kopfgröße das System in eine konventionell NN-geneigte Konfiguration zu relaxieren.

[3] Zu einer näheren Diskussion der Hysterese siehe Abschnitt 5.3.3

5.2.1 Verhalten bei niedrigem Druck

Betrachet man zuerst den Bereich niedrigen Drucks, liegt bei tieferen Temperaturen eine geordnete LC-Phase mit Neigung zu nächsten Nachbarn vor.
Dieser scheinbare Widerspruch zu den früheren Grundzustandsanalysen mag daran liegen, daß diese Analysen nicht alle dem System zur Verfügung stehenden Freiheitsgrade zuließen.

Die im Snapshot 5.1 abgebildeten Eigenschaften der LC-Phase – hexagonale Ordnung des Kopfgitters, kollektive Neigung und dichte Packung der Ketten – spiegeln sich auch in den quantitativen Größen wieder.

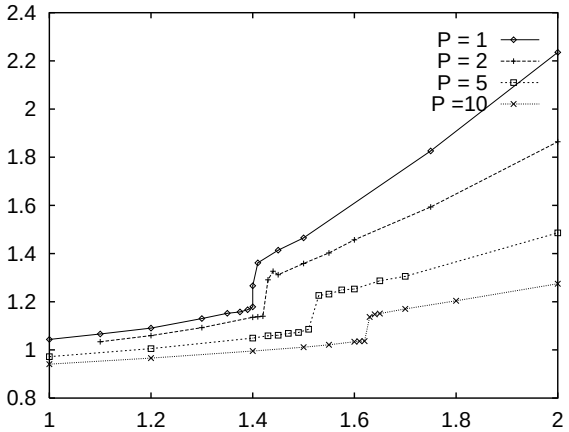


Abbildung 5.2: Fläche A/n vs. T für die Drücke $\Pi = 1, 2, 5, 10$

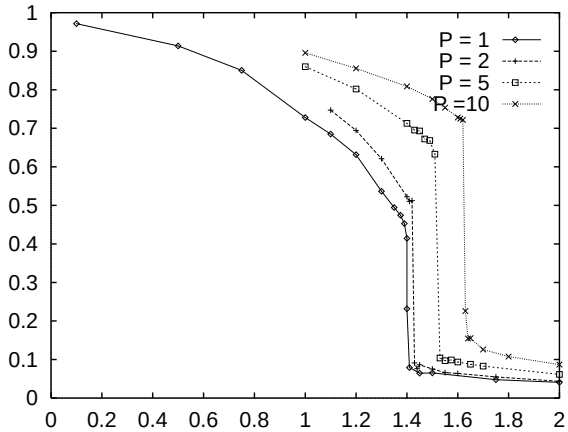


Abbildung 5.3: Ordnungsparameter Ψ_6 vs. T für die Drücke $\Pi = 1, 2, 5, 10$

Man erkennt den Übergang von der LC-NN- in die LE-Phase und die Verschiebung der Übergangstemperatur mit steigendem Druck. Der Flächen- bzw. Dichtesprung am Übergang wird mit steigendem Druck weniger ausgeprägt.
Da auch in der LC-Phase das System bei höheren Drücken eine geringere Fläche aufweist, sind die Kopfgruppen dichter gepackt und der hexagonale Ordnungsparameter Ψ_6 (vgl. (2.4.9)) näher an 1.

Vergleicht man die Simulationsergebnisse quantitativ (vgl. Abschnitt 2.2) mit experimentellen Meßwerten [44][39] ergibt sich befriedigende Übereinstimmung.
In der LC-Phase liegt die Fläche pro Molekül bei 0.87...1.18, was 13...17Å^2 entspricht. Typische Monoschichten einkettiger Fettsäuren haben in der LC-Phase Flächen von 18...20 Å^2/Kette.
Ein Druck $\Pi = 1$ ergibt umgerechnet 23 mN/m. Da experimentell nicht mehr als ca. 60 mN/m erreichbar sind, erscheinen Simulationsdrücke $\Pi > 3$ als fragwürdig, jedoch werden sich erst deutlich darüber interessante Effekte wie der Übergang zwischen NN- und NNN-Neigung zeigen.
Die Übergangstemperatur von LC-NN nach LE liegt für $\Pi = 1 \mathrel{\hat{=}} 23$ mN/m relativ realistisch bei $T = 1.40 \mathrel{\hat{=}} 64°$C.

Während die Richtungen der Moleküldirektoren in der LC-Phase völlig korreliert sind, zeigt der Parameter K_{NN} (vgl. 2.4.10), daß diese Ordnung in der LE-Phase verloren geht. Der mittlere Neigungswinkel ϑ fällt am Übergang um einige Grad ab. In der ungeordneten Phase steigt ϑ mit wachsender Fläche pro Molekül an, in der LC-Phase ergibt sich je nach Druck ein leichter Anstieg oder Rückgang des Neigungswinkels bei zunehmender Temperatur.

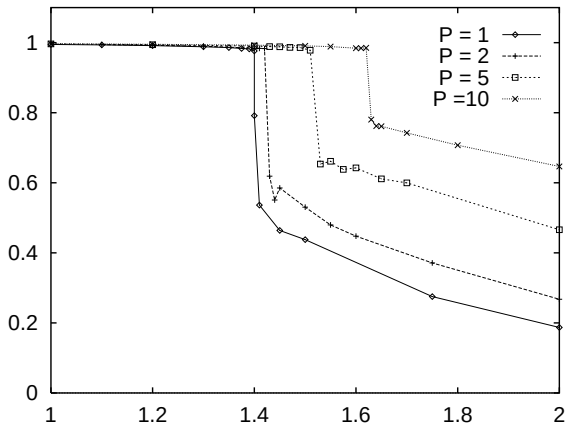


Abbildung 5.4: Neigungskorrelation K_{NN} vs. T für die Drücke $\Pi = 1, 2, 5, 10$

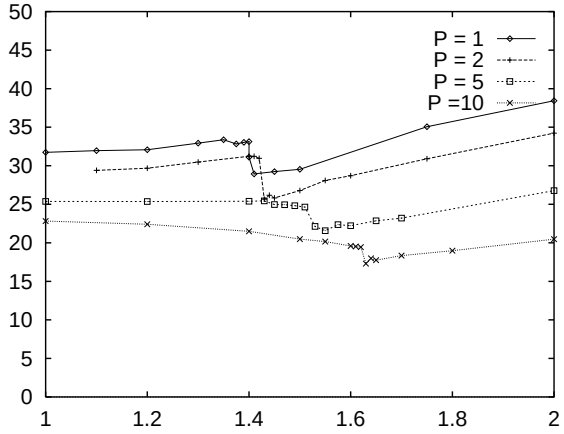


Abbildung 5.5: Neigungswinkel ϑ [°] vs. T für die Drücke $\Pi = 1, 2, 5, 10$

Mit $\langle\vartheta\rangle \approx 30°$ entsprechen die simulierten Neigungswinkel gut den experimentell gefundenen Werten [64].

Die hexagonale Packung des Kopfgitters bei kollektiver Neigung der Ketten und der Verlust beider Ordnungen beim Übergang von der LC-NN- in die LC-U-Phase spiegelt sich auch in den Voronoi-Konstruktionen[4] zu den Kopfgittern wieder.

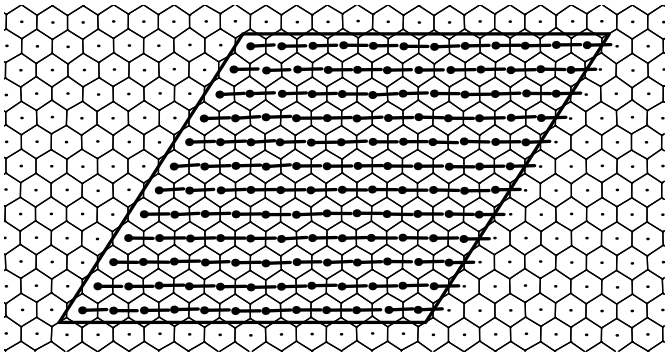


Abbildung 5.6: Voronoi-Konstruktion zur LC-NN-Phase $\Pi = 1, \quad T = 0.1$

Neben der Simulationsbox sind zusätzlich die Kopfgruppen und die Projektionen der End-zu-End-Vektoren der Ketten dargestellt.
Das Kopfgitter außerhalb der Simulationsbox entsteht durch die periodischen Randbedingungen.

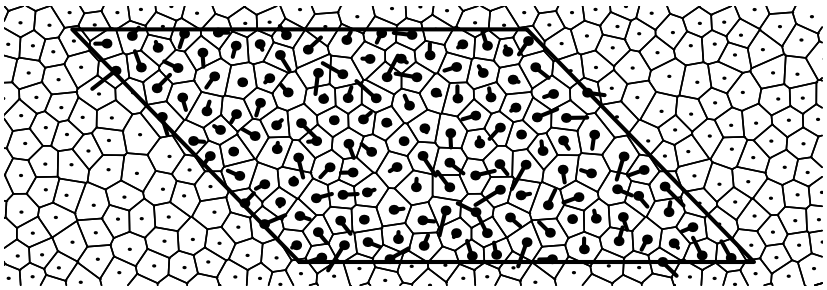


Abbildung 5.7: Voronoi-Konstruktion zur LE-Phase $\Pi = 1, \quad T = 2.0$

Die Ketten sind bzgl. ϑ und Richtung zufällig geneigt.
Im Gegensatz zur LC-Phase treten auch Voronoi-Zellen mit Koordinationszahlen 4,5,7 usw. auf.

[4]Die Voronoi-Konstruktion ist die Verallgemeinerung der Wigner-Seitz-Zelle auf nicht-periodische „Gitter“ [6].

Eine Untersuchung der radialen Paarverteilungsfunktionen für verschiedene Temperaturen zeigen die beiden folgenden Abbildungen. Als Temperaturen wurden in der LC- und LE-Phase je ein typischer Wert und ein Wert nahe der Phasengrenze gewählt.

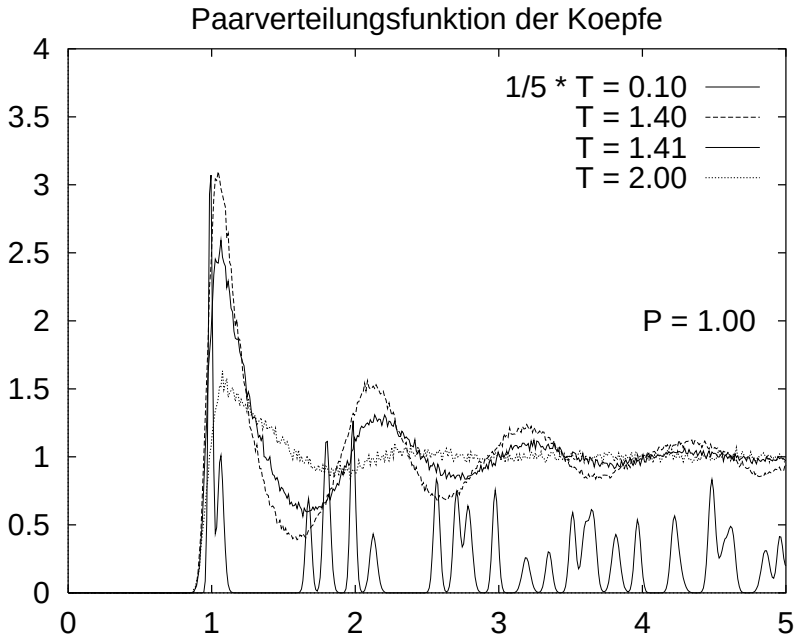


Abbildung 5.8:
radiale Paarverteilungsfunktion $g(r)$ der Kopfgruppen bei $\Pi = 1$ für die Temperaturen $T = 0.10, 1.40, 1.41, 2.00$

Die Kurve zu $T = 0.1$ wurde aus Maßstabsgründen mit 1/5 skaliert.

Aus der Verzerrung des Gitters entlang einer Raumrichtung resultiert die Verdoppelung der Strukturen gegenüber dem hexagonalen Gitter aus Abschnitt 2.5.1

Auch bei $T = 1.41$, d.h. jenseits des Übergangs in die LE-Phase ist noch eine 3. Nachbarschaftsschale zu erkennen, die bei $T = 2.00$ gänzlich verschwunden ist.

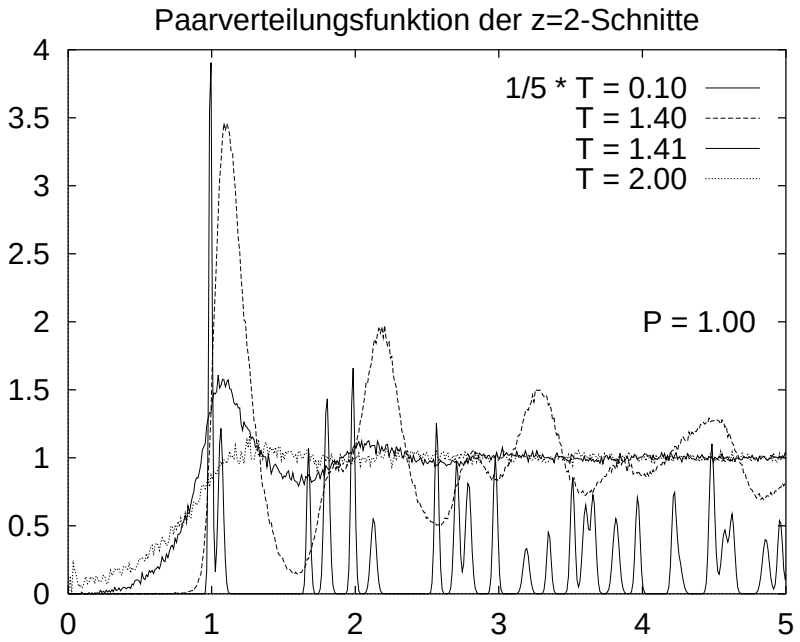


Abbildung 5.9:
radiale Paarverteilungsfunktion $g(r)$ der $z = 2$-Durchstoßpunkte bei $\Pi = 1$ für die Temperaturen $T = 0.10, 1.40, 1.41, 2.00$

Die Ebene z=2 schneidet die Ketten etwa zwischen dem 4. und 5. Monomer einschließlich der Kopfgruppe.

Bei $T = 0.10$ ergibt sich das gleiche Bild wie bei den Kopfgruppen, d.h. auch die Schnittpunkte mit der $z = 2$-Ebene bilden eine verzerrtes hexagonales Gitter.

Im Unterschied zur Verteilungsfunktion der Köpfe ist hier auch schon bei $T = 1.41$ nahezu jede Ordnung verschwunden.
Da die größeren Köpfe die $z = 0$-Ebene nicht verlassen dürfen behalten sie länger eine relativ dichte Packung bei als die kleineren Ketten-Monomere, zumal der Schnittpunkt mit der $z = 2$-Ebene von der Konfiguration der ganzen Kette abhängt.

5.2.2 Verhalten bei hohem Druck

Das Regime hohen Drucks ist dadurch gekennzeichet, daß bei sehr tiefen Temperaturen NNN-Neigung vorliegt, die mit steigender Temperatur in eine ungeneigte Phase LC-U übergeht in der die Ketten kollektiv senkrecht auf der Grenzfläche stehen. Bei noch höheren Temperaturen folgt schließlich der Übergang in die ungeordnete LE-Phase.

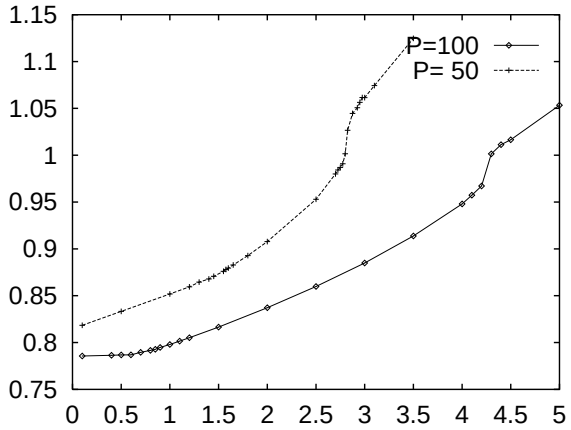


Abbildung 5.10: Fläche A/n vs. T für die Drücke $\Pi = 50, 100$

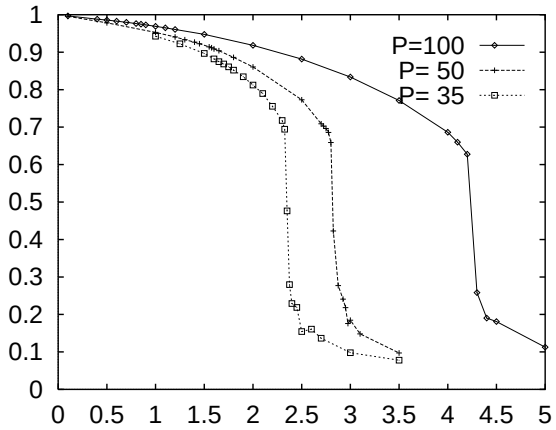


Abbildung 5.11: Ordnungsparameter Ψ_6 vs. T für die Drücke $\Pi = 35, 50, 100$

Insbesondere bei $\Pi = 100$ erkennt man neben dem Flächenänderung am Übergang LC-U↔LE auch einen Knick in der thermischen Flächenausdehnung zwischen der geneigten und der ungeneigten LC-Phase. Der hexagonale Ordnungsparameter Ψ_6 fällt mit steigender Temperatur gleichmäßig weiter bis zum Erreichen der LE-Phase. Die zunehmende thermische Unordnung des Kopfgruppengitters wirkt sich also stärker aus, als die Verzerrung des Gitters, die aufgrund der zurückgehenden Neigung verschwindet.

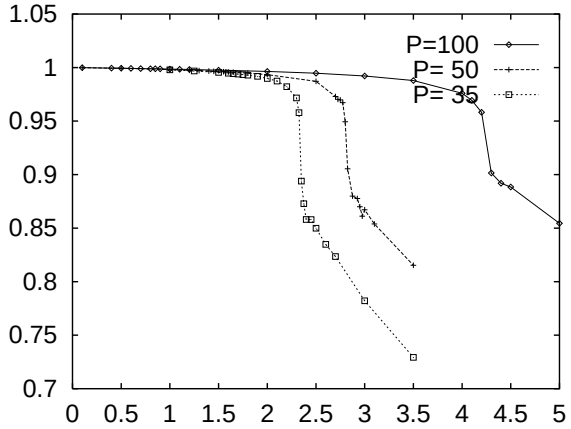


Abbildung 5.12: Neigungskorrelation K_{NN} vs. T für die Drücke $\Pi = 35, 50, 100$

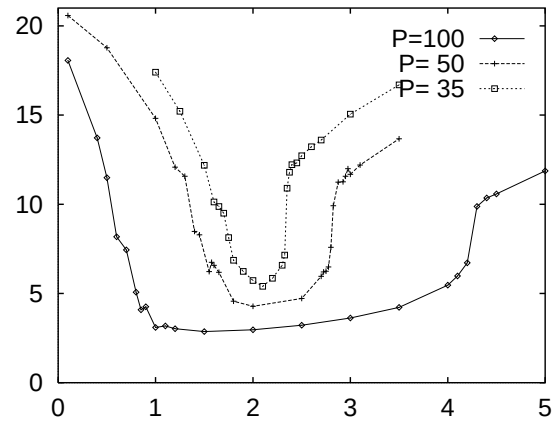


Abbildung 5.13: Neigungswinkel ϑ [°] vs. T für die Drücke $\Pi = 35, 50, 100$

Die Neigungskorrelation K_{NN} ist vom Verschwinden der Neigung unbeeinflußt, da die

Moleküldirektoren in der LC-U-Phase zwar ungeneigt, aber immer noch parallel zueinander ausgerichtet sind.
Im Verlauf des mittleren Neigungswinkels $\langle\vartheta\rangle$ zeigt sich deutlich das Zurückgehen der Neigung bis eine nahezu ungeneigte Phase erreicht ist. In dieser Phase wächst $\langle\vartheta\rangle$ aufgrund der steigenden thermischen Unordnung leicht an, bis schließlich beim Übergang in die LE-Phase ein Sprung zu deutlich größeren, unkorrelierten Neigungswinkeln eintritt.

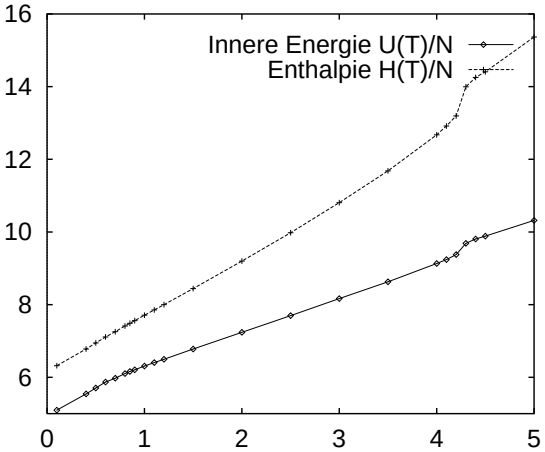


Abbildung 5.14: U/N und H/N vs. T für Druck $\Pi = 100$

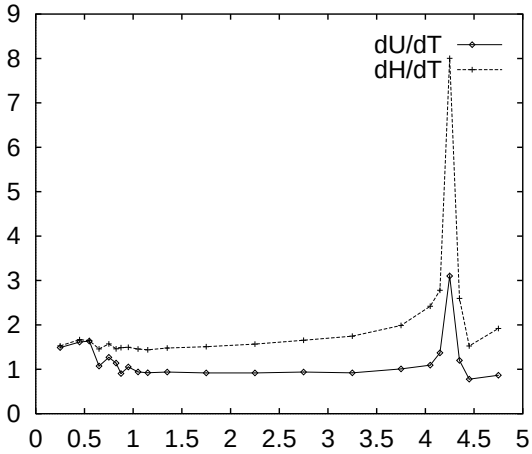


Abbildung 5.15: spez. Wärmekapazitäten $\partial U/N\partial T$ und $\partial H/N\partial T$ vs. T für $\Pi = 100$

Für $\Pi = 100$, einem charakteristischen Vertreter des Hochdruck-Regimes, weist der Verlauf der inneren Energie $U(T)/N$ eine Knick am Übergang zur ungeneigten Phase auf, der sich auch im Differenzenquotienten $\Delta U/\Delta T$ auswirkt. Die spez. Enthalpie $H(T)/N$ hingegen besitzt diese Verhalten nicht, sondern verläuft mit konstanter Steigung über den Übergang hinweg.

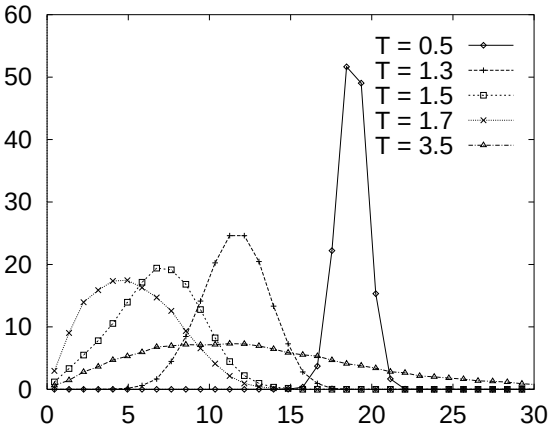


Abbildung 5.16: Neigungshistogramm $\rho(\vartheta[^\circ])$ bei $\Pi = 50$ für $T = 0.5, 1.3, 1.5, 1.73.5$

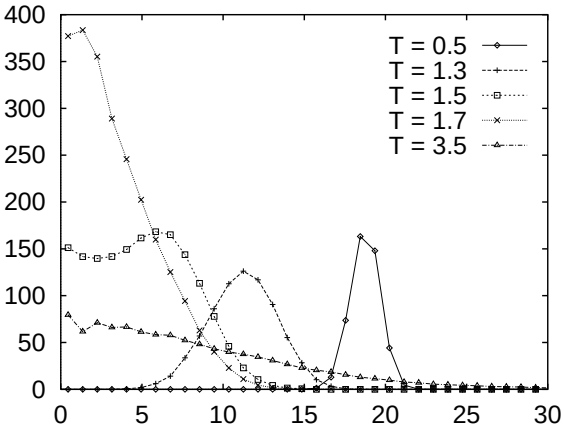


Abbildung 5.17: gew. Histogramm $\varrho(\vartheta[^\circ])$ bei $\Pi = 50$ für $T = 0.5, 1.3, 1.5, 1.73.5$

Die Verteilung $\rho(\vartheta[^\circ])$ des Neigungswinkels für verschiedene Temperaturen zum

Druck $\Pi = 50$ zeigt die Verschiebung zu kleineren Winkeln mit steigender Temperatur. Aus geometrischen Gründen wird stets $\rho(0) = 0$ sein.
Denkt man an ein System unabhängiger starrer Stäbe, die sich frei orientieren können, wird deren Neigungsverteilung die Abhängigkeit $\rho(\vartheta) \sim \sin\vartheta$ besitzen. Das gewichtete Histogramm $\varrho(\vartheta) := \rho(\vartheta)/\sin\vartheta$ ist mit eben dieser reziproken Verteilung reskaliert, so daß für völlig freie Stäbe $\varrho(\vartheta) =$ const. erwartet wird[5].

Bringt man das System durch niedrigen Druck und hohe Temperatur in die Gasphase (vgl. Abschnitt 5.3.4), erhält man eine Verteilung $\varrho(\vartheta)$ die über den *ganzen* Winkelbereich 0°...90° deutlich von null verschieden und im Bereich 0°...30° annähernd konstant ist.

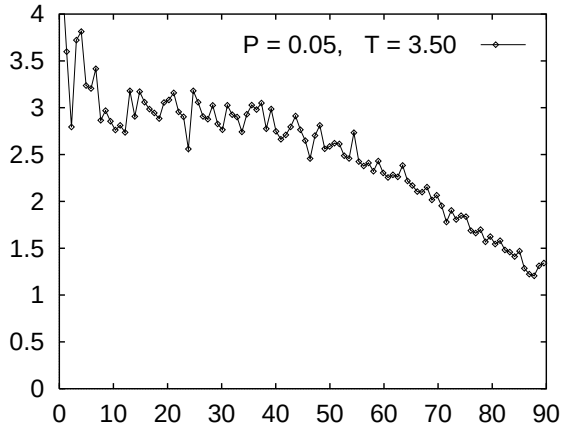


Abbildung 5.18: gew. Histogramm $\varrho(\vartheta[°])$ bei $\Pi = 0.05$, $T = 3.5$

Der mittlere Neigungswinkel $\langle\vartheta\rangle$ allein unterscheidet nicht zwischen geringer, aber kollektiver Neigung und zufällig um die Flächennormale $\vartheta = 0$ streuender Neigung. Die Größe $\varrho(0)$ scheint hierfür jedoch ein empfindlicher Indikator zu sein.
Der Verlauf von $\langle\vartheta(T)\rangle$ bei $\Pi = 50$ (siehe Abbildung 5.13) deutet auf ein Verschwinden der kollektiven Neigung im Bereich $T \approx 1.5$ hin. Während sich bei niedrigeren Temperaturen $\varrho(\vartheta)$ eine relativ schmale Verteilung um ein Maximum bei $\vartheta \neq 0$ ergibt, liegt oberhalb eine breite Verteilung mit Maximum bei $\vartheta = 0$ vor[6].

Als eigentliches Maß der kollektiven Neigung war R_{xy} definiert worden (2.4.12). Diese Größe entspricht der mittleren Komponente des Moleküldirektors $\vec{D}$ parallel zur Grenzfläche.
Abbildung 5.19 zeigt ihren Verlauf für verschiedene Drücke. Während bei $\Pi = 1, 10$ eine geneigte LC-Phase aufschmilzt und sich daher am Übergang ein deutlicher Sprung in R_{xy} ergibt, findet man bei $\Pi = 35, 50, 100$ am Übergang in die ungeneigte LC-U Phase einen stetigen Rückgang auf ein niedriges Plateau, dessen Abweichung von Null von der endlichen Systemgröße herrührt.

[5]Durch die Umgewichtung mit $1/\sin\vartheta$ geht die Normierung verloren, weswegen die Flächen unter den Kurven zu verschiedener Temperatur nicht mehr gleich groß sind.

[6]Daß auch bei $T = 3.5$ (Abb. 5.18) die Größe $\varrho(\vartheta)$ im Bereich 0°...30° deutlich abfällt liegt daran, daß das System aufgrund des hohen Drucks eine relativ große Dichte aufweist und noch weit vom Grenzfall unabhängiger Ketten entfernt ist.

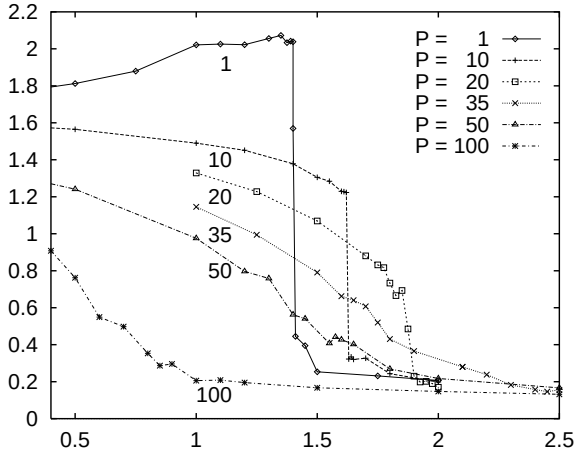


Abbildung 5.19: Neigungsmaß R_{xy} vs. T für die Drücke $\Pi = 1, 10, 20, 35, 50, 100$

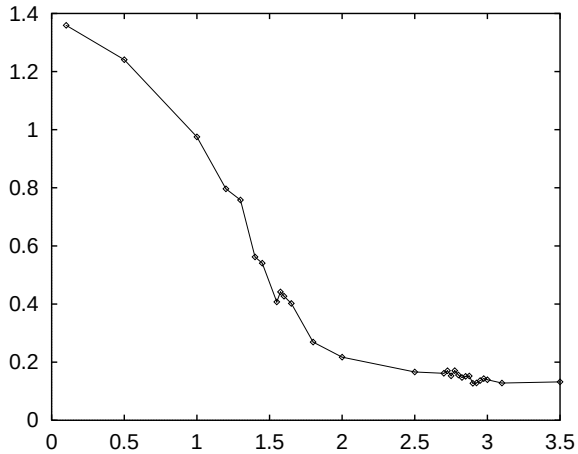


Abbildung 5.20: R_{xy} vs. T bei $\Pi = 50$

Für $\Pi = 50$ (Abb. 5.20) sinkt R_{xy} bei Temperaturen $T \approx 1.5$ von großen Werten, die eine kollektive Neigung nachweisen, deutlich ab. Allerdings ist die Grenze zwischen dem Abfall und dem Plateau relativ ausgedehnt, so daß aus dem Kurvenverlauf keine Übergangstemperatur direkt abgelesen werden kann.

Da auch keine andere Observable zu Verfügung stand, die das Verschwinden der kollektiven Neigung genauer lokalisiert hätte, erfolgte die Bestimmung des Übergangs LC-geneigt $\leftrightarrow$ LC-U anhand von $\varrho(0)$, zumal der Verlauf anderer Größen nie im Widerspruch zu den so bestimmten Werten stand.

Der *Neigungsübergang* zwischen der LC-NN- und der LC-NNN-Phase läßt sich mit der Simulation nur sehr ungenau festlegen; zum einen liegt bei tiefen Temperaturen eine weite Hysterese bezüglich des Drucks vor, zum anderen scheint bei höheren Temperaturen die Energiebarriere zwischen den beiden Neigungsrichtungen rasch zu verschwinden. Zwar ist dann noch der Neigungswinkel $\langle\vartheta\rangle$ aufgrund der Kettenpackung vorgegeben, die Neigungsrichtung φ kann jedoch *frei drehen*, ohne deutliche Bevorzugung der NN- oder NNN-Positionen.
Dieses Verhalten ist bei Annäherung an die LC-U-Phase zu erwarten, da mit verschwindendem Neigungswinkel die Richtung der Neigung keine Bedeutung mehr hat. Es tritt jedoch auch schon bei relativ niedrigen Temperaturen von $T \approx 0.5$ auf.

Zumindest läßt sich für $T = 0$ der Übergangsdruck aus der Grundzustandsiteration bestimmen (vgl. Abschnitt 5.1); für niedrige Temperaturen erlaubt die thermodynamische Integration der Gibbs'schen freien Energie (vgl. Abschnitt 4.5) das Auffinden der Phasengrenze. Nach der Clausius-Clapeyron-Gleichung (4.5.52) kann der Verlauf der Grenzlinie noch darüber hinaus extrapoliert werden.
Aus den Simulationsdaten selbst gelang keine weitere Festlegung die Phasengrenze.

Zur Beschreibung der **Richtung der kollektiven Neigung** bieten sich zwei graphische Darstellungen an.

Einerseits kann man ein Histogramm der Winkel aufnehmen, die die benachbarten Kopfgruppen-Bonds mit der momentanen Neigungsrichtung der Kette bilden. Im Fall der Neigung zu nächsten Nachbarn müßten sich Peaks bei $0°, 60°, 120°$... ergeben, während NNN-Neigung an Peaks bei $30°, 90°, 150°$... zu erkennen ist.
In der ungeordneten LE-Phase erwartet man mit gleicher Häufigkeit alle Winkel zwischen 0° und 360°.

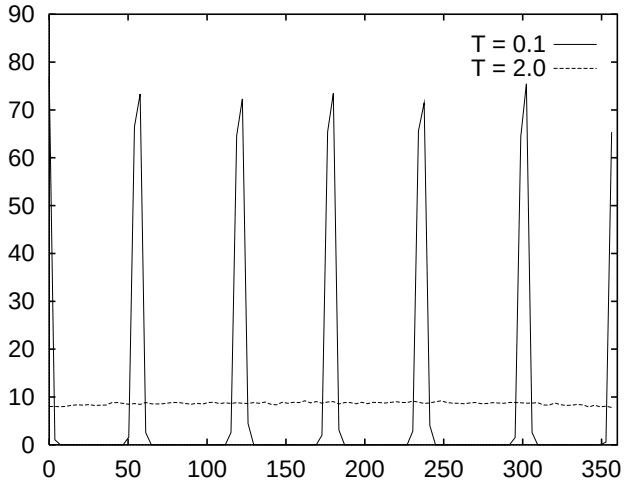


Abbildung 5.21: Verteilung der Bondrichtungen bezogen auf die Neigungsrichtung bei $\Pi = 1$ für $T = 0.1$ (LC-NN) und $T = 2.0$ (LE)

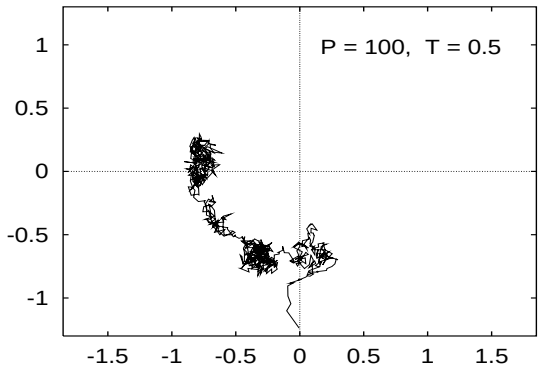


Abbildung 5.22: Trajektorie D_x-D_y über die MC-Schritte

Andererseits kann man die Trajektorie des mittleren Moleküldirektors $[\vec{D}]$ über die MC-Schritte verfolgen.
Dies ergibt ähnliche Information wie das Histogramm des Moleküldirektors [42], ist aber einfacher darzustellen und läßt zusätzlich die „Bewegung“ erkennen.

Zur quantitativen Beschreibung dieses Verhaltens wurde in Anlehnung an den hexagonalen Ordnungsparameter Ψ_6 (2.4.9) der Parameter Φ_6 definiert:

$$\Psi_6 = \left\langle \left| \frac{1}{6n} \sum_{j=1}^{n} \sum_{k=1}^{6} \exp(i6\phi_{jk}) \right|^2 \right\rangle \quad \mapsto \quad \Phi_6 := \left| \left\langle \frac{1}{6n} \sum_{j=1}^{n} \sum_{k=1}^{6} \exp(i6\phi^*_{jk}) \right\rangle \right|^2 \qquad (5.2.3)$$

Dabei unterscheiden sich die ϕ^*_{jk} von den ϕ_{jk} dadurch, daß nun die Neigungsrichtung der Ketten die Rolle der zuvor beliebigen Referenzachse übernimmt[7].
Zusammen mit der Vertauschung der Betrags- $|...|^2$ mit der Mittelwert-Bildung $< ... >$ erkennt man die Verschiedenheit der so charakterisierten Zustände am Beispiel eines

[7] Zur Auswertung wurde zwar die Neigungsrichtung jeder einzelnen Kette i als Bezugsrichtung für ihre 6 Nachbarn $\{j\}$ verwendet, da die Ketten jedoch kollektiv in *eine* Richtung geneigt sind, ist dies zulässig.

Systems, das stets kollektiv geneigt ist, dessen Neigungsrichtung aber gegenüber dem starren, hexagonal geordneten Kopfgitter im Verlauf der Simulation rotiert.
Während bei Ψ_6 die Betragsbildung in jeder Konfiguration erfolgt und somit die Phasen-Information der komplexen Zahl verloren geht, wird in Φ_6 über komplexe Zahlen gemittelt und zuletzt der Betrag dieses Mittelwertes ausgewertet.
Daher wird in jeder Konfiguration $|\Psi_6|^2 = 1$ sein, was zu $\langle|\Psi_6|^2\rangle = 1$ führt, wohingegen die Phase von Φ_6 die Drehung der Neigungsrichtung nachvollzieht und sich im Mittelwert $| < \Phi_6 > |^2 = 0$ ergibt.

Da im Simulationsprogramm bereits die Aufzeichnung eines Histogramms $h(\phi^*_{jk})$ implementiert war, ließ sich Φ_6 berechnen als

$$\Phi_6 = \int_0^{2\pi} \exp(i6\phi)h(\phi)\mathrm{d}\phi \;\Big/\; \int_0^{2\pi} h(\phi)\mathrm{d}\phi \tag{5.2.4}$$

Bei einem unverzerrten Gitter liegen zwischen einer NN- und NNN-Richtung je 30°. Durch den Faktor 6 in der Definition von Φ_6 entspricht dies 180°, weswegen sich die beiden Vorzugsneigungen gegenseitig auslöschen. Möchte man Systeme die zwischen den Vorzugsneigungen hin-und-her-springen von solchen unterscheiden, in denen die Neigungsrichtung *frei* dreht, kann man auch die Größe

$$\Phi_{12} = \int_0^{2\pi} \exp(i12\phi)h(\phi)\mathrm{d}\phi \;\Big/\; \int_0^{2\pi} h(\phi)\mathrm{d}\phi \tag{5.2.5}$$

betrachten, da nun der Winkel zwischen NN und NNN auf $12 \cdot 30° = 360°$ vergrößert wird und geringe Abweichungen von diesen Richtungen zum Verschwinden von Φ_{12} führen.

Der Verlauf von Φ_6 und Φ_{12} entlang der Isothermen $T := 0.5$ zeigt drei Bereiche:

Bei niedrigen Drücken liegt eine deutliche Kopplung der Neigungsrichtung an die Bondrichtungen des Kopfgitters vor. Ab Drücken $\Pi > 30$ geht diese Kopplung verloren und die Neigung kann annähernd frei drehen.
Da bei $T = 0.5$ bereits eine merkliche thermische Unordnung vorliegt, erreichen Φ_6 und insbesondere Φ_{12} auch bei niedrigem Druck nicht den Idealwert 1.

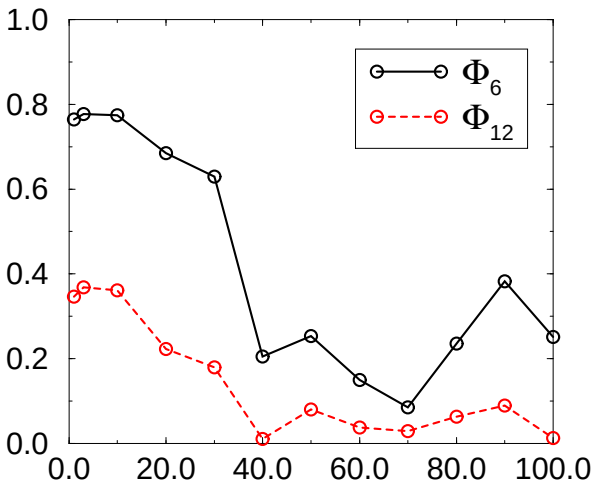


Abbildung 5.23: Φ_6 und Φ_{12} vs. Druck Π bei $T = 0.5$

Der erneute Anstieg bei Drücken oberhalb von $\Pi > 70$, der mit einer Stabilisierung der NN-Richtung einhergeht, hängt mit einem *Einrast*-Effekt zusammen, den auch *H.O. Lange* [42] beobachtet hat. Besonders deutlich wird dieser in der vertikalen Dichteverteilung der Monomere, die eine Aufspaltung der oberen Monomerlagen in je zwei Schichten aufweist. Hierbei liegen die Monomere einer Kette in den „Lücken“ der Nachbarketten.

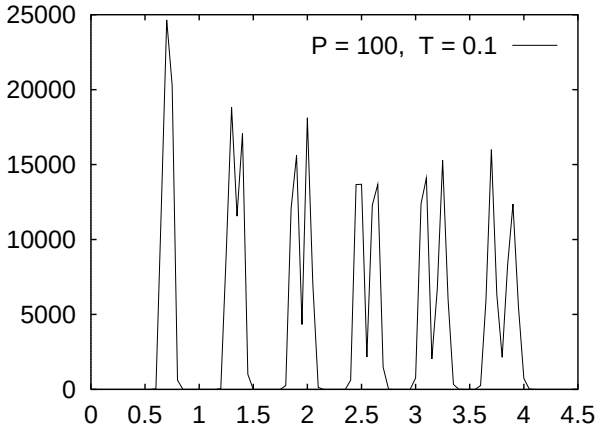


Abbildung 5.24: vertikale Dichteverteilung $\rho(z)$ bei Druck $\Pi = 100$, $T = 0.5$

Da der *Einrast*-Effekt ein Modell-Artefakt ist, das von der Verwendung effektiver Monomere und evtl. von einem zu weichen Bondlängen-Potential herrührt, wird das erneute Auftreten der NN-Phase bei sehr hohen Drücken im folgenden vernachlässigt und davon ausgegangen, daß bei Drücken oberhalb von $\Pi > 30$ und Temperaturen ab $T \gtrsim 0.5$ eine Entkopplung der Neigungsrichtung vom Kopfgitter besteht.

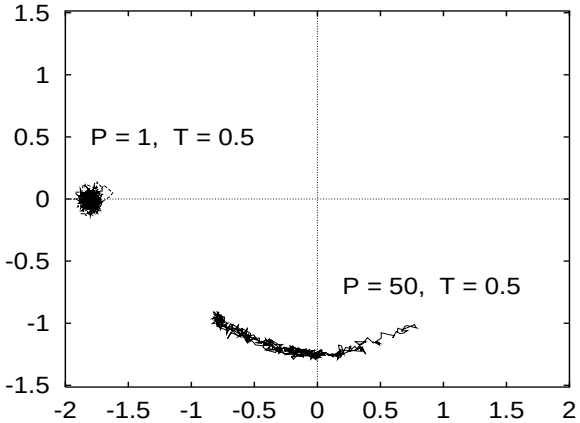


Abbildung 5.25: Trajektorien D_x-D_y über MCS bei Druck $\Pi = 1$, $T = 0.5$ und $\Pi = 50$, $T = 0.5$

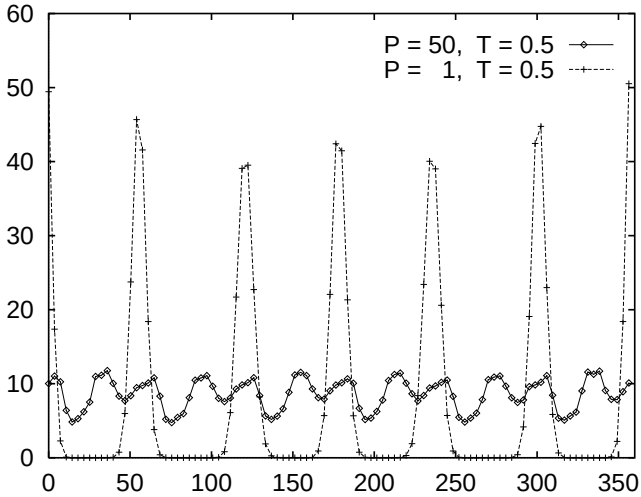


Abbildung 5.26: Verteilung der Bondrichtungen bezogen auf die Neigungsrichtung bei $T = 0.5$ für $\Pi = 1$ (NN) und $\Pi = 50$

Während bei niedrigem Druck eine stabile Neigung zu nächsten Nachbarn vorliegt, was sich darin ausdrückt, daß der Moleküldirektor über einem Punkt in der Ebene verweilt[8] und das Histogramm ausgeprägte Maxima bei $0°, 60°, 120°$... besitzt, zwischen

[8] Da die NN- und NNN-Richtungen jeweils 6-fach entartet sind, müßte die Simulation auch bei stabiler NN-Neigung zwischen diesen sechs Positionen springen, wenn das System den ganzen Konfigurationsraum ausnützt. Ein solcher spontaner Wechsel wurde innerhalb der praktikablen Simulationslängen nie beobachtet.

denen es auf null abfällt, besteht bei $\Pi = 50$ keine bevorzugte Neigungsrichtung. Die Trajektorie des Moleküldirektors verläuft etwa entlang einer Kreislinie, was in der Projektion einem konstanten Neigungswinkel $\langle\vartheta\rangle$ entspricht und das Histogramm fällt an keiner Stelle mehr auf null ab. Die verbliebene Modulation im Histogramm wird mit längeren Simulationsläufen (hier 300.000 MCS) verschwinden.

5.2.3 Phasendiagramm

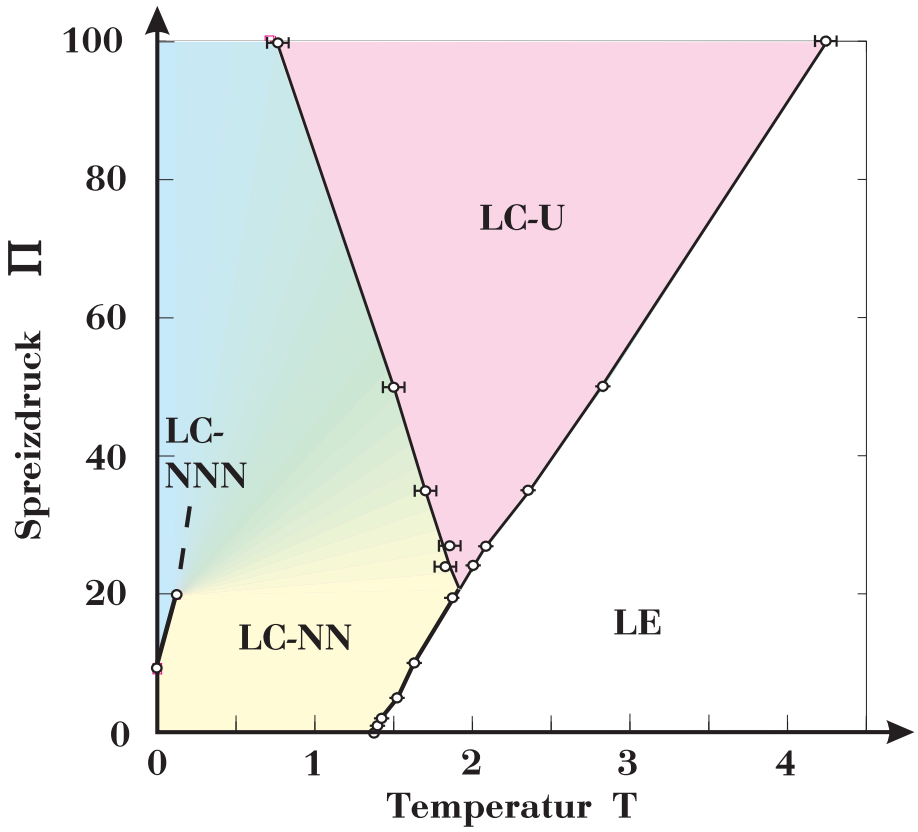


Abbildung 5.27: Phasendiagramm $\Pi(T)$ für $\sigma_H := 1.1\sigma$

Die Phasen, die für das System kleiner Köpfe $\sigma_H := 1.1\sigma$ bestimmt wurden, sind in obigem Phasendiagramm zusammengefaßt.

Der Richtungsübergang zwischen der NN- und NNN-Phase ist bei tiefen Temperaturen 1. Ordnung, da eine weite Hysterese besteht und die Symmetrie zwischen beiden Phasen diskontinuierlich wechselt.
Bei höheren Temperaturen deutet die Entkopplung der Neigungsrichtung vom Kopfgitter darauf hin, daß ein kontinuierlicher Übergang zwischen NN- und NNN-Richtung möglich ist, was man zusammen mit der verschwindenden Hysterese als eine zusätzliche, geneigte LC-Phase ohne *tilt order* interpretieren kann, die jeweils durch einen Übergang 2. Ordnung von LC-NNN bzw. LC-NN getrennt ist.

Zwischen der LC-NN- und der LE-Phase liegt ein Schmelzübergang 1. Ordnung, da hier zugleich zwei unabhängige Symmetrien – die hexagonale Ordnung der Köpfe und die Kollektivität der Neigung – entlang einer ausgedehnten Linie in der ΠT-Ebene gebrochen werden [41].
Wie bei einem Übergang 1. Ordnung zu erwarten fällt z.B. der hexagonale Ordnungsparameter Ψ_6 beim Aufheizen von der LC-NN- in die LE-Phase auf einen Wert nahe

null ab und Größen wie die Enthalpie oder die Fläche verlaufen dort ebenfalls unstetig.

Da die kollektive Neigung stetig gegen null geht und auch in keiner anderen Observablen ein Sprung beobachtet wird, ist davon auszugehen, daß der Neigungsübergang zwischen der kollektiv geneigten und der ungeneigte LC-Phase 2. Ordnung ist.

Über die Charakterisierung des Schmelzübergangs bei höherem Druck, der aus der ungeneigten LC-U-Phase in die LE-Phase erfolgt, ist keine ähnlich eindeutige Aussage möglich.
Da hier in der geordneten Phase keine Symmetriebrechung aufgrund der Neigung besteht, geht an der Phasengrenze nur die hexagonale Ordnung verloren, so daß auch ein Übergang 2. Ordnung vorliegen kann.

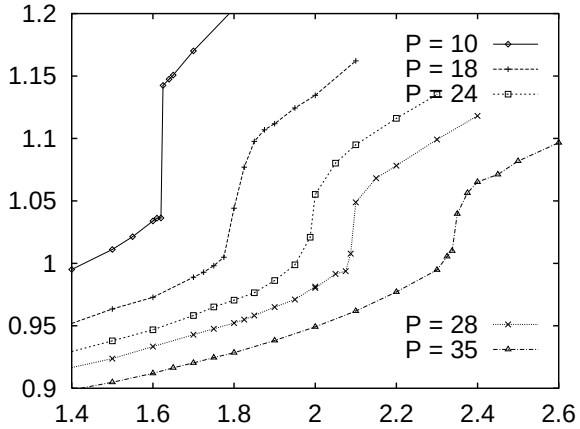


Abbildung 5.28: Fläche A/n vs. T
für die Drücke $\Pi = 10, 18, 24, 28, 35$

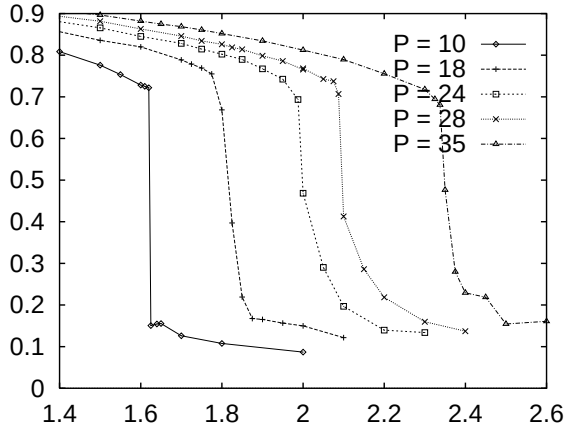


Abbildung 5.29: Ψ_6 vs. T
für die Drücke $\Pi = 10, 18, 24, 28, 35$

Sowohl die Fläche A/n pro Monomer, als auch die spezifische innere Energie U/N zeigen zwar einen Wechsel zwischen zwei annähernd linearen Verläufen innerhalb der beiden Phasen, doch erscheint der Übergangsbereich, ähnlich wie der Abfall von Ψ_6, deutlich stärker aufgeweitet als beim Übergang aus der geneigten LC-Phase, z.B. bei $\Pi = 10$.

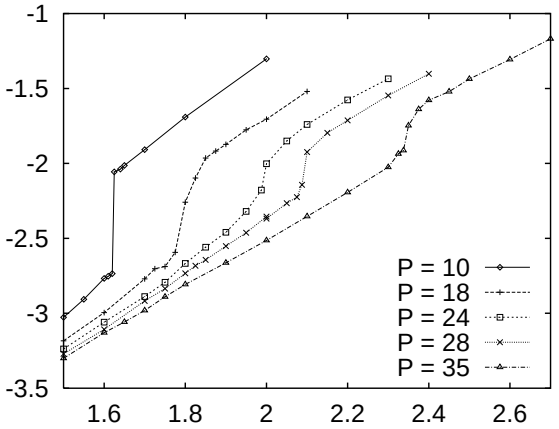


Abbildung 5.30: U/N vs. T
für die Drücke $\Pi = 10, 18, 24, 28, 35$

Eine Untersuchung der Hysterese (vgl. Abschnitt 5.3.3) des entsprechenden Übergangs am System mit $\sigma_H := 1.2\sigma$ konnte ebenfalls nicht zur Klärung beitragen.

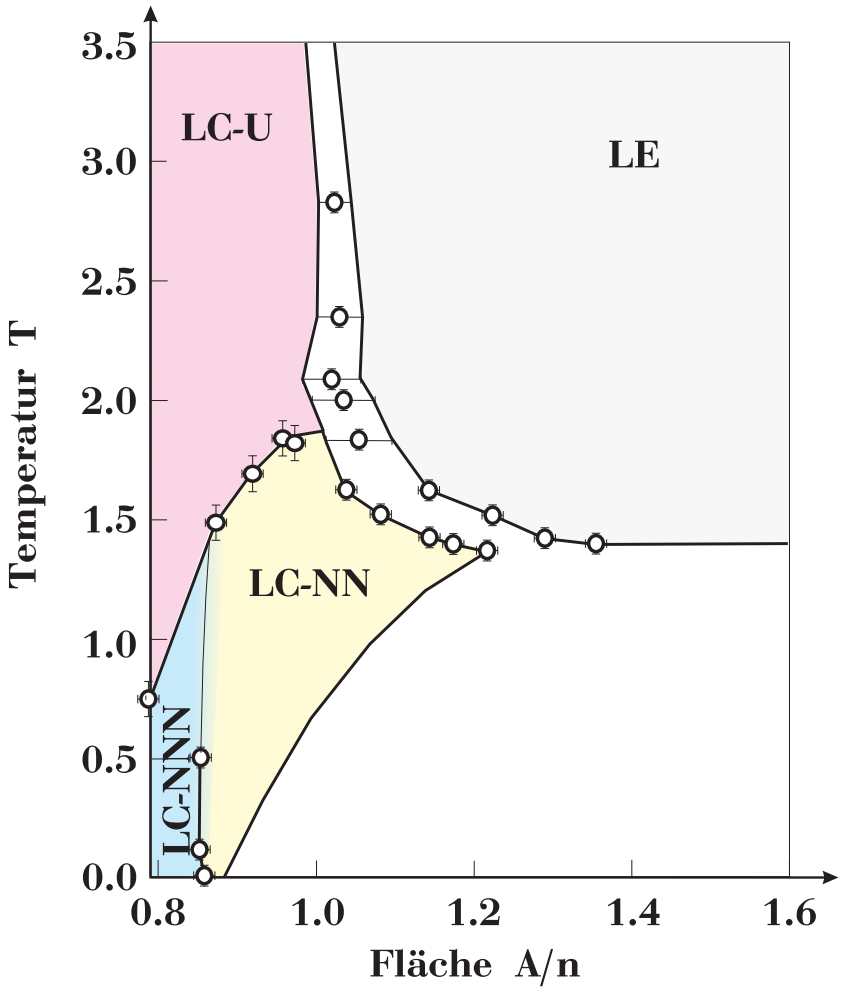


Abbildung 5.31: Phasendiagramm $T(A/n)$ für $\sigma_H := 1.1\sigma$

Eine deutlich andere Darstellung der Lage der Phasen zueinander erhält man, wenn man die Grenzlinien nicht als $\Pi(T)$, sondern als $T(A)$ aufträgt.

Der Bereich der Fläche A/n pro Molekül ist nach unten bei ca. 0.8 begrenzt, da noch höhere Dichten nur bei extrem hohen Drücken auftreten und bereits hier Einrast-Effekte einsetzen. Eine gewisse Schwierigkeit dieser Auftragung im Gegensatz zum $\Pi(T)$-Phasendiagramm liegt darin, daß Π und T als Simulationsparameter frei gewählt werden können, die Fläche aber nur indirekt als $A(\Pi, T)$ beeinflußbar ist[9].

Das Phasendiagramm zeigt die NN- und NNN-Phase als zwei *getrennte* LC-Bereiche. Da bei Temperaturen oberhalb von $T \gtrsim 0.5$ die Neigungsrichtung vom Kopfgitter ent-

[9] Da die Simulationsdaten hauptsächlich als Isobaren aufgenommen wurden, liegen die Datenpunkte hier nicht optimal.

koppelt, unterliegt die eingetragene Grenzlinie einer gewissen Willkür, aber man erkennt die relativ geringe Größe des NNN-Gebiets im Vergleich zu den anderen Phasen. Betrachtet man den mittleren Neigungswinkel $\langle\vartheta\rangle$ an der Grenze der Phasen, stellt man fest, daß er in beiden etwa gleich groß ist und mit $20°...24°$ gut mit experimentellen Werten [67][64] übereinstimmt.

Das Gebiet der LC-NN-Phase ist zu größeren Flächen hin durch den Verlauf von $A(\Pi := 0, T)$ begrenzt, der entlang der entsprechenden Isobaren „gemessen“ wurde. Da bei $\Pi = 0$ bei allen Temperaturen die Gasphase aus Entropie-Gründen stabil ist, stellt diese Linie die Koexistenz einer LC-Phase mit einer extrem verdünnten Gasphase dar.

An der Grenze zwischen LC-U und LE ist an der Übergangstemperatur jeder der aufgenommenen Isobaren zwar nur ein Datenpunkt eingetragen, dessen Fehlerbalken und der Verlauf der (doppelten) Grenzlinie geben aber etwa die Flächen wieder, die LC-U und LE am Phasenübergang bzw. in dessen unmittelbarer Nähe haben.

5.3 System großer Köpfe

5.3.1 Modulation

Im System großer Köpfe, d.h. $\sigma_H = 1.2\sigma$, tritt bei Konfigurationen mit Neigung zu nächsten Nachbarn eine (unerwartete) Abweichung von der „konventionellen“ Anordnung der Ketten auf.
Diese **Modulation**, die bei NNN-Phasen nie beobachtet wurde, hat bezüglich Druck, Temperatur und Kopfgröße einen gewissen Existenzbereich. Sie stellt sich sowohl bei rechteckiger, als auch bei scherbarer Simulationsbox ein, reproduziert die 6-fache Entartung der NN-Richtung und steht in Zusammenhang mit dem Größen-Mißverhältnis zwischen den Köpfen und dem Durchmesser des Alkanschwanzes.
Während bei den kleinen Kopfgruppen die kollektive Neigung mit *tilt order* einherging, d.h. mit einer starken Korrelation zwischen der einheitlichen Neigungsrichtung der Ketten und den Bonds des Kopfgitters, ist in Systemen mit sehr großen Köpfen (vgl. Abschnitt 5.4), bei denen Cluster-Bildung einsetzt, die einheitliche Neigungsrichtung verloren gegangen. Die hier beschriebenen modulierten Phasen bilden eine Zwischenstufe in dem Sinn, daß zwar starke Korrelation zwischen Neigung und Kopfgitter vorliegt, die Richtung aber innerhalb der Box kontinuierlich variiert.

Einen guten Eindruck von der Anordnung der Ketten in einer modulierten Phase gewinnt man aus der Projektion ihrer Bonds in die Ebene. Das Vorhandensein der Modulation ist hingegen in der Projektion der End-zu-End-Vektoren besser zu erkennen, weshalb letzterer im weiteren der Vorzug gegeben wird.

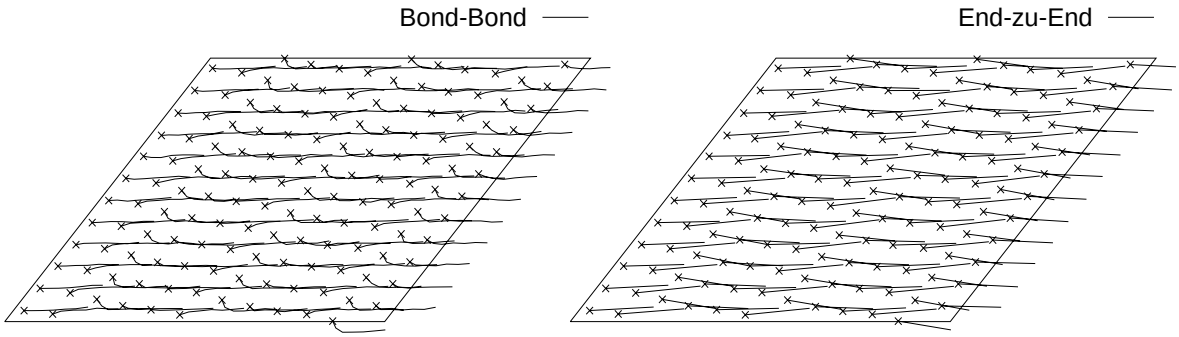


Abbildung 5.32: Projektion der Endkonfiguration zu $\Pi = 1$, $T = 0.1$, $\sigma_H = 1.2\sigma$
Zusätzlich sind die Kopfgruppen und die Simulationsbox markiert.
Die Konfiguration wurde als „konventionelle“ NN-Neigung in x-Richtung aufgesetzt.

Eine modulierte Konfiguration setzt anscheinend relativ flexible Ketten voraus, da diese deutlich von der gestreckten Anordnung abweichen. Betrachtet man nur die Kopfgruppen, fällt auf, daß das Kopfgitter trotz der starken Kettenneigung kaum verzerrt ist, und sich die Köpfe so umgeordnet haben, daß nun eine Gitterachse um ca. 10° gegen die x-Richtung der Simulationsbox verdreht ist (siehe auch Abb. 5.67).
Iteriert man für $\Pi = 1, T = 0$ je ein konventionell und ein moduliert geneigtes System

gemäß (5.1.2) findet man, daß die spezifische Grundzustands-Enthalpie der modulierten Konfiguration, die bei $T = 0$ mit der Gibbs'schen freien Energie übereinstimmt, um 0.04 pro Monomer niedriger ist und damit die stabile Phase darstellt.
In x– bzw. ξ-Richtung liegt eine Überstruktur vor, die die Translationsymmetrie des Gitters bricht, wogegen in y- bzw. η-Richtung die Periodizität unverändert bleibt. Es liegt nahe, die Überstruktur anhand ihrer „Streifenbreite" zu charakterisieren, in diesem Beispiel – wenn man die periodischen Randbedingungen berücksichtigt – als 4+4+4.

Bei tiefen Temperaturen gelingt es der Simulation kaum, ein einmal erreichtes Streifen-System wieder umzulagern. Daher ist davon auszugehen, daß die erzeugten Konfigurationen metastabil sind und es stellt sich die Frage, welche Streifenanordnung optimal ist.
Insbesondere ist mit Kommensurabilitäts-Effekten zu rechnen, da die möglichen Abfolgen der Streifen von der Größe der Systems abhängen.

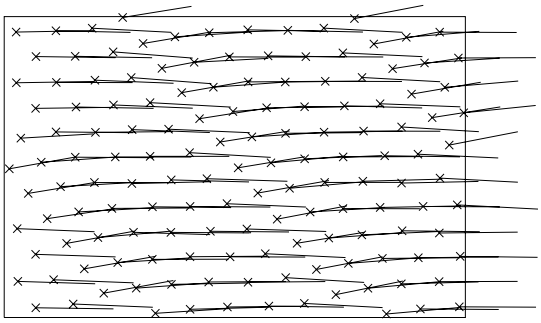


Abbildung 5.33:
Konfiguration zu $\Pi = 1$, $T = 0.1$, $\sigma_H = 1.2\sigma$
Beleg einer Modulation bei rechteckiger Box

Um dies zu klären wurde für verschiedene Systemgrößen jeweils bei $\Pi = 1, T = 0.1$ die spezifische Enthalpie H/N bestimmt und mit dem vorgefundenen Streifen-System verglichen.

Streifen	n	H/N
4+4	8×8	-4.8445
5+3	8×8	-4.8403
4+4+4	12×12	-4.8445
6+6	12×12	-4.8378
7+5	12×12	-4.8374
5+4+4	13×13	-4.8435
6+3+4	13×13	-4.8411
5+3+4+4	16×16	-4.8429
6+5+5	16×16	-4.8394

Die korrekte Größe, die zu bestimmen wäre, ist das Gibbs'sche Potential $G = H - TS$. Da in allen verglichenen Systemen ähnliche Ordnung vorliegt und die Temperatur niedrig genug gewählt wurde, erscheint es aber zulässig, die Entropie hier zu vernachlässigen. Im Gegensatz zu den konventionell geneigten Systemen gelang es leider nicht, beliebig modulierte Anordnungen als Startkonfiguration zu konstruieren.
Nach nebenstehender Tabelle scheint es eine Präferenz für die Streifenbreite **4** zu geben.

Da für das Zustandekommen der Modulation die Beweglichkeit der Kopfgruppen untereinander entscheidend ist, liegt es nahe zu untersuchen, was geschieht, wenn man die Kopfgruppen von den individuellen Monomer-Moves ausnimmt und sie nur noch an den kollektiven Volumen-Moves teilnehmen läßt. Ein Vergleich – jeweils bei $\Pi = 1, T = 0.1$ – dieser eingeschränkten Simulation (A), die zu einer konventionell NN-geneigten Phase führt, mit einer normalen Simulation (B), die zu einer (4+4+4)-Modulation führt, ergibt:

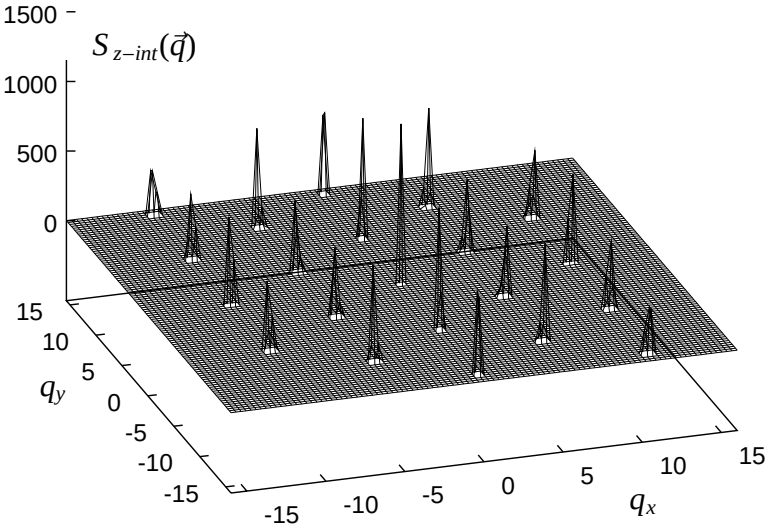
	A/n	H/N	ϑ	Ψ_6
A	1.125	-4.820	44.0°	**0.7731**
B	1.092	-4.845	41.2°	**0.9032**

Am Ordnungsparameter Ψ_6 ist deutlich zu erkennen, daß trotz ähnlich starker Neigung das Kopfgitter der modulierten Phase weit weniger stark verzerrt ist. Dadurch kommen die Köpfe einer zweidimensionalen dichtesten Packung näher, was sich auch in der Fläche pro Kette niederschlägt.

Der Parameterbereich für die Kopfgröße σ_H in dem modulierte Konfigurationen beobachtet wurden liegt etwa bei $1.14\sigma < \sigma_H < 1.27\sigma$. Bei kleineren Köpfen sind die Phasen stets konventionell geneigt, oberhalb ist der Größenunterschied zwischen Köpfen und Alkanschwanz so stark, daß keine kollektive Neigung über die ganze Monolage eintritt, sondern sich kleinere Gruppen von Ketten zu sog. Mizellen zusammenlagern (vgl. Abschnitt 5.4).
Zeigt eine Phase im Grundzustand $T \mapsto 0$ Modulation, bleibt diese bei Temperaturerhöhung in der Simulation bis etwa zum Übergang in die LE-Phase bestehen, was ggf. eine deutliche Hysterese gegenüber der unmodulierten Phase bedeutet (vgl. Abschnitt 5.3.5). Auch bezüglich des Drucks sind die modulierten Phasen bis weit in das Stabilitätsgebiet der konventionellen NNN-Phase zu beobachten. Erst wenn man den Druck auf $\Pi \gtrsim 90$ erhöht findet man eine zunehmende Verbreiterung der Streifen.

Es wurde zwar keine systematische Untersuchung der Abhängigkeit von der Kettenlänge l vorgenommen, für $l = 6$ und $l = 8$ aus Abschnitt 5.3.6 findet man bei $P = 1, T = 0.1$ aber ähnliche Modulationen, wie hier für $l = 7$ gezeigt.

Falls die beschriebenen Modulation in realen Langmuir-Monoschicht existieren würde, müßte man sie anhand des Strukturfaktors erkennen können, was ein Vergleich zwischen modulierter und nicht-modulierter NN-Phase zeigt:



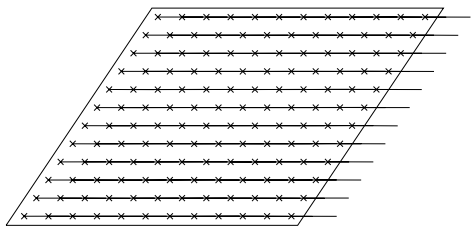
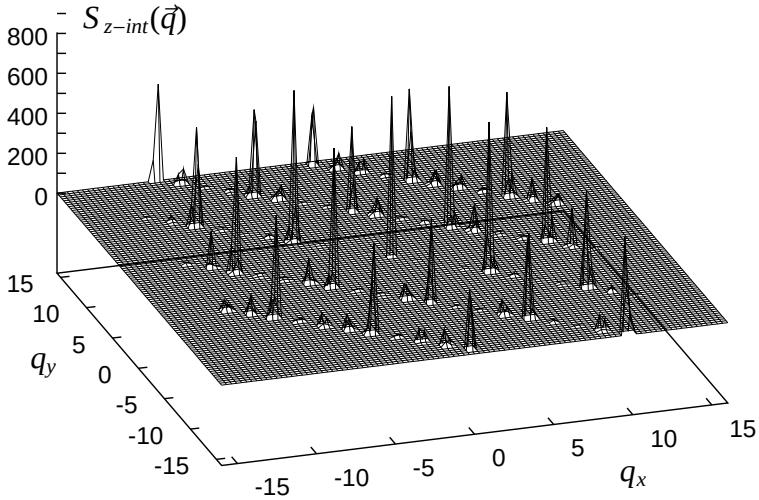


Abb. 5.34: End-zu-End-Projektion eines **unmodulierten** NN-geneigten Systems mit $\sigma_H = 1.2\sigma$, $\Pi = 25$ und $T \approx 0$

Abb. 5.35: zugehöriger Strukturfaktor $S_{z-int}(\vec{q}_{||})$ (vgl. 2.5.21)



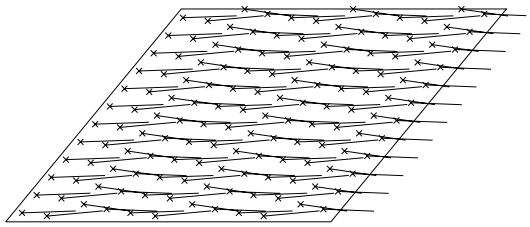


Abb. 5.36: End-zu-End-Projektion eines **modulierten** NN-geneigten Systems mit $\sigma_H = 1.2\sigma$, $\Pi = 1$ und $T \approx 0$

Abb. 5.37: zugehöriger Strukturfaktor $S_{z-int}(\vec{q}_{\|})$ (vgl. 2.5.21)

Die Strukturfaktoren enthalten jeweils die Beugungsmaxima 0.,1. und 2. Ordnung. Die modulierte Phase zeigt in ξ-Richtung eine **Überstruktur** mit einer Streifenbreite von 4 Molekülen[10], während in η-Richtung die ungestörte Periodizität vorliegt. Entsprechend findet man im Strukturfaktor in der reziproken ξ-Richtung kleine sog. *Satelliten-Peaks* im Abstand von $\frac{1}{4}$ der regulären Bragg-Peaks.

Experimentell sich die Strukturfaktoren nur als Pulver-Mittel zugänglich:

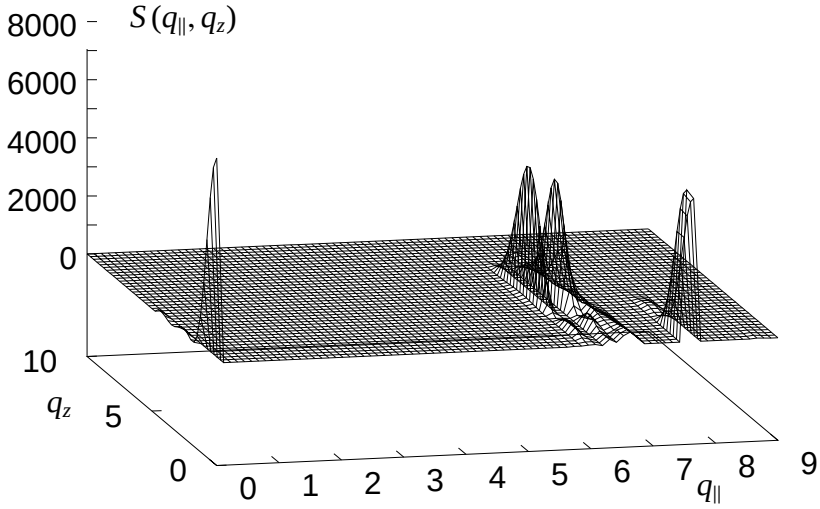


Abb. 5.38: Strukturfaktor $S(q_{\|}, q_z)$ (Pulver-Mittel) der modulierten NN-Phase; $T = 0.1, \Pi = 1$

Im Vergleich zur konventionell NN-geneigten Phase, wie sie bei $\sigma_H = 1.1\sigma$ auftritt (vgl. Abschnitt 2.5.2), erkennt man eine Aufspaltung des Peaks bei $q_z \neq 0$ in zwei getrennte Peaks. Der Abstand dieser Strukturen ist jedoch so gering, daß er nicht mehr

[10] Man beachte die periodischen Randbedingungen.

aufgelöst werden könnte, würde man die experimentell realistische Peak-Breite zugrunde legen.

Aus Sicht der Simulation stellten die modulierten Phasen in zweifacher Weise eine Schwierigkeit dar. Während es kein Problem ist, eine konventionell geneigte Phase – egal in welche der 6 NN- oder NNN-Richtungen – als Anfangskonfiguration zu konstruieren, gelang dies für modulierte Phasen nicht. Es war immer nötig, eine bereits modulierte Phase eines anderen Druck-Temperatur-Wertes hierfür heranzuziehen[11].

Als das größere Problem erwies sich, daß sich die Modulation aus einer konventionellen NN-Anfangskonfiguration nur dann einstellte, wenn der Druck ausreichend gering war ($\Pi < 10$). Bei höheren Drücken kam nur eine unvollständige Modulation mit zu großer Streifenbreite zustande.

Nebenstehende Konfigurationen wurden konventionell NN-geneigt aufgesetzt.
Man erkennt, daß sich bei $\Pi = 1$ eine suboptimale Streifenabfolge 3+5+4 gebildet hat und bei höherem Druck die Modulation sukzessive zurückgeht. Bei $\Pi = 26$ liegt hier zwar konventionelle NN-Neigung vor, die Grundzustandsiteration (vgl. Abschnittt 5.3.2) zeigt aber, daß auch bei diesem Druck die modulierte Phase eine niedrigere freie Energie hätte.

Da sowohl die Ausbildung der Modulation als auch deren Auflösung stark gehemmt sind, findet man beim Vergleich z.B. der konventionellen NNN-Phase mit der modulierten NN-Phase eine sehr große Hysterese.

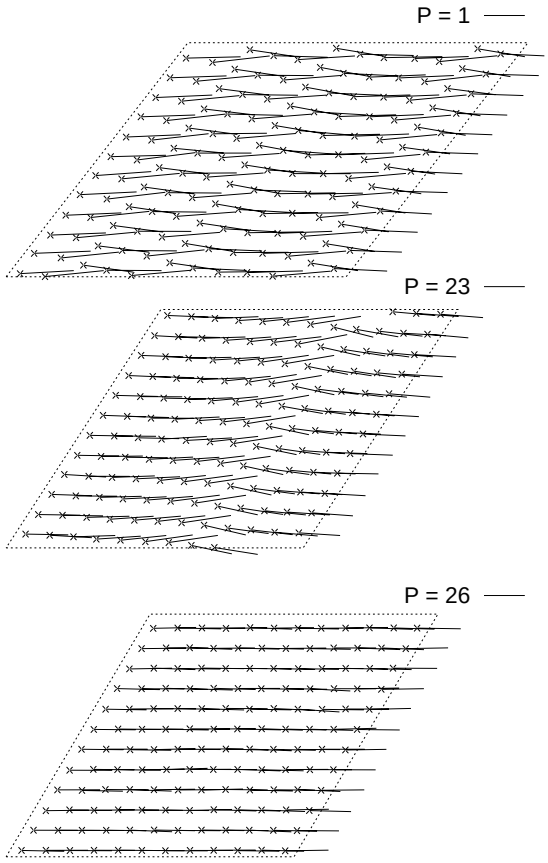


Abbildung 5.39:

NN-Konfigurationen bei $T = 0.1$, $\sigma_H = 1.2\sigma$ für die Drücke $\Pi = 1, 23, 26$ (maßstabsgerechte Abbildung)

[11] Dadurch war es auch nicht möglich, gezielt zu prüfen, ob die Simulation die 6-fache Entartung der modulierten Phase reproduzieren kann. Soweit während der Simulationsläufe Konfigurationen auftraten, die in eine andere, als die hier abgebildete Richtung moduliert waren, ergaben sich aber keine Abweichungen in den Meßwerten, so daß davon auszugehen ist, daß die Entartung erfüllt ist.

Es ist noch interessant zu untersuchen, ob sich die Überstruktur auch auf das Kopfgitter auswirkt und ob auch in der modulierten Phase der gemessene Spreizdrucktensor mit dem von außen vorgegebenen skalaren Druck übereinstimmt:

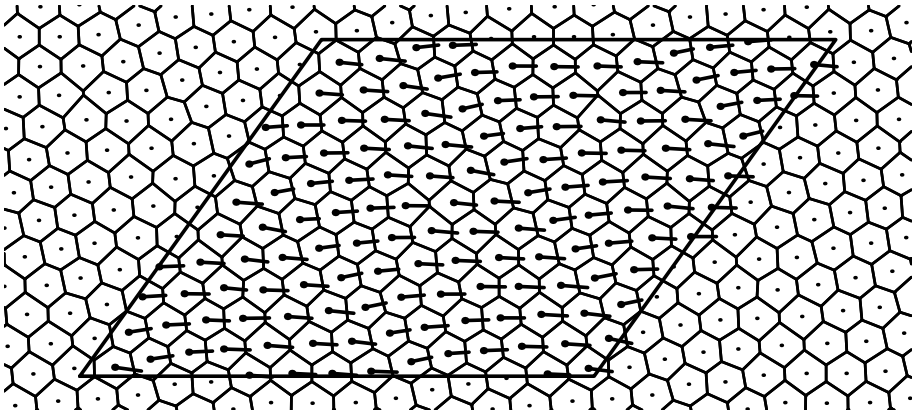


Abbildung 5.40: Voronoi-Konstruktion zur modulierten LC-NN-Phase bei $\Pi = 1,\ \ T = 0.1,\ \ \sigma_H = 1.2\sigma$

Die Voronoi-Konstruktion läßt gut die Verdrehung des Kopfgitters gegen die x-Achse der Simulationsbox erkennen. Die Form der Zellen ist ungleichmäßiger als in der unmodulierten Phase und man findet auch 5-zählig koordinierte Kopfgruppen.

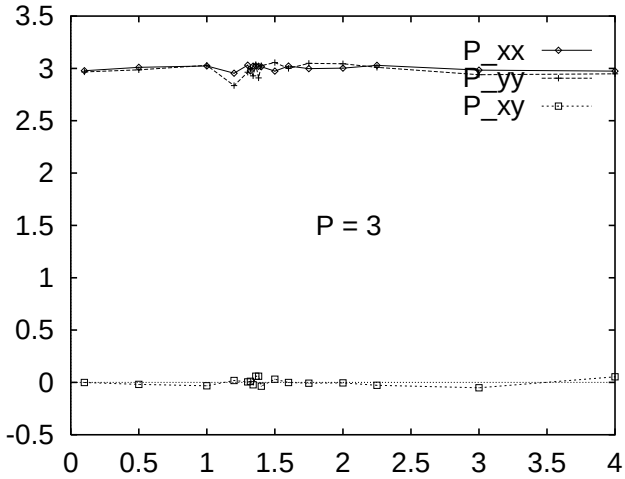


Abbildung 5.41: zu $\Pi^{ext} = 3$
Druck-Komponenten $\langle \Pi^{int}_{xx} \rangle, \langle \Pi^{int}_{yy} \rangle, \langle \Pi^{int}_{xy} \rangle$ vs. T

Bei vorgegebenem Druck $\Pi^{ext} := 3$ kann man aus der Auftragung von $\Pi^{int}_{xx}, \Pi^{int}_{yy}$ und Π^{int}_{xy} (vgl. (2.4.7)) gegen die Temperatur ablesen, daß die beiden Diagonalkomponenten miteinander und mit dem vorgegebenen Druck gut übereinstimmen und die Scherkomponente stets klein gegen die Diagonalkomponenten ist.

Die Drücke Π^{ext} und Π^{int} befinden sich somit im Gleichgewicht.

5.3.2 Simulationsdaten

Die Grundzustandsiteration für $\sigma_H := 1.2\sigma$ bei $T = 0$ erlaubt u.a. einen Vergleich zwischen modulierter und konventioneller NN-Phase:

Π	H_{NN}/N	H_{mod}/N	H_{NNN}/N
1	-4.949813	-4.989102	
15	-2.77462101		-2.65393259
25	-1.30144034	-1.46118598	-1.29812054
30	-0.61671449		-0.62953127
50		1.98189217	1.99635532
60		3.31803194	3.28473875

Interpolation zwischen $25 \leq \Pi \leq 30$ bzw. zwischen $50 \leq \Pi \leq 60$ ergibt für den (hypothetischen) Übergang LC-NN $\leftrightarrow$ LC-NNN einen Druck $\Pi \approx 26.0$ und für den (realen) Übergang LC-mod $\leftrightarrow$ LC-NNN den Druck $\Pi \approx 53.0$.
Im Bereich $1 \leq \Pi \leq 25$, in dem die konventionelle NN-Phase gegenüber NNN stabil wäre, zeigt die modulierte Phase eine noch niedrigere freie Energie. Bei $T = 0$ ist für $\sigma_H := 1.2\sigma$ die konventionelle NN-Phase also bei jedem Druck instabil bzw. metastabil.

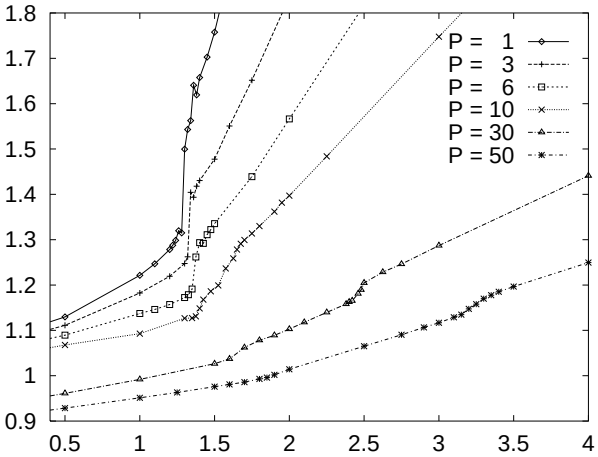


Abbildung 5.42: Fläche A/n vs. T für die Drücke $\Pi = 1, 3, 6, 10, 30, 50$

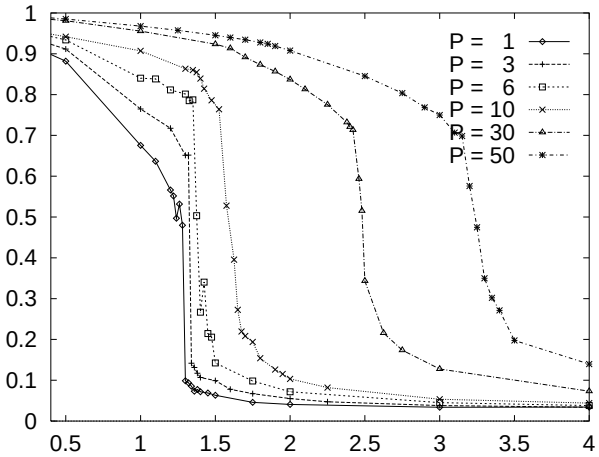


Abbildung 5.43: Ψ_6 vs. T für die Drücke $\Pi = 1, 3, 6, 10, 30, 50$

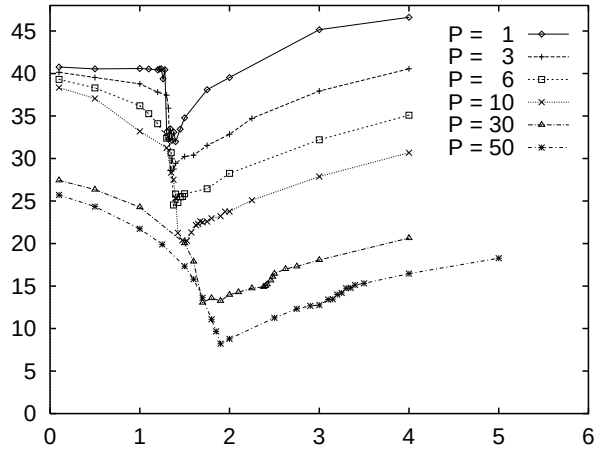


Abbildung 5.44: Neigungswinkel ϑ [°] vs. T für die Drücke $\Pi = 1, 3, 6, 10, 30, 50$

Der Verlauf der Fläche A/n und des Ordnungsparameters Ψ_6 gegenüber der Temperatur für verschiedene Isobaren zeigt ein ähnliches Verhalten wie im System kleiner Köpfe. Bei niedrigen Drücken erkennt man *einen* Übergang in eine ungeordnete Phase, dieser ist mit einem deutlichen Flächensprung verbunden.

Bei höheren Drücken findet man zuvor wieder einen weiteren Übergang in eine ebenfalls hexagonal geordnete Phase. Dieser Übergang bewirkt keinen Flächensprung, aber einen Knick in der $A(T)$-Kurve. Außerdem ist die Flächenänderung beim Übergang in die ungeordnete Phase geringer und breiter ausgedehnt.
Zusammen mit dem mittleren Neigungswinkel $\langle\vartheta\rangle$ ergibt sich, ähnlich wie im System kleiner Köpfe, daß bei niedrigem Druck eine kollektiv geneigte LC-Phase in die ungeordnete LE-Phase übergeht und bei höherem Druck dazwischen eine ungeneigte LC-Phase existiert.

Die Übergangstemperaturen verschieben sich mit steigendem Druck nach oben und sind mit denen zu $\sigma_H = 1.1\sigma$ vergleichbar.

Die kollektive Neigung ist mit $\langle\vartheta\rangle \lesssim 40°$ deutlich größer und liegt auch über den typischen, experimentell gefundenen Werten [64]. Dies gilt nicht nur in der geneigten, sondern auch in der „ungeneigten“ LC-U-Phase, da die Gitterabstände vom Durchmesser der größeren Kopfgruppen bestimmt werden und die Schwanzmonomere somit mehr Raum haben, sich darüber anzuordnen.

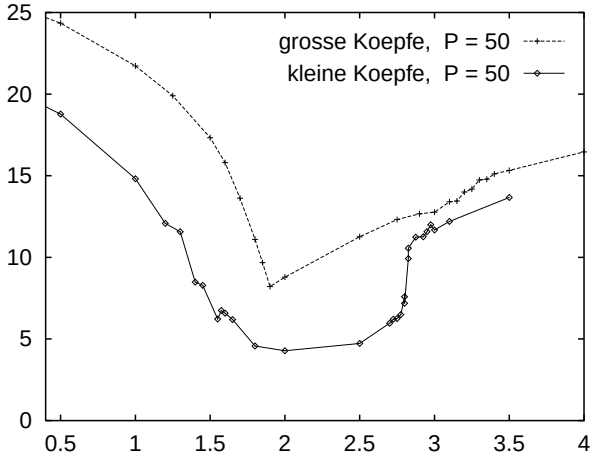


Abbildung 5.45: Neigungswinkel ϑ [°] vs. T für die Drücke $\Pi = 1, 3, 6, 10, 30, 50$

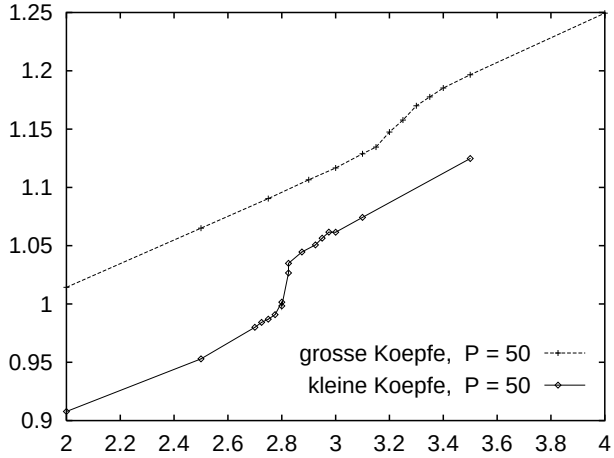


Abbildung 5.46: Fläche A/n vs. T bei $\Pi = 50$ für $\sigma_H = 1.1\sigma$ und $\sigma_H = 1.2\sigma$

Ein unmittelbarer Vergleich der Isobaren zu $\Pi = 50$ bestätigt dies. Während bei den kleineren Köpfen die Neigung in der LC-U-Phase gering bleibt und erst mit dem Aufschmelzen in die LE-Phase auch größere Neigungswinkel möglich werden, wächst im System großer Köpfe der mittlere Neigungswinkel gleichmäßig an, so daß am Übergang in die LE-Phase kein deutlicher Sprung eintritt.
Die beiden Flächenkurven belegen zunächst, daß das System der großen Köpfe und der stärkeren Neigung eine größere Fläche pro Monomer beansprucht. Im weiteren fällt auf, daß – verglichen mit den kleinen Köpfen – am Übergang in die LE-Phase kaum noch von einem „Flächensprung“ gesprochen werden kann.

Der Verlust der kollektiven Neigung am Übergang von LC-NNN bzw. LC-NN in die ungeneigte LC-U-Phase wird durch den Parameter R_{xy} (2.4.12) beschrieben.

Die Übergangstemperaturen zwischen den LC-Phasen wurden, wie bei den kleinen Köpfen und in Übereinstimmung mit den anderen Observablen, anhand des gewichteten Histogramms des Neigungswinkels (vgl. Abschnitt 5.2.2) lokalisiert.

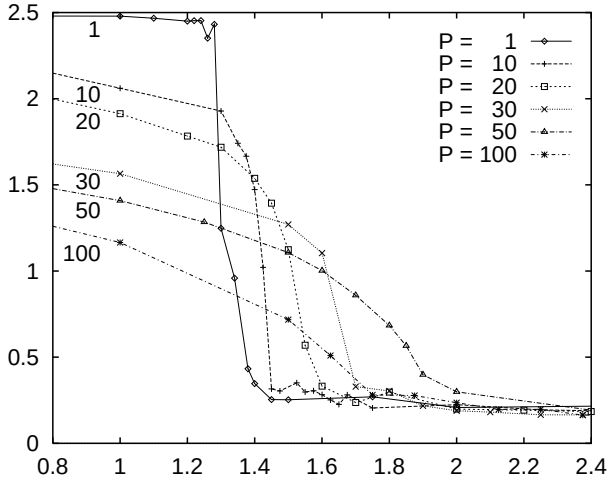


Abbildung 5.47: Neigungsmaß R_{xy} vs. T für die Drücke $\Pi = 1, 10, 20, 30, 50, 100$

Eine Grundzustandsuntersuchung der konventionellen NN- und NNN-Phase ergab, daß deren Grenzlinie etwa bei $\Pi \approx 30$ verläuft. Bei tiefen Temperaturen liegt diese Grenze innerhalb des Stabilitätsgebiets der modulierten Phase (vgl. Abschnitt 5.3.5), so daß sie erst bei $T \gtrsim 0.5$ relevant wird.
Dort tritt jedoch wie im System kleiner Köpfe eine Entkopplung der Neigungsrichtung von den Bondrichtungen des Kopfgitters auf, was zu einem weit ausgedehnten Übergangsbereich führt.

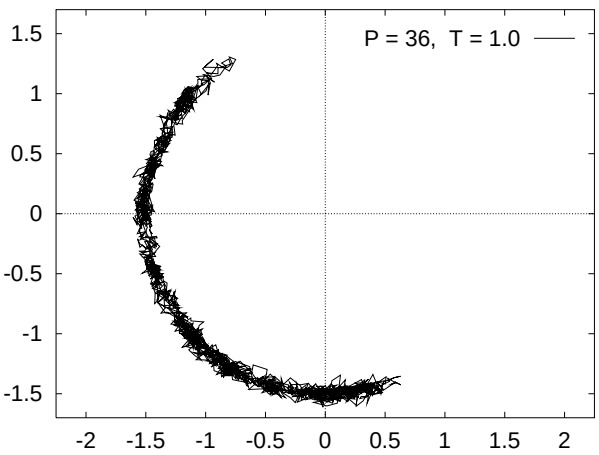


Abbildung 5.48: Trajektorie D_x-D_y für $\Pi = 36$, $T = 1$ über 1.200.000 MCS

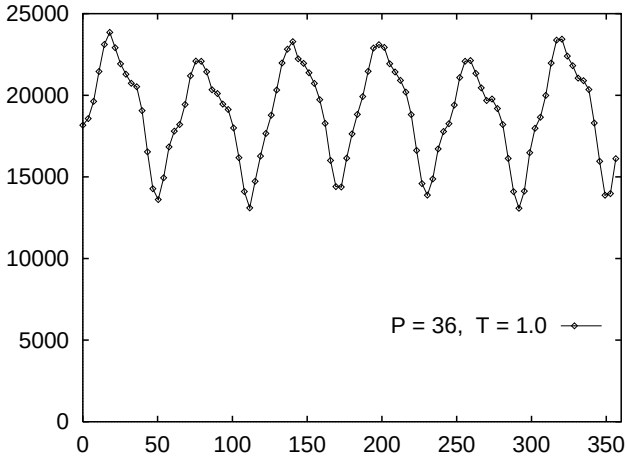


Abbildung 5.49: Verteilung der Bondrichtungen bezogen auf die Neigungsrichtung für $\Pi = 36$, $T = 1$

Am Beispiel $\Pi = 36, T = 1$, das über $1.2 \cdot 10^6$ MCS aufgenommen wurde, sieht man, wie sich der Moleküldirektor auf einem Kreis (fester Neigungswinkel) nahezu frei bewegen kann. Auch die Verteilung der Winkel, welche die Bonds des Kopfgitters mit der Neigungsrichtung bilden, zeigt nur noch geringe Modulation.

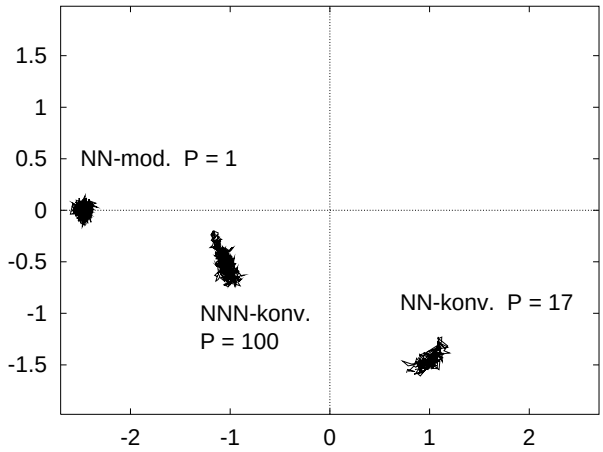


Abbildung 5.50: Trajektorien D_x-D_y bei $T = 1$ für $\Pi = 1$ (NN-mod.), $\Pi = 17$ (NN-konv.) und $\Pi = 100$ (NNN-konv.)

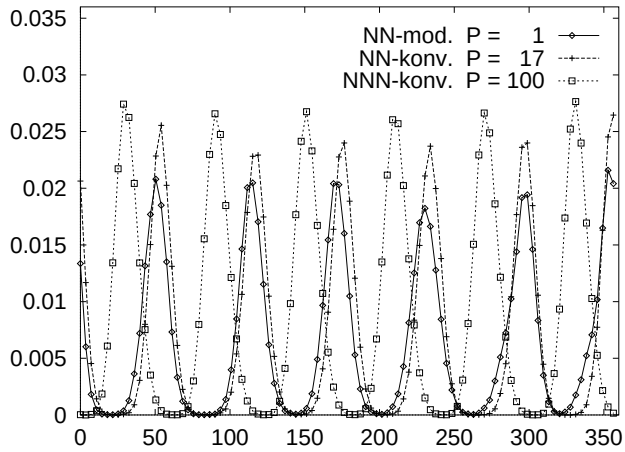


Abbildung 5.51: Verteilung der Bondrichtungen bezogen auf die Neigungsrichtung bei $T = 1$ für $\Pi = 1, 17, 100$

Im Vergleich dazu erhält man bei Drücken außerhalb des entkoppelten Bereichs eindeutige Vorzugsneigungen in NNN- bzw. NN-Richtung, letztere sowohl konventionell als auch moduliert.

Während der Übergang zwischen den beiden konventionell geneigten LC-Phasen dadurch Probleme bereitete, daß es einen ausgedehnten Bereich ohne Vorzugsneigungsrichtung gibt, trat im Vergleich der konventionellen und der modulierten Phase eine sehr große Hysterese bezüglich des Druckes ein. Z.B konnten für $T := 0.5$ im Druckbereich von $\Pi =$ 15...90 beide Phasen beobachtet werden. Erst unterhalb dieses Bereichs setzte spontane Modulation der konventionellen Phase ein, bzw. darüber verschwand die Modulation durch zunehmend größere Streifenbreiten.
Durch die Hysterese und den kontinuierlichen Übergang zwischen den konventionellen Neigungsrichtungen war es andererseits möglich, die Grenze zwischen modulierter und unmodulierter Phase durch thermodynamische Integration (vgl. Abschnitt 4.5) über einen weiten Bereich zu verfolgen.
Eine Überprüfung der so gefundenen Stützstellen auf der Phasengrenze mit dem Gesetz von Clausius-Clapeyron (4.5.52) ergibt gute Übereinstimmung und bestätigt den nahezu geraden Verlauf der Grenzlinie im $\Pi(T)$-Phasendiagramm in Abschnitt 5.3.5:

Nr.	T	Π	$\Delta\Pi/\Delta T$	$\delta\Pi/\delta T$
0	0.000	53.03		
1	0.064	50.00	-47.2	-45.7
2	0.200	43.71	-46.3	-42.6
3	0.458	32.00	-45.4	-44.3
4	0.800	16.78	-44.5	-41.8

$$\frac{\Delta\Pi}{\Delta T} = \frac{\Pi_i - \Pi_{i-1}}{T_i - T_{i-1}}$$

$$\frac{\delta\Pi}{\delta T} = \frac{H_{mod} - H_{konv}}{T \cdot (A_{mod} - A_{konv})}$$

5.3.3 Hysterese

Der Charakter der anderen Phasenübergänge geht aus den Simulationsdaten oder der Theorie (vgl. Abschnitt 5.2.3) hervor. Für den Übergang der ungeneigten LC-U-Phase in die ungeordnete LE-Phase gelingt eine solch klare Einordnung hingegen nicht.

Der Übergang zwischen einer flüssigen und einer festen Phase ist zwar notwendig von 1. Ordnung [41], jedoch liegt ein solcher am Übergang LC/LE nicht zwangsläufig vor. Die flüssig-kondensierten LC-Phasen sind verglichen mit der LE/G-Phase in höherem Maß geordnet; sie sind aber keine eigentlichen kristallinen, 3-dimensionalen Festkörper. Bei rein 2-dimensionalen Systemen kann zwischen der festen und der flüssigen Phase eine *hexatische* Phasen existieren [40][52][71] die auch in entsprechenden Simulationen [15] beobachtet wird.
In dieser zerfällt die Positionskorrelation kurzreichweitig, während die Bondrichtungen, unter denen die Nachbarteilchen gesehen werden, langreichweitig bestehen. Anstatt des unstetigen fest↔flüssig-Übergangs treten nun zwei kontinuierliche Übergänge – fest↔hexatisch und hexatisch↔flüssig – auf.
Langmuir-Monoschichten verhalten sich, trotz nicht-verschwindender Schichtdicke, wie 2-dimensionale Systeme. Um von kollektiver Neigung zu nächsten oder übernächsten Nachbarn sprechen zu können, müssen die Bondrichtungen des Kopfgitters über ausreichende Entfernungen einheitlich sein. Man geht daher davon aus, daß die LC-Phasen[12] hexatisch sind, was auch experimentell [29] bestätigt wird.
Da also ein LC-U/LE-Übergang 2. Ordnung vorliegen kann und eine Finite-Size-Untersuchung aus technischen Gründen nicht möglich war (vgl. Abschnitt 5.3.7), könnte das Auftreten von Hysterese den Ausschlag für einen Übergang 1. Ordnung geben.

Zur Aufnahme der Hysterese wurde das System sukzessive aufgeheizt bzw. abgekühlt und auf jeder Temperaturstufe eine vorgegebene Zahl von Simulationsschritten durchgeführt.

Beim Abkühlen der ungeordneten LE-Phase tritt dabei das Problem auf, daß sich zur Bildung der LC-Phase die Köpfe auf eine Art hexagonal anordnen müssen, die mit der Simulationsbox kompatibel ist, bei $n = 144$ z.B. 12 Reihen à 12 Ketten. Zudem kommen beim Übergang in eine geneigte LC-Phase Konfigurationen vor, die keine einheitliche Neigungsrichtung haben, sondern z.B. zwei verschieden geneigte Domänen.
Beide Arten von Strukturfehlern sind metastabil und werden von der Simulation kaum mehr ausgeheilt. Derartige Probleme stellten sich beim Aufheizen nicht, da hier mit fehlerlos geordneten Konfigurationen begonnen wurde.

[12] Mit Ausnahme der kristallinen CS-Phase (vgl. Abb. 1.1)

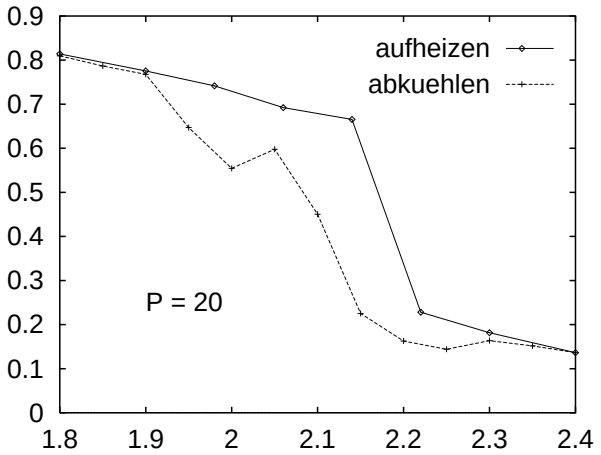


Abbildung 5.52

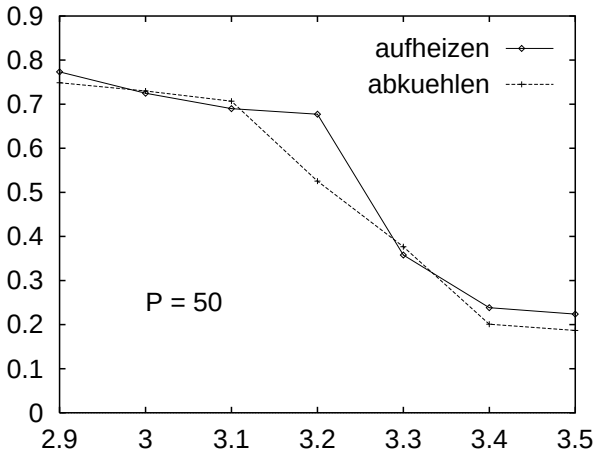


Abbildung 5.53

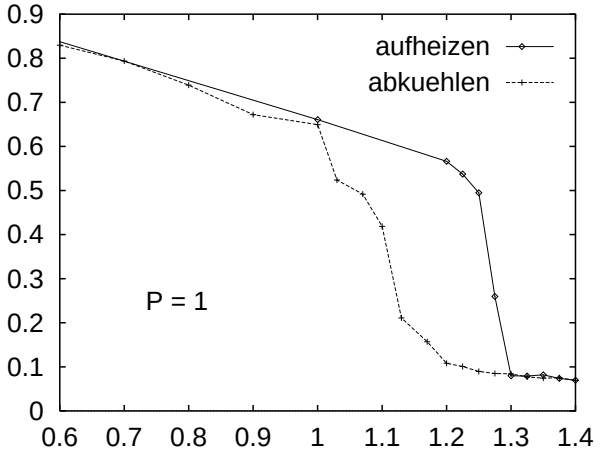


Abbildung 5.54

Abb. 5.52-5.54: Ordnungsparameter Ψ_6 vs. T für die Drücke $\Pi = 20, 50, 1$

Für das schrittweise Aufheizen und Abkühlen wurden folgende Prozess-Parameter gewählt:

Π	MCS	ΔT	$T^{equil.}_{LC\leftrightarrow LE}$
1	100.000	0.025	1.29
20	50.000	0.080	2.04
50	100.000	0.100	3.22

Zur Kontrolle wurde neben den Drücken $\Pi = 20; 50$, die zum Übergang LC-U↔LE gehören, auch die Isobare $\Pi = 1$ untersucht, bei der mit LC-NN↔LE sicher ein Übergang 1. Ordnung erfolgt.
Bei $\Pi = 1$ zeigt der Ordnungsparameter Ψ_6, der das Aufschmelzen des hexagonalen Kopfgitters beschreibt, erwartungsgemäß eine deutliche Hystereseschleife mit einer Breite von $\Delta T \approx 0.2$.

Die Verläufe von $\Psi_6(T)$ zu $\Pi = 20; 50$ lassen keine eindeutige Interpretation zu.
Für $\Pi = 50$ unterscheiden sich beide Kurven kaum. Dies kann damit zusammenhängen, daß sich mit zunehmenden Druck die LC-U- und LE-Phase immer „ähnlicher“ werden, was sowohl am zurückgehenden Flächensprung als auch an den Konfigurationsschnappschüssen (vgl. Abschnitt 5.3.5) zu sehen ist. Dadurch verbleiben beide Prozesse stets nahe dem Gleichgewicht.
Die Kurven zur Isobaren $\Pi = 20$ weichen stärker voneinander ab als bei $\Pi = 50$. Bei $\Pi = 1$ kann man aber deutlich ein Verbleiben in der Ausgangsphase auch über die Übergangstemperatur hinaus erkennen, was bei $\Pi = 20$ nur ansatzweise eintritt.

Die Untersuchung des Hystereseverhaltens läßt also auch keine eindeutige Charakterisierung des Übergangs LC-U↔LE zu.
Man muß nämlich beachten, daß auch in Monte Carlo-Simulationen kontinuierlicher Phasenübergänge eine scheinbare Hysterese immer dann beobachtet wird, wenn die Zahl der MC-Schritte für die simulierten Zustände zumindest teilweise zu gering war, um voll ins Gleichgewicht zu relaxieren.
Da wegen des *slowing down* bei kontinuierlichen Phasenübergängen unter Umständen sehr große Relaxationszeiten auftreten können, die man vorher nicht abschätzen kann, ist eine Unterscheidung der Ordnung von Phasenübergängen aufgrund solcher Hysterese-Untersuchungen oft nicht in eindeutiger Weise möglich.
Alternativ erfolgten sehr lange Simulationsläufe ($2 \cdot 10^6$ MCS) am vermuteten Übergangspunkt.
Auch hierbei konnte das erwartete, mehrfache Hin-und-Her-Wechseln des Ordnungsparameters zwischen zwei verschiedenen, den beiden koexistierenden Phasen zuzuordnenden Werten nicht zweifelsfrei beobachtet werden.

5.3.4 Auftreten der Gasphase

In den experimentellen Untersuchungen von Langmuir-Monolagen ist bei hohen Temperaturen das Auftreten einer Gasphase evident [39]. Diese Phase, die so stark verdünnt ist, daß die Ketten untereinander kaum noch wechselwirken, ist von der ebenfalls ungeordneten LE-Phase durch einen Übergang 1. Ordnung getrennt. Es stellt sich die Frage, ob diese Phase auch in der vorliegenden Simulation beobachtet werden kann.

Mit steigender Verdünnung kann man das Erreichen des 2-dimensionalen idealen Gas-Limits anhand zweier quantitativer Größen beschreiben. Zum einen sollte das

$$\text{ideale Gas-Gesetz} \quad p \cdot V = N \cdot k_B \cdot T \quad \Rightarrow \quad \frac{\Pi \cdot A}{n \cdot T} \stackrel{!}{=} 1 \tag{5.3.6}$$

erfüllt sein[13], zum anderen sollte für (starre) Stäbe, die ohne Wechselwirkung miteinander den ganzen oberen Halbraum zur Verfügung haben, sich ein

$$\text{mittlerer Neigungswinkel} \quad \langle\vartheta\rangle \stackrel{!}{=} \int_0^{\pi/2} \vartheta \sin\vartheta \, \mathrm{d}\vartheta = 1 \tag{5.3.7}$$

ergeben.

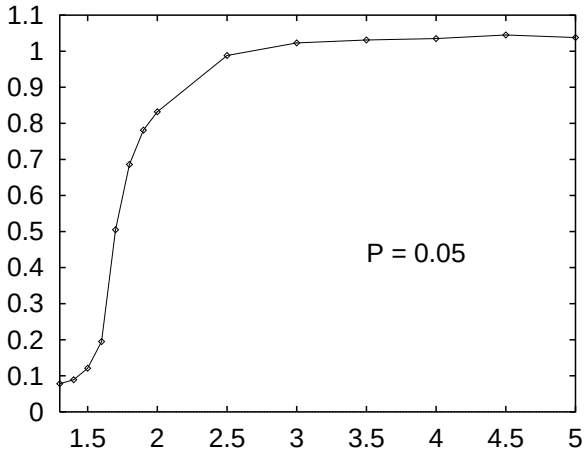


Abbildung 5.55: Gaskonstante $\Pi\langle A\rangle/nT$ vs. Temperatur T bei $\Pi = 0.05$, $\sigma_H = 1.2\sigma$

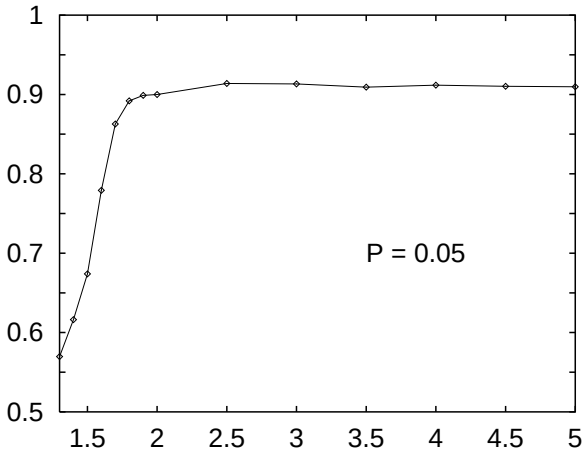


Abbildung 5.56: Neigungswinkel ϑ [rad] vs. Temperatur T bei $\Pi = 0.05$, $\sigma_H = 1.2\sigma$

Beim Aufheizen entlang der Isobaren $\Pi = 0.05$ stellt man fest, daß sowohl die „Gaskonstante“ als auch der Neigungswinkel bei Temperaturen ab $T \gtrsim 2$ Plateaus erreichen, deren Werte[14] der Gas-Phase gut entsprechen. Die Auftragung beginnt jeweils knapp oberhalb des Übergangs von der LC-NN in die LE-Phase.

[13] Zur genaueren Beschreibung der ungeordneten Phase(n) kann man z.B. eine Van-der-Waals-Zustandsgleichung ansetzen. *Israelachvili* schlägt eine *surface-micelle*-Isotherme vor [32], um die Bildung von Clustern zu berücksichtigen.

[14] $\langle\vartheta\rangle$ erreicht nicht den Wert 1, da es sich nicht um starre Stäbe handelt und sich die Ketten der Grenzfläche nur von oben nähern können.

Technisch stellen diese Simulationen insofern ein Problem dar, als in der LE- bzw. G-Phase die Kopfgruppen nicht mehr hexagonal geordnet sind, sondern eine 2-dimensionale Flüssigkeit bilden. In dieser ist nur noch die *Fläche* der Simulationsbox relevant, da diese in den Hamiltonian eingeht, nicht mehr aber deren *Form*.

Die Entartung bezüglich L_x, L_y und α führt zu Schwierigkeiten, sobald im Verlauf der Simulation Konfigurationen eintreten, die – bei vorgegebener Fläche – extrem lang oder extrem schmal sind.

Für die Auswertung der periodischen Randbedingungen nach der *minimum image convention* ist es nötig, daß die Systemausdehnung mindestens so groß ist, daß jedes Teilchen i mit höchstens einem periodischen Abbild des Teilchens j wechselwirkt. Im verwendeten Algorithmus muß hierfür gefordert werden, daß die Verletkugel (vgl. Abschnitt 3.3) noch vollständig in die Simulationsbox hineinpaßt [3].

Versucht man die ungeordneten LE- oder G-Phasen mit der scherbaren Box-Geometrie zu simulieren, die sich für die LC-Phasen als notwendig erwiesen hat, ist es eine Frage der MC-Schritte, bis die beschriebenen pathologischen Box-Formen auftreten. Andererseits ist es in diesen Phasen unnötig, der Box die Möglichkeit zu geben, neben der Größe auch ihre Form zu ändern.

Die Untersuchung des Übergangs LE $\leftrightarrow$ G erfolgte daher mit *festem Seitenverhältnis* der Box, i.allg. mit quadratischem Grundriß. Vergleiche mit Daten zu fluktuierendem Seitenverhältnis, soweit diese möglich waren, ergaben völlige Übereinstimmung.

Neben der Isobaren $\Pi = 0.05$ wurde die Gas-Phase auch anhand der Isothermen $T = 1.45$ bei niedrigen Drücken untersucht.

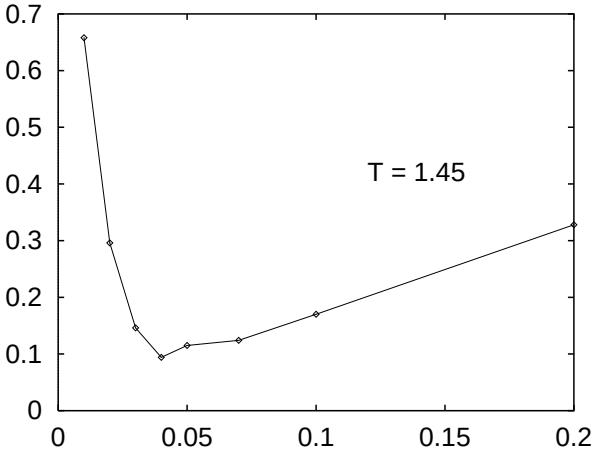


Abbildung 5.57: Gaskonstante $\Pi\langle A\rangle/nT$ vs. Druck Π bei $T = 1.45$, $\sigma_H = 1.2\sigma$

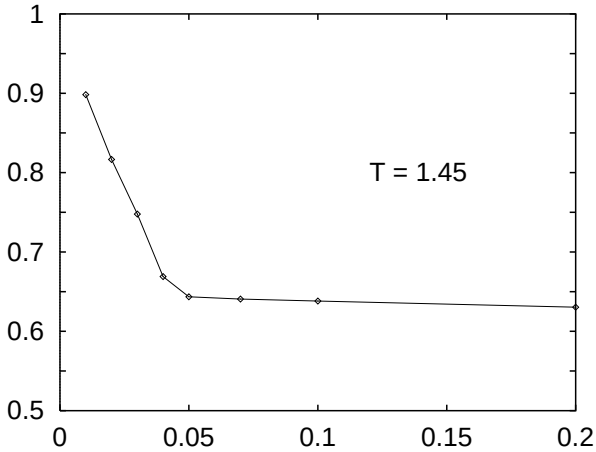


Abbildung 5.58: Neigungswinkel ϑ [rad] vs. Druck Π bei $T = 1.45$, $\sigma_H = 1.2\sigma$

Das System wurde, von höheren Drücken kommend, schrittweise entspannt. Unterhalb von $\Pi \lesssim 0.05$ tritt eine Änderung des Verhaltens ein. Wenngleich die „Gas-Grenzwerte“ nicht vollständig erreicht werden, dürfte dies bei noch tieferen Drücken schließlich der Fall sein.

Im folgenden sind jeweils die Isobaren-Daten den Isothermen-Daten gegenübergestellt:

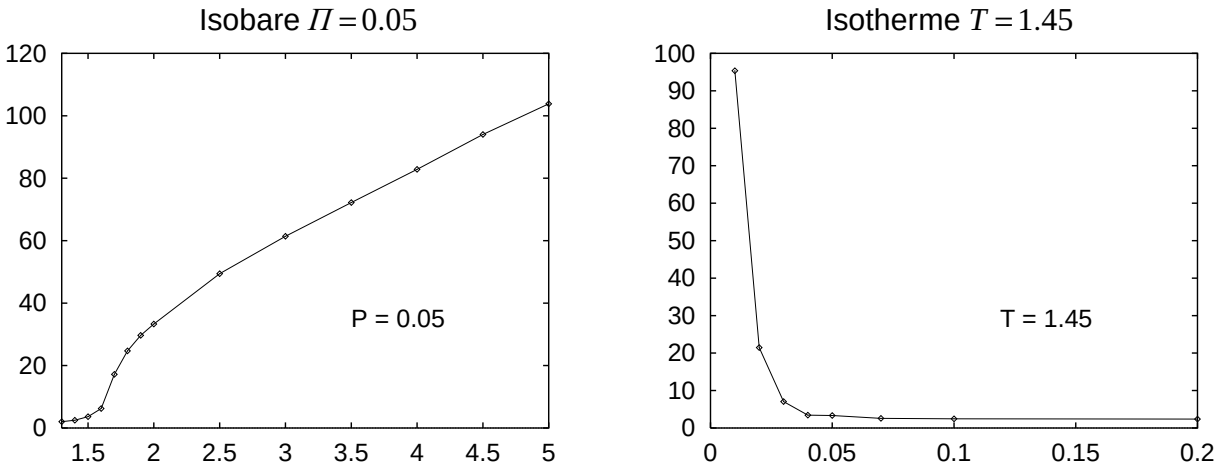


Abbildung 5.59: Fläche A/n vs. Temperatur T bzw. Druck Π

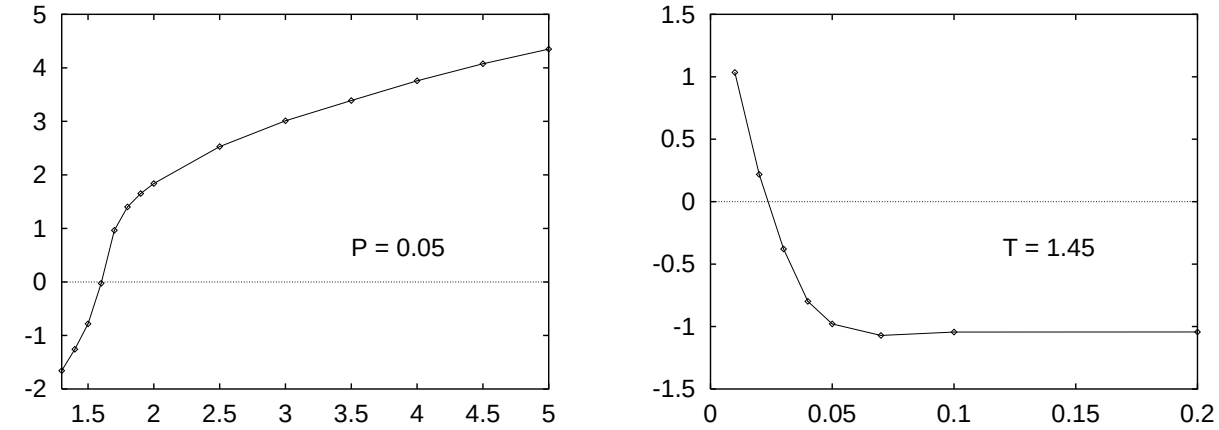


Abbildung 5.60: spez. Enthalpie H/N vs. Temperatur T bzw. Druck Π

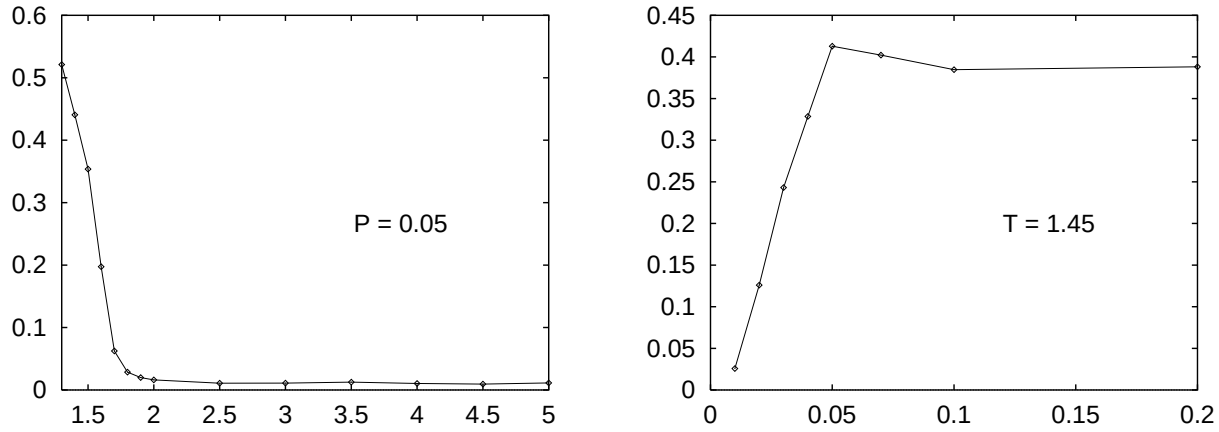


Abbildung 5.61: Neigungskorrelation K_{NN} vs. Temperatur T bzw. Druck Π

Zusammenfassend ergeben diese Plots das Bild, daß bei Druck $\Pi = 0.05$ im Temperaturbereich $1.5 < T < 1.7$ das System zwar eine Umwandlung erfährt, dieser Vorgang

aber relativ breit ausgeschmiert ist, so daß eine konkrete Übergangstemperatur bzw. ein Übergangsdruck nicht lokalisiert werden kann.

Es bietet sich an, diese Umwandlung hinsichtlich *Hysterese* und *Finite-Size*-Abhängigkeit zu untersuchen.

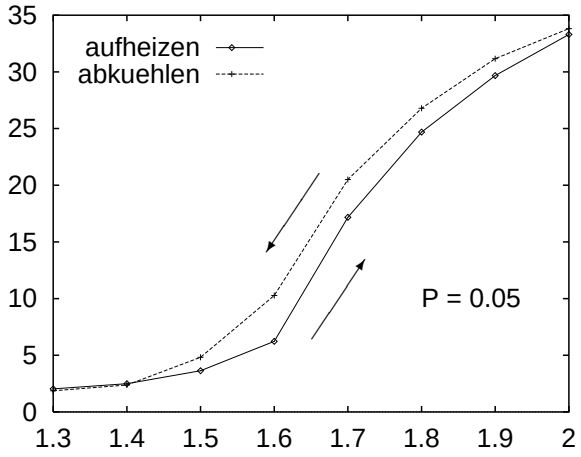


Abbildung 5.62: Fläche A/n vs. T bei $\Pi = 0.05$ für schrittweises Aufheizen/Abkühlen

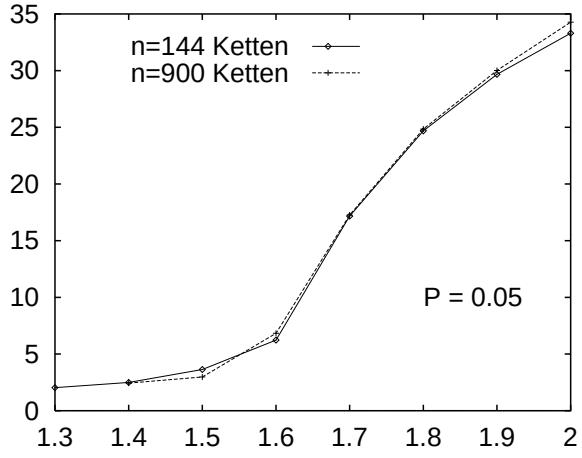


Abbildung 5.63: Fläche A/n vs. T bei $\Pi = 0.05$ für Systemgrößen $n = 144$ und $n = 900$

Der Vergleich mit einem System von n = 900 Ketten deutet auf keine ausgeprägte Abhängigkeit von der Teilchenzahl hin.
Auch die „Hysterese-Schleife“ entspricht nicht der Vorstellung des Verbleibens in der einen Phase mit verzögertem Sprung in die andere. Vielmehr scheint das System sich stetig auf Gleichgewichtszustände[15] hin zu bewegen, die im Zwischenraum zwischen der Aufheiz- und Abkühlkurve liegen.

Diese Vermutung bestätigt auch die Auftragung der Fläche bzw. der Dichte n/A gegen die MC-Schritte:

Selbst über 500.000 MCS ist die Fläche offensichtlich noch nicht äquilibriert.

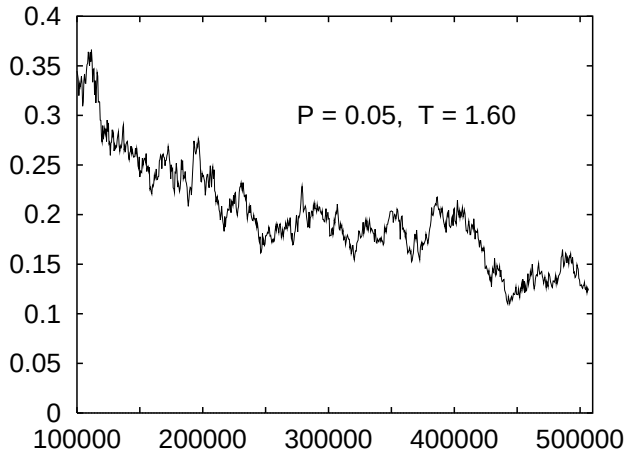


Abbildung 5.64: **Dichteverlauf** n/A vs. MCS für Druck $\Pi = 0.05$ und Temperatur $T = 1.6$

[15]Da es sich hier um den Übergang zwischen zwei ungeordneten Phasen handelt, treten beim Abkühlen keine metastabilen Gitterfehler auf, wie sie beim Übergang in die LC-Phase beobachtet werden (vgl. Abschnitt 5.3.3).

Die zusätzliche Simulation der Isothermen $T = 2$ im gleichen Druckgebiet ergab ein ähnliches Verhalten wie die Isotherme $T = 1.45$, nur daß die „Gas-Grenzwerte“ bereits bei höheren Drücken erreicht werden und der Übergang deutlich breiter ausschmiert.

Nebenstehende Projektionen der Ketten in die Grenzfläche sind maßstabsgerecht und zeigen deutlich die starke Expansion des Systems im Temperaturbereich $T = 1.3...1.9$.
Während die Konfiguration bei $T = 1.9$ einem idealen Gas schon recht nahe kommt, erkennt man aus den niedrigeren Temperaturen, daß diese Phase dadurch entsteht, daß die geschlossene Monolage in einzelne Cluster aufbricht.
Diese Cluster, die in sich eine annähernd dichte Packung der Ketten aufweisen, driften beim Aufheizen auseinander und verringern ihre Aggregationszahl.

Das Auftreten sog. *surface micelles* dient auch zur Erklärung des experimentell beobachteten, nicht gänzlich horizontalen Verlaufs der gemessenen $\Pi(A)$-Kurven am Übergang LC-LE [19][32].
Wie in den abgebildeten Konfigurationen zu sehen ist, sind die Micelles, die in der Simualtion auftreten, thermisch instabil und lösen sich bei endlicher Temperatur (hier $T \gtrsim 2$) gänzlich auf, so daß ein System vereinzelter Ketten vorliegt.

Für dieses Regime scheint der verwendete particle-displacement-Algorithmus nicht sehr effizient zu sein. Da hier Umlagerungen ganzer Ketten gute Akzeptanzchancen hätten, könnte die CCBMC-Methode vorteilhaft sein – dies wurde allerdings nicht erprobt.

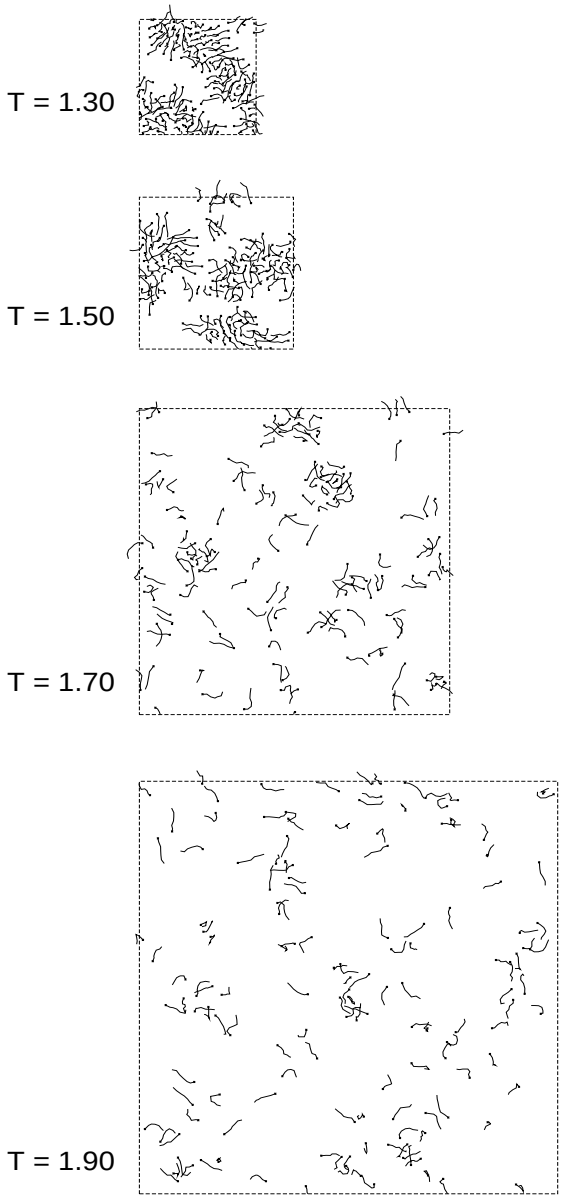


Abbildung 5.65: Konfigurations-Sequenz bei $\Pi = 0.05$ für $T = 1.30, 1.50, 1.70, 1.90$

Zusammenfassend kann man die vorliegenden Daten so interpretieren, daß man sich bei $\Pi = 0.05, T = 1.5$ zwar vermutlich in der Nähe eines Phasenübergangs LE $\leftrightarrow$ G befindet, im untersuchten Bereich beide Phasen aber identisch sind.
Von der Grenzlinie des LE/G-Übergangs weiß man, daß sie für ein 2-dimensionales Lennard-Jones-System in einem kritischen Punkt endet [1][70]. Wenn nun hier kein

direkter Phasenübergang LE/G beobachtet wird, kann dies daran liegen, daß man stets jenseits des kritischen Punktes simuliert hat, d.h. in dem Gebiet, in dem beide Phasen ununterscheidbar sind.
Eine andere Erklärung ist, daß die repulsive Wechselwirkung zwischen den Köpfen (bzw. den Clustern) diesen Übergang gegenüber dem Lennard-Jones-System so verändert, daß keine zwei verschiedenen fluiden Phasen mehr existieren.

5.3.5 Phasendiagramm

In den folgenden, charakteristischen Konfigurations-Schnapschüsse wurden die effektiven Monomere jeweils raumfüllend als Kugeln mit Durchmesser σ bzw. σ_H dargestellt.

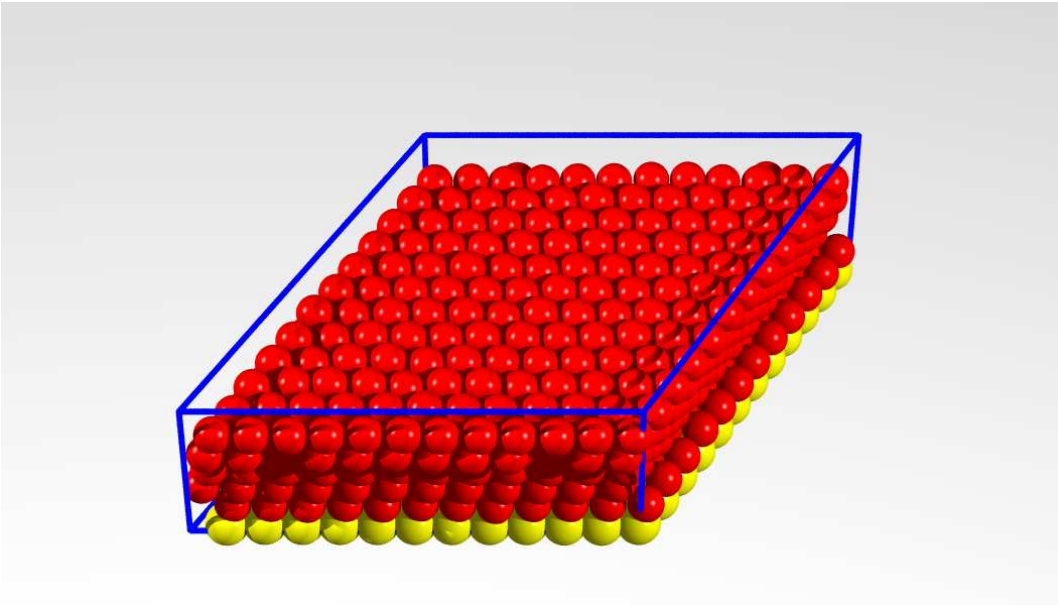


Abbildung 5.66: Snapshot der **konventionellen LC-NNN-Phase**: $\Pi = 100, \quad T = 0.1$

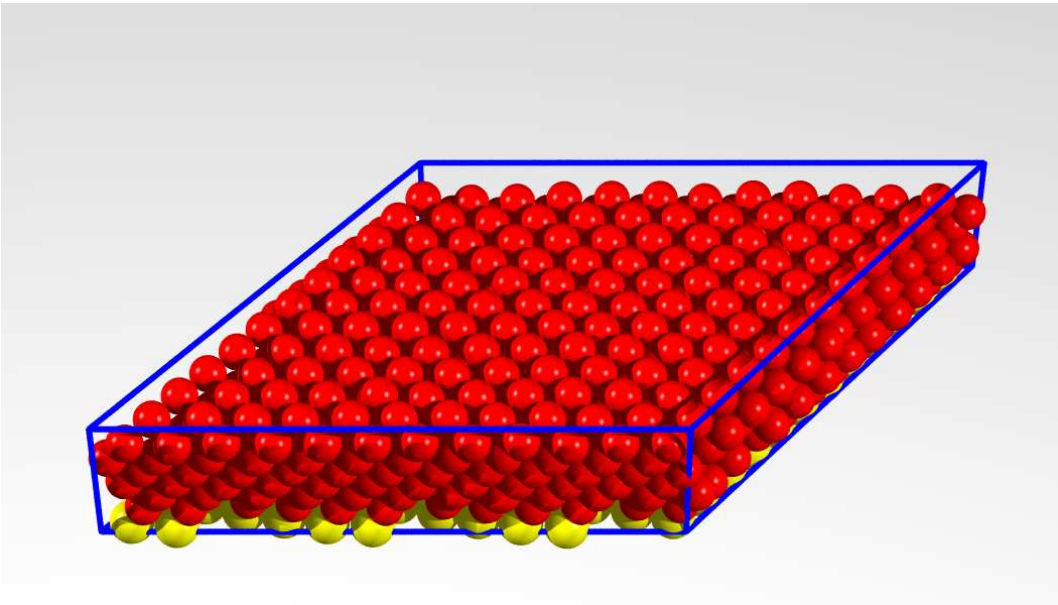


Abbildung 5.67: Snapshot der **modulierten LC-NN-Phase**: $\Pi = 1, \quad T = 0.1$
Die Verdrehung des Kopfgitters gegen die Simulationsbox ist an der Unterkante des Bildes deutlich zu erkennen, während die oberste Monomerlage hexagonale Ordnung parallel zu den Boxkanten besitzt.

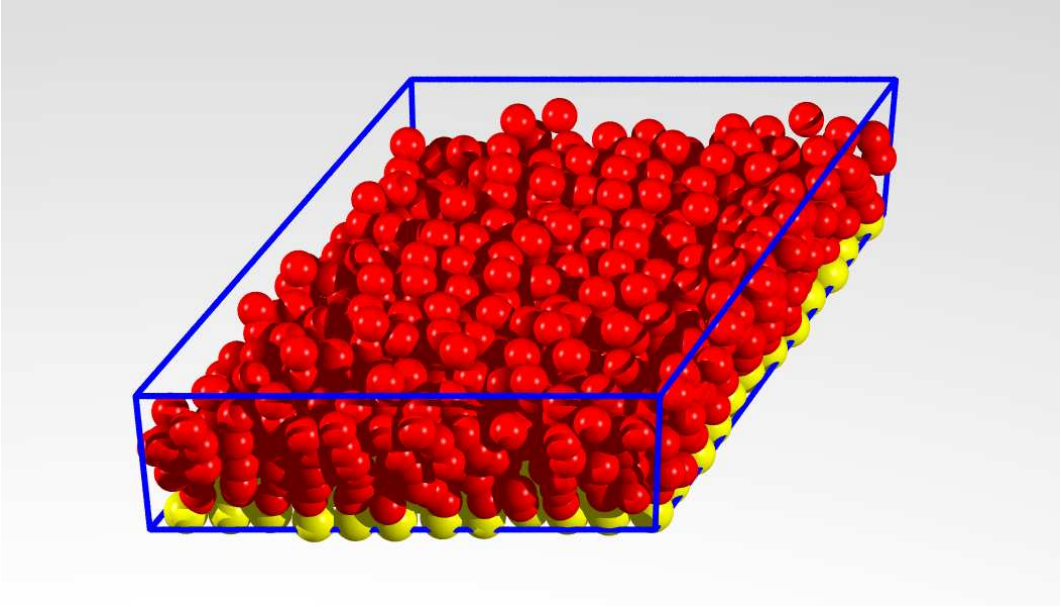


Abbildung 5.68: Snapshot der **ungeneigten LC-U-Phase**: $\Pi = 30, \quad T = 2.0$
Da die ungeneigte Phase nur bei relativ hohen Temperaturen existiert
beobachtet man schon deutliche thermische Unordnung.

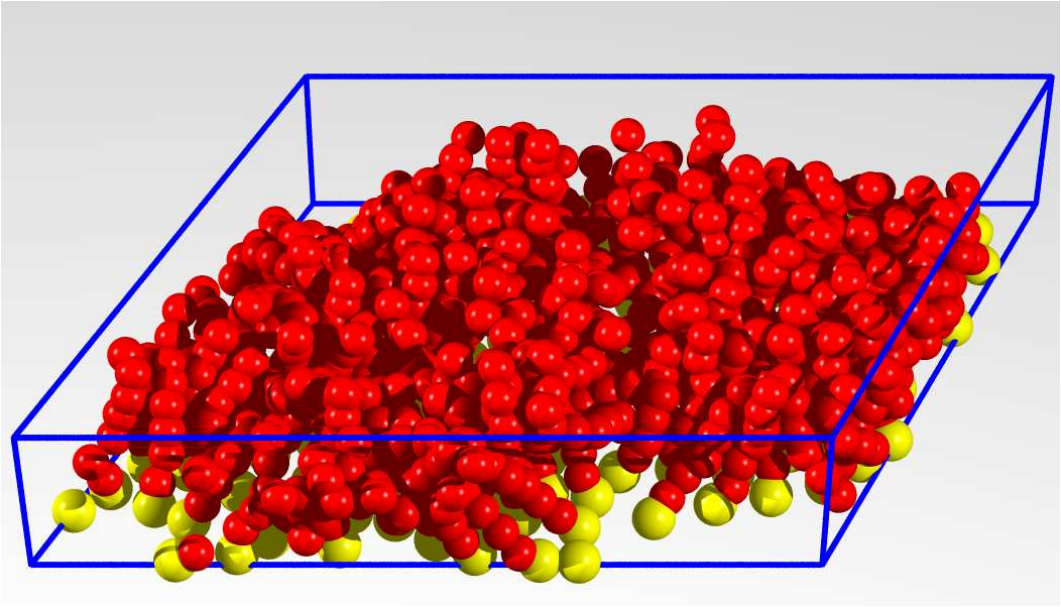


Abbildung 5.69: Snapshot der **ungeordneten LE-Phase**: $\Pi = 0.3, \quad T = 1.5$
Im Unterschied zur LC-U-Phase ist das Kopfgitter nicht mehr hexagonal geordnet
und es treten auch sehr große Neigungswinkel auf.

Die Abbildungen entsprechen jeweils gleichem „Beobachter-Standpunkt“ und gleichem „Bildwinkel“, so daß der Größenunterschied der Konfigurationen deren tatsächliche Ausdehnung wiedergibt.
Die „Blickrichtung“ verläuft parallel zur y-Achse, d.h. die Schräge der Simulationsbox ist nicht perspektivisch bedingt, sondern entstammt dem Scherwinkel α der Box-Grundfläche.

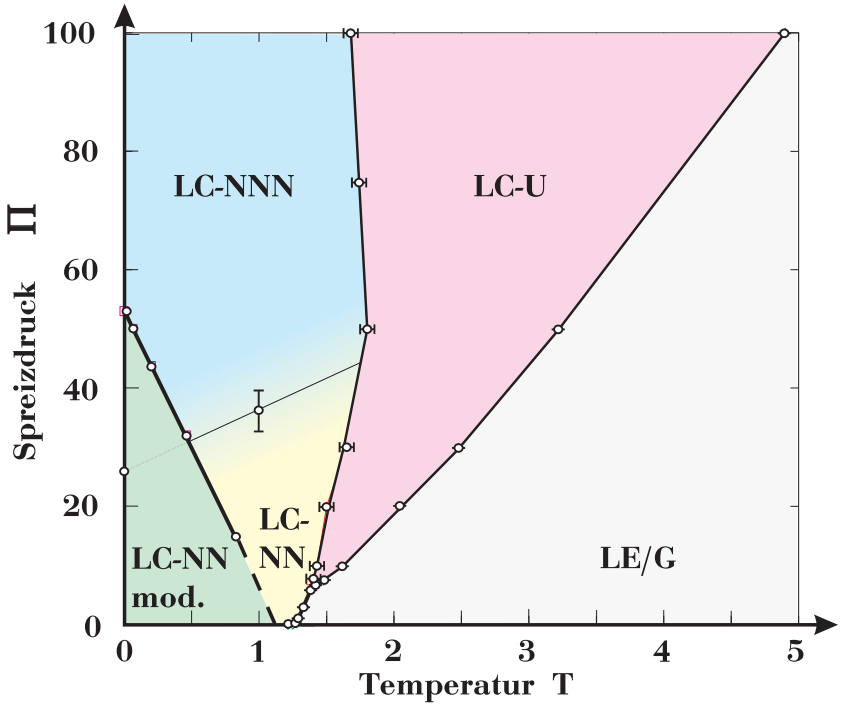


Abbildung 5.70: $\Pi(T)$-Phasendiagramm für das System $\sigma_H = 1.2\sigma$

Die durch thermodynamische Integration bestimmte Grenze zwischen der konventionell und moduliert geneigten Phase ergibt, daß die modulierte Phase selbst bei niedrigstem Druck nicht bis zum Schmelzübergang in die LE-Phase existiert, sondern dort stets eine konventionell geneigte Phase vorliegt, deren Ketten zu nächsten Nachbarn geneigt sind.
Das System großer Köpfe besitzt also zwei NN-Phasen – eine mit konventioneller Neigung bei hoher Temperatur und eine modulierte bei niedriger Temperatur.

Zwischen den beiden konventionell geneigten Phasen ist bei höherem Druck und höherer Temperatur, analog zum System kleiner Köpfe, eine Entkopplung der Neigungsrichtung vom Kopfgitter zu beobachten, so daß die Unterscheidung in einem augedehnten Gebiet entfällt und man dort von *einer* kollektiv geneigten Phase ausgehen muß.
Dies erschwert es, eine Grenze zwischen beiden Vorzugs-Neigungen anzugeben[16].
Andererseits erlaubt es aber die thermodynamische Integration entlang der Grenze zur modulierten Phase, weil kein zusätzlicher Übergang 1. Ordnung zwischen den beiden konventionellen Phasen gekreuzt wird.
Wegen der thermischen Unordnung, die den Unterschied zwischen modulierter und unmodulierten NN-Phase verschwimmen läßt, gelingt eine Integration über $T \gtrsim 0.9$ hinaus nicht.

[16] Verbindet man dennoch den Datenpunkt der Grundzustandsiteration zu $T = 0$ mit dem der Isothermen $T = 1$ erkennt man, däs die Verlängerung dieser Linie die LC-U-Phase etwa bei $\Pi = 50$ schneidet, d.h. dort, wo die Grenzlinie wieder zu niedrigeren Temperaturen abknickt (vgl. auch Abb. 5.71).

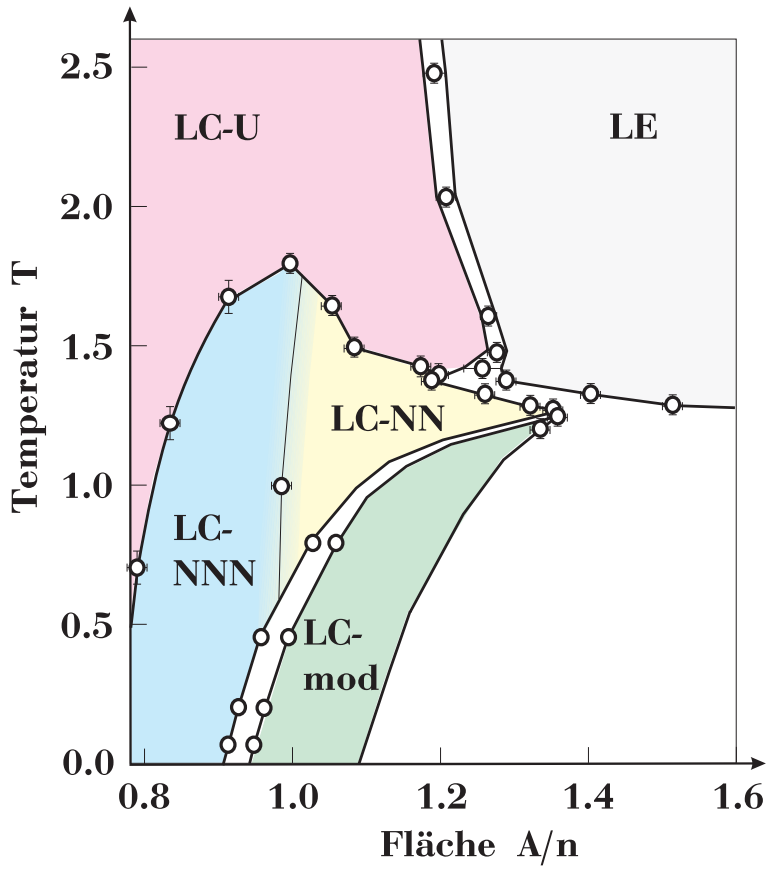


Abbildung 5.71: $T(A/n)$-Phasendiagramm für das System $\sigma_H = 1.2\sigma$

Eine Grenze zwischen der LE- und der Gas-Phase wurde hier nicht eingetragen, da der Übergang nicht lokalisiert werden konnte (vgl. Abschnitt 5.3.4).
Am rechten unteren Rand dieses Diagramms liegt aber sicher eine Gasphase vor.

Wie bei den kleinen Köpfen wurde für das $T(A)$-Phasendiagramm (Abb. 5.71) die Koexistenzfläche der LC- mit der Gasphase bei sehr niedrigen Temperaturen aus der Isobaren zu $\Pi := 0$ gewonnen.
Die Grenze zwischen der LC-U- und der LE-Phase ist mit Datenpunkten markiert, deren Lage einer mittleren Fläche am Übergang entspricht. Die Fehlerbalken und der Verlauf der doppelten Grenzlinie gibt die Flächen der beiden Phasen in der Nähe des Übergangs wieder.

Im $T(A)$-Diagramm scheint die Grenzlinie zwischen der modulierten und der konventionellen NN-Phase auf den Punkt zuzulaufen, an dem die LC-Phase die größte Fläche besitzt. Die durch thermodynamische Integration gewonnen Stützstellen weisen jeweils einen deutlichen Flächensprung auf.

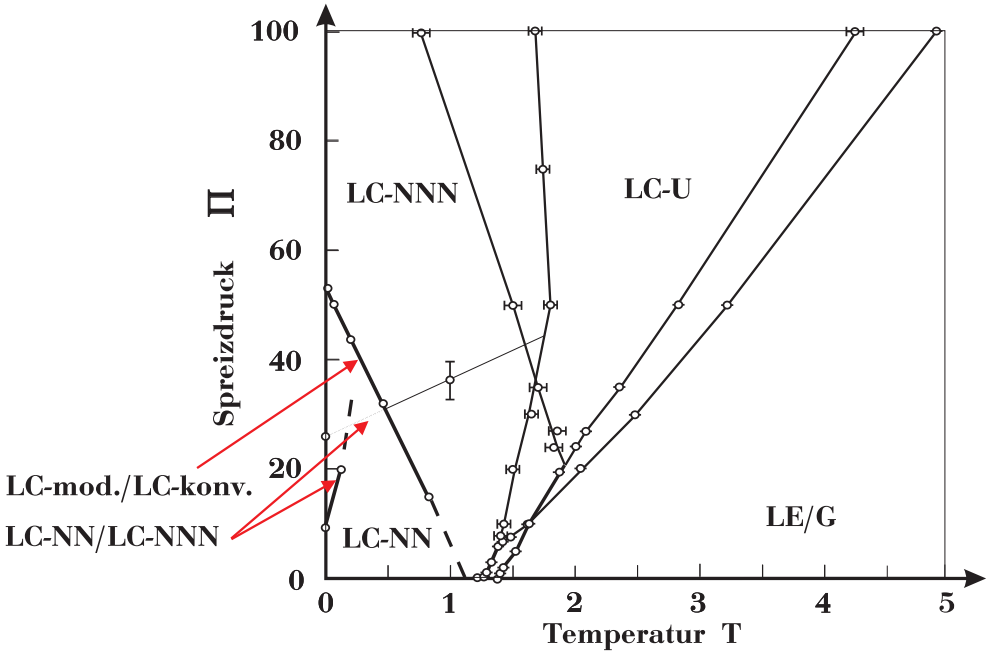


Abbildung 5.72: Überlagerung der $\Pi(T)$-Phasendiagramme der Systeme $\sigma_H = 1.1\sigma$ (kleine Köpfe) und $\sigma_H = 1.2\sigma$ (große Köpfe)

Im Gegensatz zum System kleiner Köpfe besitzt die Grenzlinie zwischen der geneigten und der ungeneigten LC-Phase eine ausgeprägtes Maximum[17]. Aus dem $\Pi(T)$-Diagramm (Abb. 5.70) erkennt man, daß dieses in der Nähe von $\Pi = 50$ liegt.
Dadurch ergibt sich, wenn man z.B. die Isotherme $T = 1.5$ verfolgt, ein sog. *re-entrance*-Verhalten, bei dem mit sinkenden Druck bsw. zunehmender Fläche zuerst die LC-U-Phase zugunsten der LC-NNN verlassen und danach in die LC-U-Phase zurückgekehrt wird.

Ein Vergleich der $\Pi(T)$-Phasendiagramme (Abb. 5.72) beider untersuchter Kopfgruppen zeigt, daß es für die Abhängigkeit von der Kopfgruppen-Größe σ_H anscheinend keine ähnlich einfache Gesetzmäßigkeit wie für die Kettenlänge gibt (vgl. Abschnitt 5.3.6).
Der Schmelzübergang LC↔LE wird von der Kopfgröße nur wenig beeinflußt.

5.3.6 Abhängigkeiten von der Kettenlänge

Experimentell ist seit langem bekannt [44], daß die Länge der Alkylgruppe der Moleküle nur relativ geringen Einfluß auf das Phasendiagramm der Monolagen hat. Für die betrachteten Fettsäuren bewirkt eine zusätzliche CH_2-Gruppe im wesentlichen nur eine Verschiebung der Temperaturskala um ca. 8 K nach oben [10].
Um diesen Effekt im Modell zu untersuchen, wurden neben der üblichen Kettenlänge

[17] Durch die großen Kopfgruppen bedarf es noch höherer Drücke, um die Fläche pro Kette auf $A/n \lesssim 0.9$ zu komprimieren, so daß der letzte Punkt auf der LC-U/LC-NNN-Linie bereits zu $\Pi = 300$ gehört.

$l = 7$ auch Simulationen mit $l = 6$ und $l = 8$ durchgeführt[18]. Sogenannte *odd-even*-Effekte, wie sie in Molekulardynamik-Simulationen mit Pseudoatom-Modellen [16] beobachtet werden, bestehen darin, daß sich Ketten mit ungerader bzw. gerader Zahl von C-Atomen deutlich verschieden verhalten. Derartige Effekte sind im hier betrachteten Modell nicht zu erwarten, da sich aufgrund des isotropen Kettenquerschnitts verschieden lange Moleküle nicht in ihrer Symmetrie unterscheiden.
Analog zur Wahl im Abschnitt 5.3.7 wurden wiederum die beiden charakteristischen Drücke $\Pi = 1$ und $\Pi = 50$ untersucht.

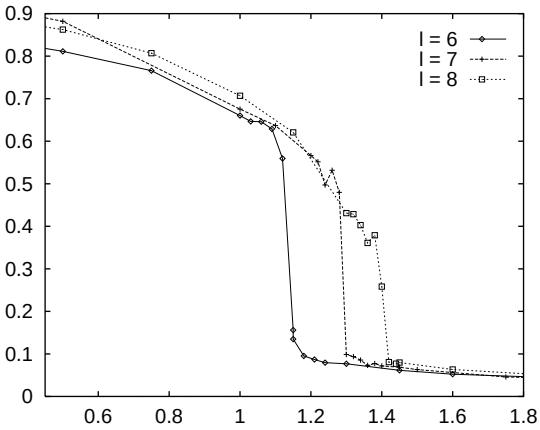


Abbildung 5.73: Ordnungsparameter Ψ_6 vs. T für $\sigma_H = 1.2\sigma$, $\Pi = 1$

Aus dem Ordnungsparameter Ψ_6, der das Aufschmelzen des Kopfgruppengitters wiedergibt, bestimmt man für $\Pi = 1$ folgende Übergangstemperaturen:

Länge l	6	7	8
$T_{\mathrm{LC-NN} \leftrightarrow \mathrm{LE}}$	1.142	1.293	1.399

Hieraus ergibt sich eine Temperaturverschiebung von $\Delta T \approx +0.13$ pro effektivem Monomer bzw. $\Delta T \approx +15$ K pro zusätzlicher CH_2-Gruppe.

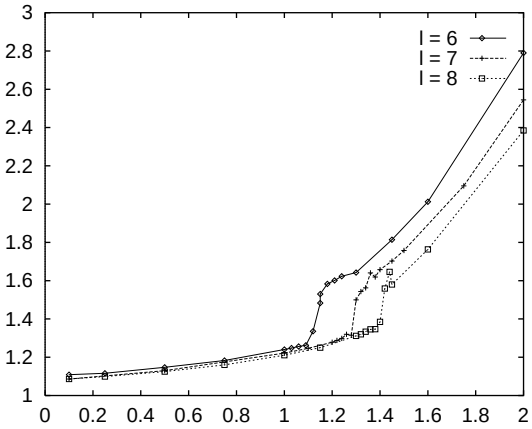


Abbildung 5.74: Fläche A/n vs. T für $\sigma_H = 1.2\sigma$, $\Pi = 1$

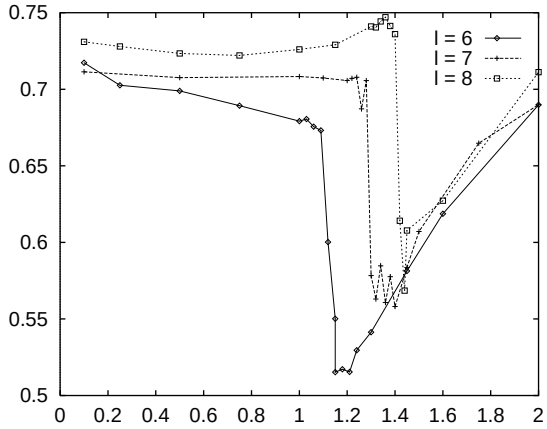


Abbildung 5.75: Neigungswinkel ϑ vs. T für $\sigma_H = 1.2\sigma$, $\Pi = 1$

Während die Kettenlänge auf die Dichte der kondensierten Phase keinen merklichen Einfluß ausübt, zeigt der Neigungswinkel eine Tendenz zu stärkerer Neigung bei längeren Ketten.

Bei $\Pi = 50$ erhält man für den Übergang LC-NNN $\leftrightarrow$ LC-U aus dem gewichteten Histogramm $\varrho(\vartheta)$ des Neigungswinkels (vgl. Abschnitt 5.2.2) die Temperaturen

[18] Ein zusätzliches effektives Monomer entspricht 2-3 zusätzlichen CH_2-Gruppen.

Länge l	6	7	8
$T_{\mathrm{LC-NNN} \leftrightarrow \mathrm{LC-U}}$	1.71±0.10	1.80±0.10	1.93±0.10

die z.B. mit dem Verlauf des mittleren Neigungswinkels $\langle\vartheta\rangle$ konsistent sind.
Im Modell liegt somit eine Verschiebung von $\Delta T \approx +0.11$ pro weiterem Monomer vor, was im Experiment $\Delta T \approx +13$ K pro zusätzlicher CH_2-Gruppe entspräche.

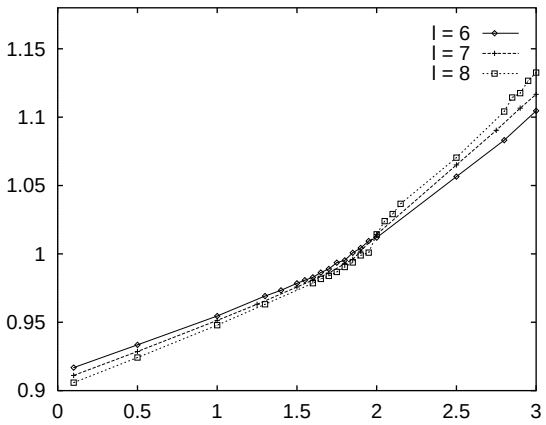


Abbildung 5.76: Fläche A/n vs. T für $\sigma_H = 1.2\sigma$, $\Pi = 50$

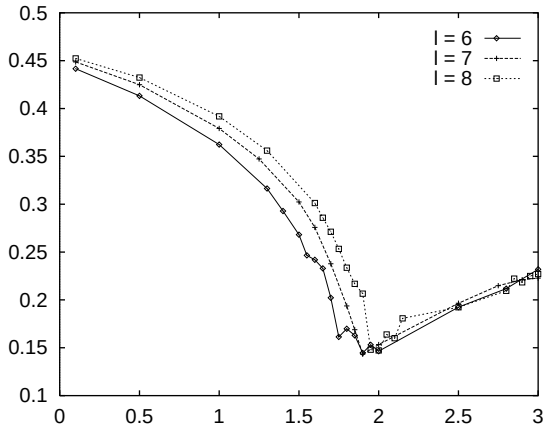


Abbildung 5.77: Neigungswinkel ϑ vs. T für $\sigma_H = 1.2\sigma$, $\Pi = 50$

In der geneigten Phase LC-NNN steigt die Dichte mit der Kettenlänge leicht an, wohingegen in der ungeneigten Phase LC-U die kürzeren Ketten das dichtere System bilden. Die Abhängigkeit des Neigungswinkels entspricht der Phase LC-NN, d.h. die längeren Ketten sind stärker geneigt.

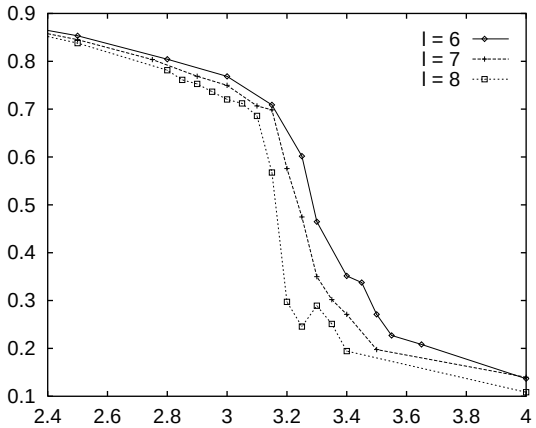


Abbildung 5.78: Ordnungsparameter Ψ_6 vs. T für $\sigma_H = 1.2\sigma$, $\Pi = 50$

Bei höherem Druck ($\Pi = 50$) findet man den Übergang LC-U $\leftrightarrow$ LE bei den Temperaturen

Länge l	6	7	8
$T_{\mathrm{LC-U} \leftrightarrow \mathrm{LE}}$	3.35	3.28	3.18

Die Verschiebung erfolgt also **zu niedrigeren** Temperaturen und beträgt im Modell $\Delta T \approx -0.09$ bzw. in SI-Einheiten $\Delta T \approx -10$ K pro zusätzlicher CH_2-Gruppe.

Diese Umkehrung der erwarteten Temperaturverschiebung ergibt zusammen mit dem Anstieg der Übergangstemperatur $T_{\mathrm{LC-NNN} \leftrightarrow \mathrm{LC-U}}$ eine Verkleinerung des Gebiets der ungeneigten Phase LC-U.
Vom Standpunkt der inneren Energie könnte man dieses Verhalten damit erklären, daß in dieser Phase der Kettenabstand von der Größe der Kopfgruppen $\sigma_H > \sigma$ bestimmt wird, d.h. die Schwanzmonomere befinden sich in einer energetisch unbefriedigenden

Situation. Mit wachsender Kettenlänge wird das System daher eher in eine der angrenzenden Phasen übergehen.
Da der Übergang LC-U↔LE bei relativ hohen Temperaturen erfolgt, muß auch die Entropie in die Betrachtung einbezogen werden. Aus dieser Sicht wächst die Entropie der Kettenkonformation mit steigender Länge, weswegen die ungeordnete Phase aufgrund der höheren Entropie die niedrigere freie Energie hätte und dadurch stabilisiert würde.

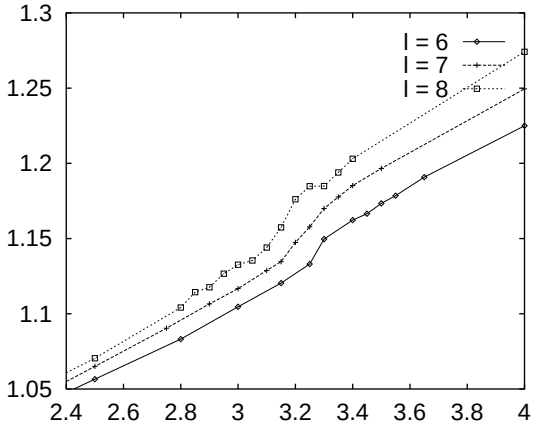


Abbildung 5.79: Fläche A/n vs. T für $\sigma_H = 1.2\sigma$, $\Pi = 50$

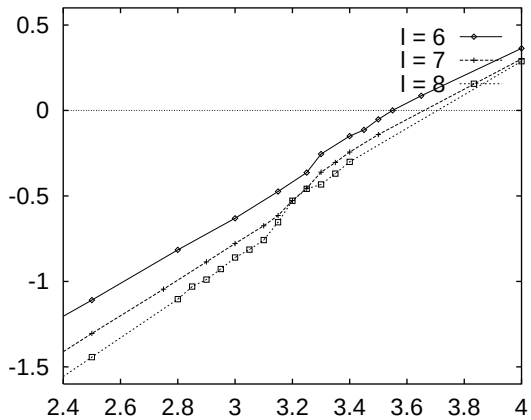


Abbildung 5.80: innere Energie U/N vs. T für $\sigma_H = 1.2\sigma$, $\Pi = 50$

Bei hohen Drücken ist der Flächensprung am Übergang LC-U↔LE sehr gering, so daß dessen geringe Verschiebung zwischen den verschiedenen Kettenlängen kaum zu erkennen ist.
Im Regime niedrigen Druckes $\Pi = 1$ hatte das System mit den längsten Ketten die kleinste spezifische Fläche – sowohl in der LC- als auch in der LE-Phase.
Bei hohem Druck liegen in der kollektiv geneigten LC-NNN-Phase diese Verhältnisse zwar auch vor, kehren sich aber in der LC-U- *und* in der LE-Phase um, d.h. daß dort das System der kürzesten Ketten die kleinste spezifische Fläche hat.

Die innere Energie U/N pro Monomer liegt umso tiefer, je länger die Ketten sind. Dies könnte damit zusammenhängen, daß bei längeren Ketten der Anteil der „Bulk"-Monomere im Inneren der Monolage gegenüber den „Oberflächen"-Monomeren, d.h. Kopf- und End-Monomeren, größer ist und daher mehr Potentialzentren optimal koordiniert sind.

5.3.7 Finite-Size-Effekte

Für eine detaillierte Finite-Size-Analyse wäre es nötig, die Systemgröße wesentlich, d.h. um Größenordnungen zu variieren.
Eine deutliche Verkleinerung des Systems ist weder sinnvoll noch möglich, da unterhalb von 8×8 Ketten aufgrund der periodischen Randbedingungen Mehrfachwechselwirkungen mit ein und demselben Teilchen eintreten können. Nach oben sind der

Systemgröße dadurch Grenzen gesetzt, daß mit der Teilchenzahl der Speicherbedarf stark ansteigt (vgl. Abschnitt 3.3) und die Zahl der notwendigen Rechenoperationen pro MonteCarlo-Schritt ebenfalls anwächst.
Daher sei hier auf ein explizite Finite-Size-Untersuchung verzichtet und nur ein grober Vergleich zweier Systemgrößen $n = 144, l = 7$ und $n = 900, l = 7$ angestellt.
Das bisher präsentierte Phasendiagramm kennt vereinfacht zwei Druck-Regimes; bei niedrigem Druck den Übergang aus der LC-NN-Phase in die LE-Phase und bei hohem Druck die Übergänge LC-NNN ↔ LC-U ↔ LE. Entsprechend wurden Isobaren für zwei charakteristische Drücke $\Pi = 1$ und $\Pi = 50$ aufgenommen.

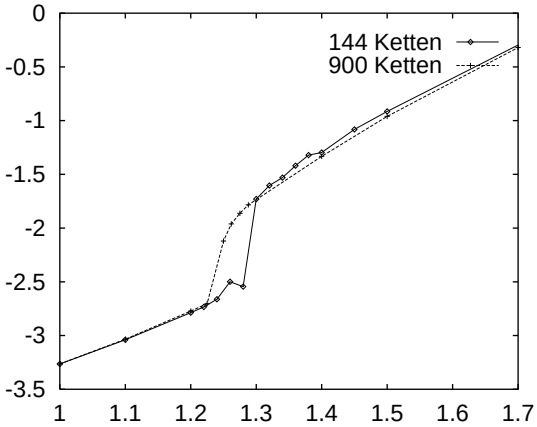


Abbildung 5.81: spez. Enthalpie H/N vs. T für $\sigma_H = 1.2\sigma$, $\Pi = 1$

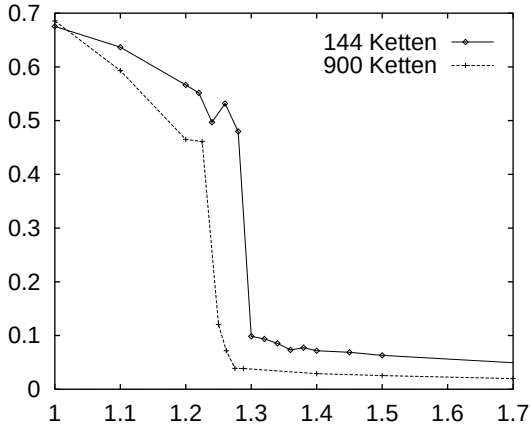


Abbildung 5.82: Ordnungsparameter Ψ_6 vs. T für $\sigma_H = 1.2\sigma$, $\Pi = 1$

Bei niedrigem Druck ergibt sich gute Übereinstimmung zwischen den beiden Systemgrößen. Man findet eine Verschiebung zu niedrigeren Temperaturen bei größeren Systemen, was zu einer Abweichung der Übergangstemperaturen $T_{\mathrm{LC-NN}\leftrightarrow\mathrm{LE}}$ von $\Delta T \approx 0.05$ führt.
Während die spezifische Enthalpie H/N von der Teilchenzahl tatsächlich kaum beeinflußt wird, ist im Ordnungsparameter Ψ_6 deutlich zu erkennen, daß das größere System in der ungeordneten Phase dem „korrekten“ Wert $\Psi_6 = 0$ viel näher kommt, d.h. einen geringeren *Finite-Size-Tail* zeigt.

Da bei $\Pi = 1$ die LC-Phase als NN-moduliert vorliegt und die Simulation kaum in der Lage ist, die Streifenbreite der Überstruktur zu äquilibrieren (vgl. Abschnitt 5.3.1), bestehen zwischen verschieden großen Systemen Probleme mit der exakten Vergleichbarkeit der LC-Phasen.

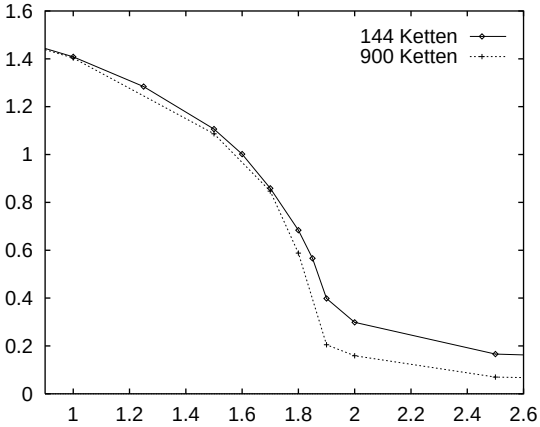


Abbildung 5.83: Neigungsmaß R_{xy} vs. T für $\sigma_H = 1.2\sigma$, $\Pi = 50$

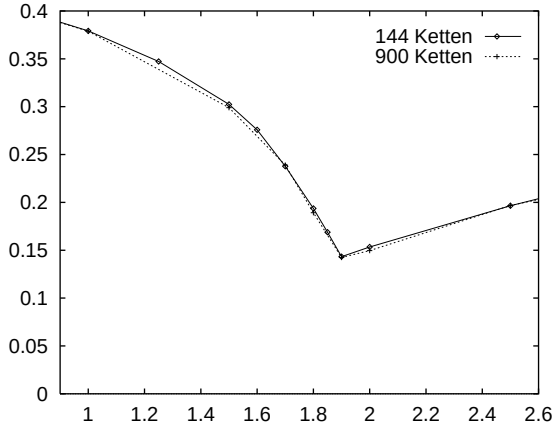


Abbildung 5.84: Neigungswinkel ϑ vs. T für $\sigma_H = 1.2\sigma$, $\Pi = 50$

Bei <u>höherem Druck</u> treten bei beiden Phasenübergängen keine deutlichen Verschiebungen der Übergangstemperaturen auf.
Für LC-NNN $\leftrightarrow$ LC-U ist keine Abweichung der Temperaturen festzustellen. Wie zu erwarten mittelt sich in der ungeneigten Phase LC-U die Restneigung durch die größere Kettenzahl vollständiger aus, was zu einem Wert von R_{xy} führt, der näher an Null liegt.

Am Übergang LC-U $\leftrightarrow$ LE zeigt der Verlauf von Ψ_6 zwar einen deutlichen Unterschied, die daraus bestimmten Übergangstemperaturen stimmen dennoch innerhalb ihrer Fehler überein.

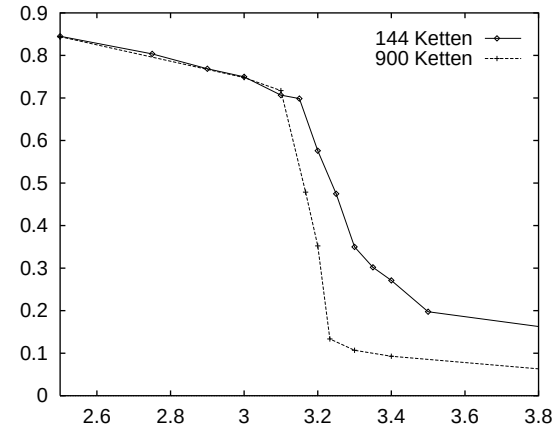


Abbildung 5.85: Ordnungsparameter Ψ_6 vs. T für $\sigma_H = 1.2\sigma$, $\Pi = 50$

Im Zusammenhang mit *Finite-Size*-Untersuchungen betrachtet man häufig Ordnungsparameter- oder Energie-Kumulanten [11][45].
Neben der Kumulante 4. Ordnung (vgl. (2.4.14)) wurde auch die enstprechende Kumulante 2. Ordnung $g_2(T) := \langle [x]^2 + [y]^2 \rangle / \langle \sqrt{[x]^2 + [y]^2} \rangle^2$ ausgewertet. Da in g_2 niedrige-

re Momente der Verteilung von R_{xy} eingehen, sollte der Verlauf aufgrund der besseren Statistik „glatter“ sein.

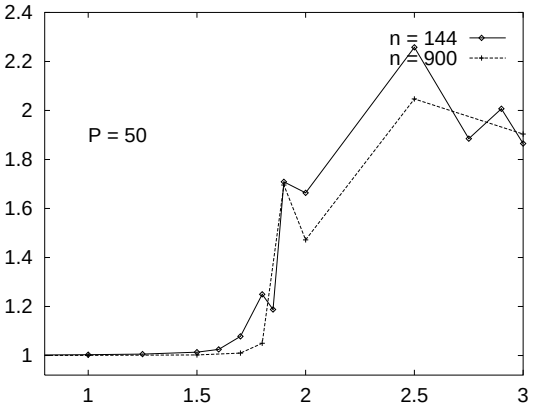


Abbildung 5.86: Kumulante 4. Ordnung g_4 vs. T für $\sigma_H = 1.2\sigma$, $\Pi = 50$ (LC-NNN↔LC-U)

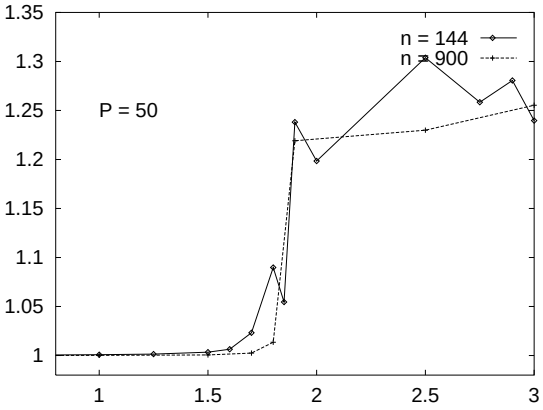


Abbildung 5.87: Kumulante 2. Ordnung g_2 vs. T für $\sigma_H = 1.2\sigma$, $\Pi = 50$ (LC-NNN↔LC-U)

Die beiden Abbildungen zeigen, daß mit der Länge der Simulationsläufe, die genügt um den Verlauf der Observablen und die Lage der Übergänge zu bestimmen – d.h. typischerweise 200.000-400.000 MC-Schritte – noch keine ausreichenden statistischen Daten zur Berechnung der Kumulanten vorliegen.
Soweit man dies aus den Graphen beurteilen kann, verlaufen $g_4(T)$ bzw. $g_2(T)$ für beide Systemgrößen nur wenig unterschiedlich. Um eine Aussage über die Abhängigkeit von der Teilchenzahl treffen zu können, wäre es nötig, diese über einen größeren Bereich zu variieren, insbesondere zu deutlich größeren Teilchenzahlen zu gehen.

Zusammenfassend kann gesagt werden, daß eine Systemgröße von $n = 144, l = 7$ ausreicht, um das Phasen-Verhalten sinnvoll zu charakterisieren. Die *Finite-Size*-Effekte sind gering genug, um auf das Verhalten eines unendlich ausgedehnten Systems schliessen zu können. Eine genauere Bestimmung der Phasengrenzen erscheint nicht nötig, da das Modell nicht soweit spezifiziert wurde, daß konkrete experimentelle Meßergebnisse reproduziert werden müßten.
Auch in den experimentellen Arbeiten über Langmuir-Monolagen sind „Finite-Size-Effekte“ beobachtet worden. *Albrecht et al.* [2] berichten, daß im Koexistenzbereich der LC- und LE-Phase die Isothermen keinen gänzlich waagrechten Verlauf zeigen und der Übergang daher nicht völlig dem Bild eines Übergangs 1. Ordnung entspricht. Da sie Verunreinigungen als Fehlerquelle ausschließen folgern sie, daß in den Monolagen je ca. 10^2–10^3 Moleküle Aggregate bilden, die gemeinsam die Phasenumwandlung durchführen.
Mittels bildgebender Methoden wie der Fluoreszenzmikroskopie (PFM) [46] kann man aber die Koexistenz einzelne Domänen der Phasen LC und LE sichtbar machen, was einen Übergang 1. Ordnung anzeigt.

5.4 Sehr große Köpfe

Bei der Modulation, die im Abschnitt 5.3.1 beschrieben wurde, hatte sich bereits gezeigt, daß das Verhalten des Systems bei tiefen Temperaturen weitgehend dadurch bestimmt wird, daß die großen, repulsiven Kopfgruppen nahezu dichtest-gepackt sind, während die übrigen Monomere versuchen, über diesem hexagonalen Gitter mit zu großer Gitterkonstante ihre Lennard-Jones-Potentiale zu optimieren.
Vergrößert man den Durchmesser der Kopfgruppen σ_H gegenüber den Schwanz-Monomeren σ noch weiter, gelangt man in einen Bereich, in dem aufgrund des Größenunterschiedes eine kollektive Neigung nicht mehr genügt, um die Konfiguration energetisch zu optimieren. Vereinzelte Cluster aufeinander zu geneigter Ketten, sog. *Micelles* [44][18][23] leisten dies dann besser.
Eine solche Vergrößerung des Molekülkopfes erreicht man experimentell z.B. durch eine Senkung des *p*H-Wertes des Wassers [63], oder durch eine andere chemische Struktur der Kopfgruppen [67].
Zur Untersuchung dieses Verhaltens wurden bei konstanter Temperatur[19] $T = 0.1$ und konstantem Druck $\Pi = 1$ Systeme mit Kopfgrößen $\sigma_H = 1.3\sigma, 1.5\sigma$ und 1.8σ untersucht. Die beobachteten Konfigurationen sind vermutlich nur metastabil, aber charakteristisch in dem Sinn, daß sie deutlich von den bisher beschriebenen, kollektiv geneigten Phasen abweichen.
Für die drei Fälle wurden als Anfangskonfigurationen je eine kollektiv NN-geneigte Phase und ein kollektiv ungeneigtes System verwendet.

[19] Bei dieser tiefen Temperatur sind Entropie-Effekte noch unbedeutend und das Verhalten kann mit Energie-Betrachtungen erklärt werden.

Bei $\sigma_H = 1.3\sigma$ liegt noch eine kollektive Neigung vor, aber die Alkanschwänze bilden kein regelmäßiges Gitter mehr. Offensichtlich ist es für sie günstiger, sich über größere Bereiche mit einem Abstand von etwa σ zusammen zu lagern und dafür Lücken in der Schicht in Kauf zu nehmen. Das Kopfgruppengitter ist dicht-gepackt und zeigt kaum noch neigungsbedingte Verzerrung.

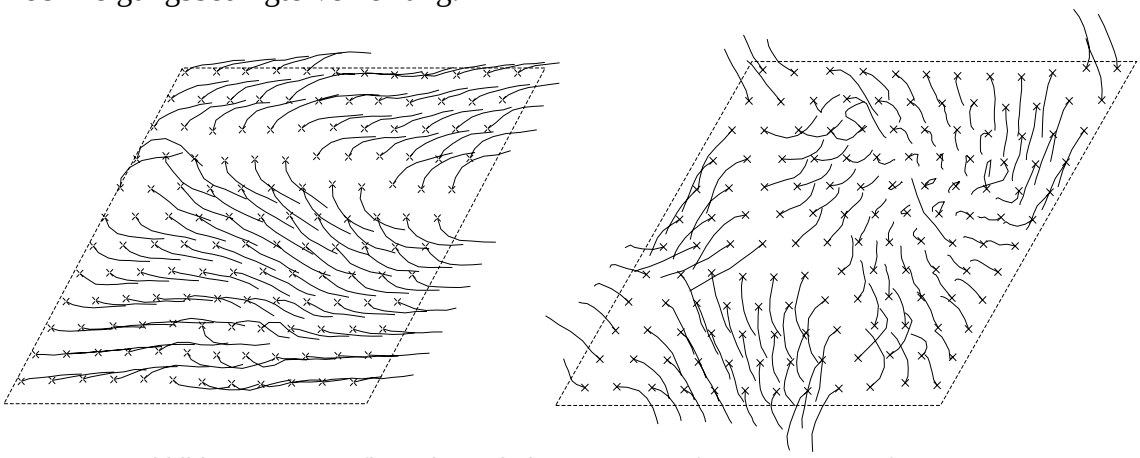


Abbildung 5.88: Konfigurationen bei $\Pi = 1$, $T = 0.1$ für $\sigma_H = 1.3\sigma$, aufgesetzt als nicht-modulierte LC-NN-Phase (links) bzw. als senkrechte LC-U-Phase (rechts)

Da sich die Ketten in dieser Anordnung stark krümmen, scheint das Bondwinkelpotential weich genug zu sein, die Flexibilität der Alkanschwänze nicht wesentlich zu beeinträchtigen.

Geht man mit den Kopfgruppen weiter zu $\sigma_H = 1.5\sigma$ sind in der Konfiguration, die geneigt aufgesetzt wurde, noch Reste der ursprünglichen kollektiven Neigung zu erkennen. Die ungeneigte Anordnung zeigt bereits Clusterbildung, d.h. das Verhalten, welches man für Systeme mit starkem Größenunterschied zwischen Kopfgruppe und Alkanschwanz erwartet [18]. In beiden Snapshots wurde eine raumfüllende Darstellung gewählt, d.h. der Durchmesser der Kugeln entspricht σ bzw. σ_H.

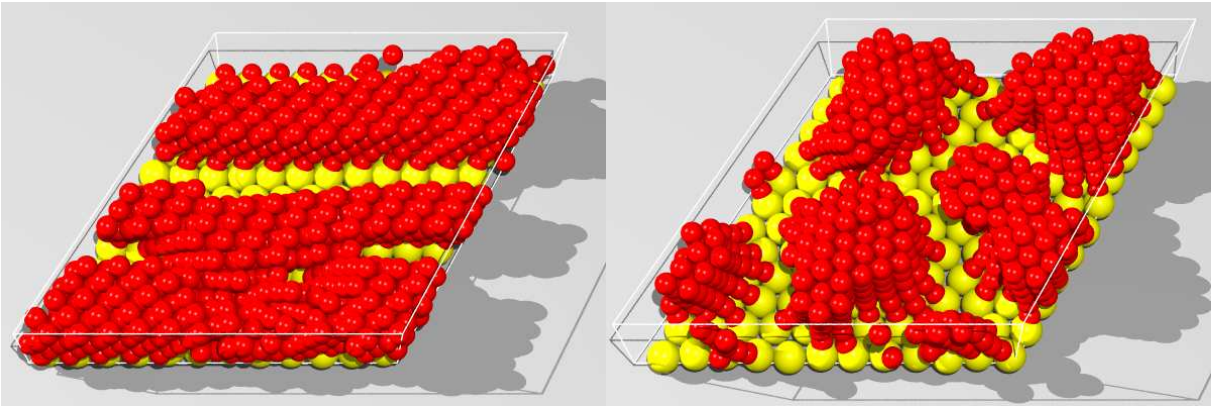


Abbildung 5.89: Konfigurationen bei $\Pi = 1$, $T = 0.1$ für $\sigma_H = 1.5\sigma$, aufgesetzt als nicht-modulierte LC-NN-Phase (links) bzw. als senkrechte LC-U-Phase (rechts) In der linken Konfiguration ergibt sich ein Neigungswinkel $\langle\vartheta\rangle \approx 48°$.

Der Simulationsalgorithmus, der Konformationsänderungen lediglich durch Bewegung einzelner Monomere bewirkt, ist hier kaum noch in der Lage, das System zu äquilibrieren. Da in derartigen Zuständen die Möglichkeit der Umlagerung ganzer Ketten gegeben ist, könnte[20] *Configurational Biased Monte Carlo* (Abschnitt 3.2.2) hier deutliche Vorteile bringen.

Bei einer Kopfgröße von $\sigma_H = 1.8\sigma$ ist schließlich jede kollektive Neigung verschwunden und es liegen nur noch Cluster verschiedener Größe vor. Während die Cluster in sich durch die Lennard-Jones-Potentiale der Alkanschwänze gebunden sind, resultiert der Zusammenhang des Kopfgruppengitters allein durch den äußeren Druck, da zwischen den Köpfen rein repulsive Soft Core-Potentiale wirken.

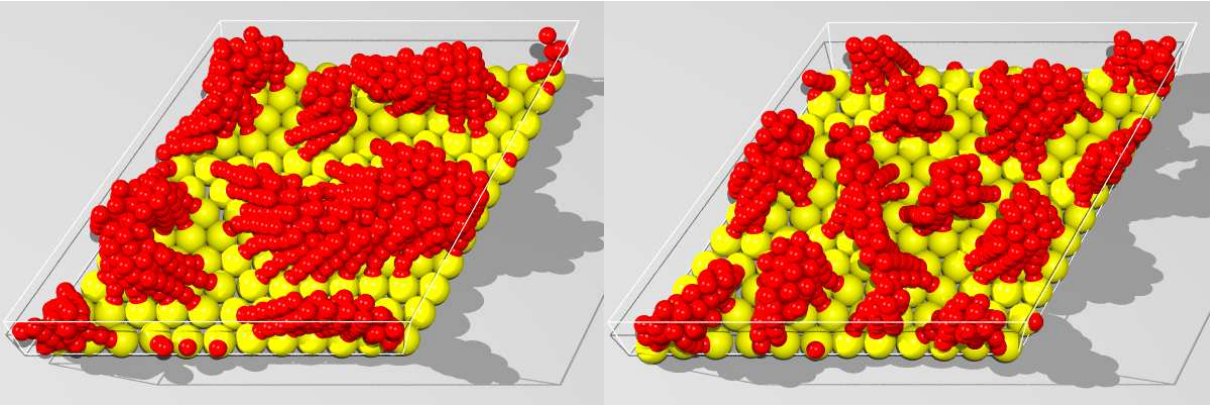


Abbildung 5.90: Konfigurationen bei $\Pi = 1$, $T = 0.1$ für $\sigma_H = 1.8\sigma$, aufgesetzt als nicht-modulierte LC-NN-Phase (links) bzw. als senkrechte LC-U-Phase (rechts)

Beim Aufheizen dieses Systems beobachtet man, daß bei einer Temperatur $T \approx 0.6$ das Kopfgitter aufschmilzt, was sich an einem starken Abfall des hexagonalen Ordnungsparameters Ψ_6 bemerkbar macht. Erst bei einer Temperatur von $T \approx 1.0$ geht auch die Neigungskorrelation K_{NN} der Ketten zurück.

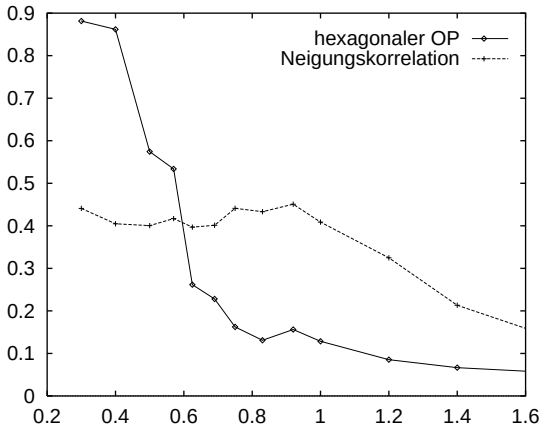


Abbildung 5.91: Ψ_6 und K_{NN} vs. T bei Kopfgröße $\sigma_H = 1.8\sigma$

Dieses Verhalten läßt die Vermutung zu, daß im Temperaturbereich $0.6 < T < 1.0$ die Cluster in sich zwar noch geordnet, gegeneinander aber leicht beweglich sind.

[20] CCBMC konnte hierauf aus Zeitgründen nicht mehr erprobt werden, weil beim Übergang von rechteckiger auf scherbare Box auf die Fortschreibung dieses Algorithmus' verzichtet wurde.

Kapitel 6

Zusammenfassung und Ausblick

Die vorliegende Arbeit präsentiert als modellhafte Beschreibung von Langmuir-Monolagen die Ergebnisse von Monte Carlo-Simulationen amphiphiler Moleküle an einer homogenen Grenzfläche.
Die Moleküle, die einkettigen Fettsäuren entsprechen, bestehen aus effektiven Monomeren die durch semiflexible Bonds miteinander verbunden sind. Während die übrigen Monomere sich im dreidimensionalen Kontinuum über der Grenzfläche bewegen können, dürfen die größeren Kopfgruppen, die rein repulsiv wechselwirken, die Ebene, welche die Wasser-Luft-Grenze abbildet, nicht verlassen. Die azimutale Asymmetrie realer Kohlenwasserstoff-Ketten ist in diesem Modell nicht wiedergegeben. Dadurch kann eine Anordnung der Rückgrat-Ebenen der Alkanketten in der Simulation nicht beobachtet werden.
Die Simulation behandelt das System im $N\Pi T$-Ensemble, d.h. bei konstanter Teilchenzahl, Temperatur und konstantem Druck.
Die hierfür nötigen Bewegungsversuche, die die Fläche der Simulationbox verändern, sind so angelegt, daß nicht nur die Größe, sondern auch die Form der Boxgrundfläche fluktuiert.
Es wurden zwei Varianten des Modells untersucht, wobei im „System kleiner Köpfe“ die Kopfgruppen der Ketten um 10% größer sind als die Schwanzmonomere und im „System großer Köpfe“ dieser Größenunterschied 20% beträgt. Für beide Systeme wurde aus den Simulationsdaten das Phasendiagramm bestimmt.

Das System kleiner Köpfe zeigt vier verschiedene Phasen die auch experimentell beobachtet werden. Neben der ungeordneten flüssig-expandierten Phase sind dies drei kondensierte Phasen, die sich darin unterscheiden, daß die Ketten gegenüber dem hexagonalen Kopfgitter hin zu nächsten oder zu übernächsten Nachbarn geneigt sind, oder kollektiv senkrecht auf der Grenzfläche stehen.

Bei niedrigem Druck und niedriger Temperatur findet man die NN-Phase, bei höherem Druck die NNN-Phase, die bei höherer Temperatur in die ungeneigte Phase übergeht. Beim weiteren Aufheizen gelangt man schließlich an den Schmelzübergang in die un-

geordnete Phase.
Um die jeweils 6-fache Entartung der NN- bzw NNN-Richtung reproduzierten zu können, erwies es sich als notwendig, anstatt einer rechteckigen Boxgrundfläche eine parallelogramm-förmige zu verwenden, deren Scherwinkel wie die Kantenlängen fluktuieren kann. Andernfalls wurden nur solche Neigungen korrekt dargestellt, deren Neigungsrichtungen parallel zu einer der Boxkanten verlaufen.

Während experimentell noch ein Übergang 1. Ordnung zwischen der flüssig-expandierten und der ebenfalls ungeordneten Gas-Phase eintritt, erreicht die Simulation stetig das Regime des zweidimensionalen, idealen Gases. Dies läßt die Vermutung zu, daß die Grenze LE/G in einem kritischen Punkt endet, der bei niedriger Temperatur und niedrigem Druck noch in dem Bereich liegt, in dem die Simulation die LC-Phase favorisiert. Die Suche nach dem Übergang LE/G erfolgte dann stets jenseits der Phasengrenzlinie.

Im System großer Köpfe treten alle diese Phasen auch auf.
Zusätzlich liegt bei niedrigem Druck und tiefer Temperatur eine NN-Phase vor, die sich durch eine Überstruktur von den konventionell geneigten Phasen unterscheidet. Diese „modulierte“ Phase wurde weder zusammen mit NNN-Neigung noch im System kleiner Köpfe beobachtet und ist durch eine Verdrehung des Kopfgitters gegen die kollektive NN-Neigung der Ketten ausgezeichnet.
Zwischen der modulierten und den konventionellen Phasen tritt eine weiträumige Hysterese auf.

Innerhalb des Gebiets der kondensierten Phasen konnte die Gibbs'sche freie Energie durch thermodynamische Integration ausgewertet werden. Die hierfür nötigen Anfangswerte wurden aus der harmonischen Näherung des Systems bei $T = 0$ gewonnen. Um aus den Simulationsdaten zu Grundzustandskonfigurationen zu gelangen wurde ein Gradientenverfahren eingesetzt das es zudem erlaubte, die Phasengebiete entlang der $T = 0$-Linie zu bestimmen.
Durch die thermodynamische Integration konnte z.B. die Grenzlinie der modulierten und der konventionellen Phasen festgelegt werden.
Um diese Methode auf den Übergang zwischen der kondensierten und der ungeordneten Phase anzuwenden würde auch ein Referenzzustand definierter freier Energie in der Gasphase benötigt. Einen solchen zu bestimmen wurde nicht mehr versucht, sollte aber grundsätzlich möglich sein.

Der Charakter des Übergangs zwischen der ungeneigten Phase LC-U und der ungeordneten LE-Phase konnte nicht abschließend geklärt werden.

Zusammenfassend kann gesagt werden, daß das verwendete, stark vergröberte Modell in der Lage ist, einige wesentliche Eigenschaften von Langmuir-Monolagen, wie z.B. die verschiedenen Neigungsrichtungen, wiederzugeben.
Als typischer Vorzug von Simulationen lassen sich die Einflüsse einzelner Modellparameter studieren. So kann die Veränderung der Kettenlänge, -steifigkeit oder der Kopfgruppengröße untersucht werden.

Einige Erweiterungen des simulierten Modells würden die Untersuchung zusätzlicher Problemstellungen ermöglichen:

Im Zusammenhang mit Surface Freezing wird den Fluktuationen, die innerhalb der Oberflächenlage auftreten und sich in Verschiebungen der Ketten entlang ihrer Längsrichtung äußern, große Bedeutung beigemessen [68]. Wollte man derartige Fluktuationen auch in dem hier simulierten Modell zulassen, müßte man die Bindung der Köpfe in die Grenzfläche „weicher“ gestalten, z.B. durch ein Oberflächenpotential das attraktiv auf die Köpfe und repulsiv auf die Schwanzmonomere wirkt.
Beispiele hierfür finden sich in MD-Simulationen [36][62] oder [14].

Um sich der Simulation von Lipid-Doppellagen zuzuwenden, die der Grundbaustein biologischer Membranen sind, müßte man die Bindung der Kopfgruppen an die Wasseroberfläche gänzlich aufgeben und neben den Lipidketten auch die Wassermoleküle im verwendeten Modell berücksichtigen [24].

Ohne wesentliche Änderungen könnte man das Verhalten polydisperser Monolagen untersuchen oder zusätzlich „Hindernisse“ in die Grenzfläche einfügen, die z.B. Protein-Molekülen entsprechen.

Anhang A

Hamiltonian für das $N\Pi T$-Ensemble

Ausgangspunkt sei eine unendlich hohe Box mit parallelogrammförmiger Grundfläche (L_x, L_y, α) mit $N := n \cdot l$ Teilchen darin für deren Konfiguration $\underline{r}$ gilt:

$$\underline{r} = \{r_{ix}, r_{iy}, r_{iz}\}, \quad i = 1 \ldots N, \quad 0 \leq r_{ix} < L_x, \quad 0 \leq r_{iy} < L_y \sin\alpha, \quad 0 \leq r_{iz} < \infty \tag{A.0.1}$$

Die Box kann auch über ihre Fläche A und ein Seitenverhältnis a parametrisiert werden

$$A = L_x L_y \sin\alpha \qquad a := \frac{L_x}{L_y \sin\alpha} \tag{A.0.2}$$

Bei einem Volumen-Move, der z.B. $L_x \mapsto L'_x$ ändert – analog für L_y oder α – verändern sich nicht nur die Box-Variablen, sondern auch die Teilchenkoordinaten:

$$\begin{pmatrix} r_{ix} \\ r_{iy} \\ r_{iz} \end{pmatrix} \mapsto \begin{pmatrix} r_{ix} \cdot L'_x/L_x \\ r_{iy} \\ r_{iz} \end{pmatrix} \quad \text{nun mit } \; 0 \leq r_{ix} < L'_x \;\; \forall i \tag{A.0.3}$$

Dies kann man durch Verwendung *skalierter* Teilchenkoordinaten $\underline{s}$ vermeiden [22].

$$\begin{gathered} \underline{s} = \{s_{ix}, s_{iy}, s_{iz}\} \;\; i = 1 \ldots N \quad 0 \leq s_{ix} < 1 \quad 0 \leq s_{iy} < 1 \;\; 0 \leq s_{iz} < \infty \\ s_{ix} := r_{ix}/L_x \quad s_{iy} := r_{iy}/L_y \sin\alpha \quad s_{iz} = r_{iz} \quad \forall i \\ \underline{r} = \underline{r}(\underline{s}, L_x, L_y, \sin\alpha) \end{gathered} \tag{A.0.4}$$

Das $3N$-dimensionale Volumenelement $\underline{\mathrm{d}r}$ transformiert sich dabei wie

$$\begin{aligned} \underline{\mathrm{d}r} &= \mathrm{d}r_{1x}\mathrm{d}r_{1y}\mathrm{d}r_{1z} \ldots \mathrm{d}r_{Nx}\mathrm{d}r_{Ny}\mathrm{d}r_{Nz} \\ &= L_x \mathrm{d}s_{1x}\, L_y \sin\alpha \mathrm{d}s_{1y}\, \mathrm{d}r_{1z} \ldots L_x \mathrm{d}s_{Nx}\, L_y \sin\alpha \mathrm{d}s_{Ny}\, \mathrm{d}s_{Nz} \\ &= A^N \underline{\mathrm{d}s} \end{aligned} \tag{A.0.5}$$

Da im NAT-Ensemble nur die Größe der Fläche, nicht notwendigerweise aber ihre Form konstant ist, lautet die kanonische Zustandssumme[1]

$$Z_{NAT} = \frac{1}{\sigma^{3N+2}} \int_0^\infty \mathrm{d}a \int_0^\pi \mathrm{d}\alpha \int_{\mathrm{Box}(A,a,\alpha)} \underline{\mathrm{d}r} \exp(-\beta V(\underline{r})), \tag{A.0.6}$$

woraus man die $N\Pi T$-Zustandssumme durch eine Laplace-Transformation erhält.

$$\begin{aligned}
\sigma^{3N+2} Z_{N\Pi T} &= \int_0^\infty \mathrm{d}A \exp(-\beta \Pi A) Z_{NAT} \\
&= \int_0^\infty \mathrm{d}A \int_0^\infty \mathrm{d}a \int_0^\pi \mathrm{d}\alpha \int_{\mathrm{Box}(A,a,\alpha)} \underline{\mathrm{d}r} \exp\left(-\beta(V(\underline{r}) + \Pi A)\right) \\
&= \int_0^\infty \mathrm{d}A \int_0^\infty \mathrm{d}a \int_0^\pi \mathrm{d}\alpha \left(\prod_{i=1}^N \int_0^{L_x} \mathrm{d}r_{ix} \int_0^{L_y \sin\alpha} \mathrm{d}r_{iy} \int_0^\infty \mathrm{d}r_{iz} \right) \exp\left(-\beta(V(\underline{r}) + \Pi A)\right) \\
&= \int_0^\infty \mathrm{d}A \int_0^\infty \mathrm{d}a \int_0^\pi \mathrm{d}\alpha \left(\prod_{i=1}^N \int_0^1 \mathrm{d}s_{ix} \int_0^1 \mathrm{d}s_{iy} \int_0^\infty \mathrm{d}s_{iz} \right) A^N \exp\Big(-\beta(V(\underline{s}, A, a, \alpha) + \Pi A)\Big) \\
&= \int_0^\infty \mathrm{d}A \int_0^\infty \mathrm{d}a \int_0^\pi \mathrm{d}\alpha \int_{\mathrm{S}} \underline{\mathrm{d}s} \exp\Big(-\beta(V(\underline{s}, A, a, \alpha) + \Pi A - N k_B T \ln A)\Big) \\
&= \int_0^\infty \mathrm{d}L_x \int_0^\infty \mathrm{d}L_y \int_0^\pi \mathrm{d}\alpha \ \ldots \qquad \text{mit} \qquad \mathrm{S} := ([0;1]^2 \times \mathbb{R})^N \\
&\quad \ldots \int_{\mathrm{S}} \underline{\mathrm{d}s} \exp\Big(-\beta(V(\underline{s}, L_x, L_y, \alpha) + \Pi L_x L_y \sin\alpha - N k_B T \ln(L_x L_y \sin\alpha))\Big)
\end{aligned} \tag{A.0.7}$$

Letzteres ist gerade die kanonische Zustandssumme eines Systems mit Konfigurationsraum $(L_x, L_y, \alpha, \underline{s})$ das dem effektiven Hamiltonian

$$H_{eff} := V(\underline{s}, L_x, L_y, \alpha) + \Pi A - N k_B T \ln A \tag{A.0.8}$$

unterliegt.

Die Impulskoordinaten, die eigentlich in der Zustandstsumme zu berücksichtigen währen, wurden hier weggelassen, da in den Monte Carlo-Simulationen nur statische Eigenschaften untersucht werden. Die Observablen $X(\underline{R})$ hängen also nur von den Orten $\underline{R}$, nicht aber von den zug. Impulsen $\underline{P}$ ab. Bei der Berechnung von Erwartungs-

[1] Der Faktor $\sigma^{-(3N+2)}$ mit der Lennard-Jones-Länge σ macht den Ausdruck dimensionslos.

werten kürzen sich demnach die Impulse heraus

$$\begin{aligned}\langle X(\underline{R})\rangle &= \frac{\int \underline{\mathrm{d}R\mathrm{d}P} X(\underline{R}) \exp(-\beta(H(\underline{R}) + T(\underline{P})))}{\int \underline{\mathrm{d}R\mathrm{d}P} \exp(-\beta(H(\underline{R}) + T(\underline{P})))} \\ &= \frac{\int \underline{\mathrm{d}R} X(\underline{R}) \exp(-\beta H(\underline{R}))}{\int \underline{\mathrm{d}R} \exp(-\beta H(\underline{R}))}.\end{aligned} \tag{A.0.9}$$

$T(\underline{P})$ bezeichnet hierbei die kinetische Energie.

Als weitere Vereinfachung wurde in dieser Darstellung unterschlagen, daß die periodischen Randbedingungen in einem Parallelogramm ausgewertet werden und nicht in einem Rechteck. Man hätte also eigentlich

$$r_{iy} \cos\alpha \;\leq\; r_{ix} \;<\; L_x + r_{iy} \cos\alpha \quad \forall i \tag{A.0.10}$$

zu beachten.

Die Transformation (A.0.3) impliziert, daß das Simulationsprogramm bei einem Volumen-Move tatsächlich *alle* Koordinaten der N Monomere in gleicher Weise reskaliert. Man kann sich auch Volumen-Moves vorstellen, wie sie z.B. *Frenkel* und *Smit* vorschlagen [22] und wie sie hier im Abschnitt (3.2.1) diskutiert werden, in denen nur die Schwerpunkte bzw. die Koordinaten der Köpfe reskaliert werden, die innere Konfigurationen der Ketten aber unberührt bleiben, da sie als starre Gebilde eine Translation erfahren.
Zwischen beiden Varianten zu wählen ist keine reine Frage der Programm-Implementation, sondern es ist dabei zu beachten, daß im letzteren Fall die Zahl der transformierten Koordinaten in (A.0.5) nur noch n und nicht mehr N beträgt.
Entsprechend ist als effektiver Hamiltonian

$$H_{eff}^{transl} := V(\underline{s}, L_x, L_y, \alpha) + \Pi A - n k_B T \ln A \tag{A.0.11}$$

zu verwenden.

Anhang B

Berechnung der Kopplungskonstanten

Für eine harmonische Näherung des Hamiltonians benötigt man die Matrix aller 2. Ableitungen $\partial^2 H/\partial u_i \partial u_j$ (Hessematrix) des Grundzustands-Hamiltonians $H = U + \Pi A$ der von $F = (3l-1)n + 3$ Freiheitsgraden $\{u_i\}$ abhängt.
Morris und *Ho* [49] beschreiben ein Verfahren, diese Kopplungskonstanten direkt aus den Korrelationen $\langle u_i u_j \rangle - \langle u_i \rangle \langle u_j \rangle$ zu gewinnen, die während der Simulation aufgenommen werden. Die Hessematrix hier wurde jedoch mittels numerischer Differentiation der Gesamtenthalpie bestimmt, wobei die Grundzustandskonfiguration Endprodukt einer Iteration gemäß (5.1.2) war.
Betrachtet man nur zwei Auslenkungen x, y der Freiheitsgrade, so lautet die Taylorreihe

$$\begin{aligned} H(x,y) = \quad & H(0,0) + x\partial_x H + y\partial_y H + \\ & \tfrac{1}{2}x^2\partial_{xx}H + \tfrac{1}{2}y^2\partial_{yy}H + \tfrac{1}{2}xy\partial_{xy}H + \tfrac{1}{2}yx\partial_{yx}H + \ldots \end{aligned} \tag{B.0.1}$$

Mit einer Umformung[1] erhält man

$$\begin{aligned} & H(x,y) - H(0,0) = \\ & y^2\left[\tfrac{1}{2}\partial_{yy}H\right] + y^1\left[\tfrac{1}{2}x\partial_{xy}H + \tfrac{1}{2}x\partial_{yx}H + \partial_y H\right] + y^0\left[\tfrac{1}{2}x^2\partial_{xx}H + x\partial_x H\right]. \end{aligned} \tag{B.0.2}$$

Wenn man also zu festem $x \neq 0$ die Auslenkung y systematisch variiert, erwartet man ein Polymom zu finden, dessen linearer Koeffizient $[\frac{1}{2}x\partial_{xy}H + \frac{1}{2}x\partial_{yx}H + \partial_y H]$ ist.
Nun kann man diese linearen Koeffizienten für verschiedene x bestimmen, wiederum ein Polynom anpassen und erhält als linearen Koeffizienten $\frac{1}{2}[\partial_{xy}H + \partial_{yx}H]$. Da die Matrix der 2. Ableitungen symmetrisch ist, hat man somit zwei der Matrixelemente bestimmt.
Um diese Berechnungen durchzuführen muß man die Energiedifferenz $\Delta E = H(x,y) - H(0,0)$ zweier Systemkonfigurationen kennen, die sich nur wenig unterscheiden.
Dies genau und schnell zu bestimmen ist aber gerade die „Spezialität“ des Monte Carlo-Programms. Man muß daher nur zwei wesentliche Modifikationen vornehmen:

[1] Wenn um den Grundzustand entwickelt wird, müßten alle 1. Ableitungen $\{\partial_i H\}$ verschwinden.

- Die zu ändernden Koordinaten nicht mehr zufällig auswählen, sondern in systematischer Reihenfolge, Richtung und Weite
- Die Monte Carlo-Moves vor der Auswertung des Metropoliskriteriums abbrechen und sich die Energiedifferenz ausgeben lassen

Natürlich kann auch hier die Periodizität des NNN-Grundzustand und die endliche Reichweite des Lennard-Jones-Potentials ausgenutzt werden, was den Rechenaufwand drastisch reduziert[2].
Da bei diesen Berechnungen kleinste Enthalpiedifferenzen betrachtet werden, treten alle numerischen Ungenauigkeiten der Simulation überdeutlich zutage.
Beispielsweise wurden die Lennard-Jones- und Soft Core-Potentiale aus Geschwindigkeitsgründen in 10^5 Schritten tabellarisiert. Während dies bei den Simulationen gegenüber dem korrekten, analytischen Potential keinen Unterschied ergab, war es nicht möglich, mit den gerasterten Potentialen die Kopplungskonstanten genauer als auf 2 Stellen zu berechnen. Mit Verwendung der analytischen Potentialformel traten diese Probleme nicht mehr auf.

[2] Selbst dann verbrauchte die Berechnung aller Matrixelemente noch ca. 10 CPU-Stunden.

Anhang C

Thermische Expansion und harmonische Näherung

Bei einem Kristall verschwindet in harmonischer Näherung der Ausdehnungskoeffizient für $T \mapsto 0$ [6]. Die hier durchgeführten Berechnungen sollten dieses Verhalten ebenfalls zeigen.
Da die Abmessungen der Simulationsbox neben den Monomerauslenkungen als gleichberechtigte Freiheitsgrade in die harmonische Näherung eingehen, ist dies nicht offensichtlich. Die übliche Argumentation (vgl. *Ashcroft, Mermin* [6]) beschreibt nämlich eine Volumenänderung als kollektive Auslenkung aller Monomere voneinander weg bzw. aufeinander zu, während nun dieser Vorgang einer reinen Anregung der zusätzlichen Boxfreiheitsgrade entspricht.
Unter Verwendung der harmonischen Zustandssumme (4.4.42) erhält man die temperaturabhängige Fläche $S(T)$ der Box als

$$\langle S(T)\rangle = \frac{1}{Z_{N\Pi T}} \frac{e^{-\beta H^0}}{\sigma^{F-1}} \int_{-\infty}^{\infty} \mathrm{d}A_1 ... \int_{-\infty}^{\infty} \mathrm{d}A_F \, S(\vec{A}) \prod_{s=1}^{F} \exp(-\beta\lambda_s A_s^2) \,. \tag{C.0.1}$$

Für die Fläche S findet man wegen (4.2.11) in den Amplituden $\vec{A}$ die Darstellung

$$S(\vec{A}) = \left(L_x^0 + \sum_{i=1}^{F} A_i(\vec{\epsilon}_i)_{F-2}\right) \cdot \left(L_y^0 + \sum_{j=1}^{F} A_j(\vec{\epsilon}_j)_{F-1}\right) \cdot \sin\left(\alpha^0 + \sum_{k=1}^{F} A_k(\vec{\epsilon}_k)_F\right) \tag{C.0.2}$$

was sich unter der Annahme kleiner Auslenkungen für α[1] umschreiben läßt zu

$$\begin{aligned} S(\vec{A}) = S^0 \cdot \left(1 + \sum_i A_i c_x(i)\right) \cdot \left(1 + \sum_j A_j c_y(j)\right) \cdot \\ \left(1 + \sum_k A_k c_\alpha(k) - \frac{\tan^2\alpha^0}{2}\left[\sum_k A_k c_\alpha(k)\right]^2\right) \end{aligned} \tag{C.0.3}$$

$$c_x(i) := \tfrac{1}{L_x^0}(\vec{\epsilon}_i)_{F-2} \qquad c_y(j) := \tfrac{1}{L_y^0}(\vec{\epsilon}_j)_{F-1} \qquad c_\alpha(k) := \tfrac{\cos\alpha^0}{\sin\alpha^0}(\vec{\epsilon}_k)_F \,,$$

[1] Man könnte auch $\sin\alpha$ anstatt α als F-ten Freiheitsgrad verwenden.

wenn man Terme bis $O(T^1)$ vollständig berücksichtigt. Nach dem Ausmultiplizieren erhält man

$$\begin{aligned}\frac{S(\vec{A})}{S^0} &= 1 + \sum_i A_i\Big(c_x(i) + c_y(i) + c_\alpha(i)\Big) + \sum_{ijk} A_i A_j A_k c_x(i) c_y(j) c_\alpha(k) \\ &+ \sum_{ij} A_i A_j\Big(c_x(i)c_y(j) + c_x(i)c_\alpha(j) + c_y(i)c_\alpha(j) - \frac{\tan^2\alpha^0}{2} c_\alpha(i)c_\alpha(j)\Big) + O(T^2)\,.\end{aligned} \tag{C.0.4}$$

Setzt man diese Reihen in das Integral (C.0.1) und vertauscht die Integrationen mit den Summationen kann man sich folgende Beziehungen zunutze machen:

$$\int_{-\infty}^{\infty} A\, e^{-\beta\lambda A^2} \mathrm{d}A = 0 = \int_{-\infty}^{\infty} A^3\, e^{-\beta\lambda A^2} \mathrm{d}A \qquad \int_{-\infty}^{\infty} A^2 e^{-\beta\lambda A^2} \mathrm{d}A = \frac{1}{2\beta\lambda} \int_{-\infty}^{\infty} e^{-\beta\lambda A^2} \mathrm{d}A \tag{C.0.5}$$

Mit deren Hilfe vereinfacht sich (C.0.1) zu

$$\frac{\langle S(T)\rangle}{S^0} = 1 + \sum_{i=1}^{F} \frac{k_B T}{2\lambda_i}\Big(c_x(i)c_y(i) + c_x(i)c_\alpha(i) + c_y(i)c_\alpha(i) - \frac{\tan^2\alpha^0}{2} c_\alpha^2(i)\Big)\,. \tag{C.0.6}$$

Ein verschwindender Ausdehnungskoeffizient liegt also vor, falls

$$k_B \sum_{i=1}^{F} \frac{c_x(i)c_y(i) + c_x(i)c_\alpha(i) + c_y(i)c_\alpha(i) - \frac{\tan^2\alpha^0}{2} c_\alpha^2(i)}{2\lambda_i} \stackrel{!}{=} 0 \tag{C.0.7}$$

erfüllt ist.
Für beide Fälle, für die Abschnitt 4.5 die Übergangstemperatur zwischen der NN- und NNN-Phase bestimmt wurde, seien hier die Flächenausdehnungskoeffizienten

$$\phi := \frac{1}{n} \lim_{T\to 0} \frac{S(T) - S^0}{T} \approx \frac{S(T) - S^0}{nT} \tag{C.0.8}$$

bei niedrigen Temperaturen $T \leq 0.03$ angegeben, die in der Simulation gefunden wurden:

		ϕ_{sim}	ϕ_{harm}
$\sigma_H = 1.1\sigma$	NN	0.050	$\lvert\phi\rvert < 10^{-6}$
$\Pi = 20$	NNN	0.060	$\lvert\phi\rvert < 10^{-6}$
$\sigma_H = 1.2\sigma$	mod	0.028	$\lvert\phi\rvert < 10^{-6}$
$\Pi = 50$	NNN	0.032	$\lvert\phi\rvert < 10^{-6}$

Die nach (C.0.7) berechneten Werte ϕ_{harm} sind – verglichen mit den Simulationswerten – in guter Näherung als **Null** anzusehen. Erst eine Berücksichtigung anharmonischer Effekte, die sich als Flächenabhängigkeit der Eigenwerte auswirken, kann die Flächenausdehnung erklären.

Anhang D

Programm-Listing

Die hier wiedergegebenen Programmteile umfassen nur die eigentliche Simulation im $N\Pi T$-Ensemble mit scherbarer Box und unter Verwendung einer Verlet-Tabelle.
Das Paket läuft, übersetzt mit den jeweils generischen C-Compilern [37], auf den skalaren Architekturen

IBM	RS6000	AIX	IBM	PowerPC	AIX
SGI	PowerChallenge	IRIX	SGI	MIPS R10000	IRIX
Intel	Pentium	Linux	DEC	Alpha	DigUnix
SUN	Ultra2-Sparc	Solaris	HP	RS6000	HP-UX

Die Maximalgeschwindigkeit für ein System von $n = 144, l = 7$ Teilchen liegt in der kondensierten Phase bei 25000 MC-Schritte/CPU-Stunde, in der LE-Phase darüber.

Die Programmteile besitzen folgende grobe Gliederung:

`switchboard.h`	:	technische Parameter wie maximale Teilchenzahlen usw. Schalter für kollektive Moves und andere Zusatzfunktionen
`MC.c`	:	Hauptprogramm
`MC_s.c`	:	alle Initialisierungen, Anfangskonfiguration, Verlet-Update
`MC_e.c`	:	Auswertung der intermolekularen Wechselwirkungen
`MC_m.c`	:	„Messung“ aller Observablen

Zur Orientierung seien noch einige wichtige Variablen-Namen ihren Entsprechungen in dieser Arbeit zugeordnet:

Ensemble	:	$T =$ `temp`, $\Pi^{ext} =$ `pressure`, $N =$ `ntot`, $n =$ `npm`, $l =$ `lpm`
Box	:	$L_x =$ `SSX`, $L_y =$ `SSY`, α[rad] $=$ `alpha`, $L_y \sin\alpha =$ `SHY`
Lennard-Jones-Potential	:	$\epsilon =$ `u0`, $\sigma =$ `ljmin`
Soft Core-Potential	:	$\epsilon_{\mathrm{SC}} =$ `u0_hh`, $\sigma_H =$ `ljmin_hh` (Kopf–Kopf) $\epsilon_{HM} =$ `u0_hm`, $\sigma_{HM} =$ `ljmin_hm` (Monomer–Kopf)
Bondlängen-Potential	:	$\epsilon_{\mathrm{BL}} =$ `sprng`, $d_0 =$ (`bond_min + bond_max`)/2 $d_{\mathrm{BL}} =$ (`bond_max - bond_min`)/2
Bondwinkel-Potential	:	$\epsilon_{\mathrm{BW}} =$ `cospr`

Monomerkoordinaten: $\vec{r}_i = \begin{pmatrix} \texttt{xmm[i]} \\ \texttt{ymm[i]} \\ \texttt{zmm[i]} \end{pmatrix}$ kartesisch/absolut $\qquad \begin{pmatrix} \xi \\ \eta \end{pmatrix} = \begin{pmatrix} \texttt{xbox[i]} \\ \texttt{ybox[i]} \end{pmatrix}$ schiefwinklig/relativ

Weitere Details sind den Kommentaren im Code selbst zu entnehmen.

switchboard.h 1/1

```
/* bekannte Architekturen : */

#undef ALPHA_DEC
#undef HP
#undef IBM_RISC
#undef LPC
#undef SGI
#undef SPARC

/* ------------------------ */
/* Architektur bestimmen    */

#define IBM_RISC
/* ------------------------ */

#define MAX_LPM 8
#define MAX_MONOMERS 1024
#define MAX_POLYMERS 144
#define WW_triangle (MAX_MONOMERS*MAX_MONOMERS/2)
#define MAX_LJWW 256       /* fuer 2. Dimension der Verlet-Tabelle :   */
                           /* maximale Anzahl von WW-Partnern          */
#define LJ_clip 1000000.0  /* innere Abschneidehoehe des LJ-Potentials */
#define LJ_grid 100000     /* Zahl der Rasterpunkte der LJ-Tabellen    */
#define LJ_hh_grid 50000
#define LJ_hm_grid 50000

/* moegliche Zusatzfunktionen : */

#undef kollektiv    /* wird von PT und CCB aktiviert  !    */
#undef SHEAR     /* Schermoves (trotz PR !) deaktiviert */
#undef SIGHAND   /* Signalhandler nicht aktiv           */
#undef PT_Flag   /* keine Ketten -Translationen         */
#undef VH_Flag   /* keine homogenen Volumen-Moves       */
#undef DIAGNOSE  /* keine Ueberpruefung der Konsistenz  */
#undef NVT_Flag  /* NPT, nicht NVT                      */

/* ------------------------- */
/* Zusatzfunktionen aktiv :  */

#define SIGHAND
#define SHEAR
/* ------------------------- */

#ifdef VH_Flag   /* entweder nur homogene Volumen-Moves, */
#undef SHEAR     /* oder eine scherbare Box zulassen.    */
#endif

#ifdef PT_Flag
#define kollektiv
#define CCB_TRIALS MAX_LPM
#endif
```

```c
#include "switchboard.h"

#define  min(x,y ) ((x)<(y) ? (x):(y))
#define  max(x,y ) ((x)>(y) ? (x):(y))
#define  period(x,size ) (fmod(((x)+32*(size)),(size)))

#if (defined(SPARC)||defined(SGI)||defined(LPC)||defined(HP))
#define itrunc(x ) ((int) (x))
#endif

#ifndef TRUE
#define TRUE    1
#endif

#ifndef FALSE
#define FALSE   0
#endif

struct part
{
        int     pn;
        struct part *next;
};

extern struct part      ****cell,*list;
extern int      npm,lpm,number_of_cells_x,number_of_cells_y,number_of_cells_z ;
extern int      dim,ntot,nsav,nsavb;
extern int      *jpx,*jmx,*jpy,*jmy,*jmz,*jpz ;
extern int      *xcell,*ycell,*zcell;
extern double   mcs,mcsmax,mcs_aequi;
extern double   LJ[LJ_grid+1],LJ_hh[LJ_hh_grid+1],LJ_hm[LJ_hm_grid+1],engy[MAX_M
ONOMERS];
extern double   *pLJ[3],ljsc[3],ljcq[3],lju0[3],ljmq[3];
extern char     lj_type[MAX_MONOMERS][MAX_MONOMERS];
extern double  *xmm,*ymm,*zmm,*xbox,*ybox;
extern double  SSX,inv_SSX,SSY,SHY,inv_SHY,alpha,alpha_io,ctg_alpha,cosa2,system
_size_z;
extern double  CSX,CSY,cell_size_z;
extern double  hvar,volvar,alphvar,alphvar_io,lj_safety,temp,u0,pressure;
extern double  ljcut,ljcut_hm,ljcut_hh,ljcut_max;
extern double  u0_hh,u0_hm,lj0,lj0_hh,lj0_hm,ljmin,ljmin_hh,ljmin_hm;
extern double  ljcut_quadrat,ljcut_hh_quadrat,ljcut_hm_quadrat ;
extern double  lj_scala,lj_hh_scala,lj_hm_scala;
extern double  bond_limit_quadrat,bond_length,bond_min,bond_max,sprng ;
extern double  bond_min_quadrat,bond_max_quadrat ;
extern double  cospr,costry1,costry2,costry3;
extern double  *cospot,*energy,*newcpot,*newenrg;
extern double  *xn2t,*yn2t,*zn2t,*xendt,*yendt,*zendt;
extern double  *xax,*yax,*zax,*ncth,*nphi;
extern char    **swtch,heads;
extern unsigned long *histoxy,*histoz,*histothe,*histophi,*histodst,*historpvf;

extern double x0ver[MAX_MONOMERS],y0ver[MAX_MONOMERS],z0ver[MAX_MONOMERS];
extern double d_diff_max,d_diff_max2,rad_verlet,skin_verlet,skin;
extern int    tab_verlet[MAX_MONOMERS][MAX_LJWW],upper_verlet[MAX_MONOMERS];
extern int    verlet_update_counter;

extern double *xtrans,*ytrans;
#ifdef VH_Flag
extern double *xtrans_box,*ytrans_box;
#endif
```



```c
extern double *WW1,*WW2;
extern double **WW1_in,**WW2_in;
extern int WW_max;

int  read_seed();
void write_seed(int iseed);
void read_parameters();
void read_config_sysi();
void setup_pointers();
void addp2cell(int ip, int ix, int iy, int iz);
void start_from_ortho();
void start_from_shear_ortho();
void start_from_shear_nn(int idir_start);
void start_from_ground_nn();
void start_from_shear_nnn(int idir_start);
void start_from_ground_nnn();
void start_from_config(int ifile);
void read_from_config(double *pmcs0);
void initialize_LJ_table();

void initialize_energy(double *egy);
void initialize_ang_energies(double *copo);
double se_mono(int ip, double x, double y, double z,
             double ljcut_quad, double lj_scal, double *L);
double calculate_energy(int ip,double x,double y,double z);
void calc_all_lj(double *e_field);
void update_energies(int ip);

#ifndef VH_Flag
void svm_initialize_energy(double new_ssx, double new_ssy, double new_cosa2,
                           double *egy);
void svm_initialize_ang_energies(double *copo);
#else
void svm_hom_init_energy(double trysyx, double trysyy, double cosa2,
                         double *newenrg);
#endif


void allocate_msr_memory();
void initialize_histo();
void read_histo(int *isamp);
void add_to_histo();
void print_histoxy(int isamp);
void print_histoz(int isamp);
void print_histoang(int isamp);
void print_histodst(int isamp);
void print_historpvf(int isamp);
double msr_energy();
void prepare_msr_files();
void measurement(double accpt,
                 double vaccptx,double vaccpty, double vaccptz,
                 double shracpt,double ptaccpt);

double bond_laengen_potential(double dist);
double save_atan2(double y,double x);

void verlet_update();

void shear_period(double *xb,double *yb,double x, double y,
                  double SSX,double SHY,double alpha);
```

MC.c 1/30

```c
/* ======================================== */
/* Dieses File enthaelt das Hauptprogramm ! */
/* ======================================== */

/* Zentrale Schalttafel laden */
/* -------------------------- */

#include "switchboard.h"

/* Rechnerspezifische Includes */
/* --------------------------- */

/* Allgemeine Includes */
/* ------------------- */

#include <stdlib.h>
#include <malloc.h>
#include <sys/types.h>
#include <sys/times.h>
#include <unistd.h>
#include <time.h>
#include <stdio.h>
#include <math.h>

#include "r250.h"
#include "matrix.h"
#include "MC.h"

#ifdef SIGHAND
#include <signal.h>
#endif

/* Globale Deklarationen */
/* --------------------- */

double *WW1,*WW2,WW1_fix[WW_triangle],WW2_fix[WW_triangle];
double **WW1_in,**WW2_in;
int WW_max;

struct part     ****cell,*list;
int     npm,lpm,number_of_cells_x,number_of_cells_y ,number_of_cells_z ;
int     ncx_alloc,ncy_alloc,ncz_alloc;
int     dim,ntot,nsav,nsavb;
int     *jpx,*jmx,*jpy,*jmy,*jpz,*jmz ;       /* naechste Nachbarzellen */
int     *xcell,*ycell,*zcell;
double  mcsmax,mcs_aequi,mcs;
double  LJ[LJ_grid+1] ;       /* LJ-Potential fuer Monomer-Monomer-WW   */
double  LJ_hh[LJ_hh_grid+1] ; /* LJ Potential fuer Kopf-Kopf-WW         */
double  LJ_hm[LJ_hm_grid+1] ; /* LJ-Potential fuer Kopf-Monomer-WW      */
double  ljmin,ljcut,lj0,u0 ;              /* Parameter fuer Mono-Mono-WW */
double  ljmin_hh,ljcut_hh,lj0_hh,u0_hh ; /* Parameter fuer Kopf-Kopf-WW */
double  ljmin_hm,ljcut_hm,lj0_hm,u0_hm ; /* Parameter fuer Kopf-Mono-WW */
double  ljcut_quadrat,ljcut_hh_quadrat,ljcut_hm_quadrat,ljcut_max;
double  lj_scala,lj_hh_scala,lj_hm_scala;
double   *pLJ[3],ljsc[3],ljcq[3],lju0[3],ljmq[3];
char     lj_type[MAX_MONOMERS][MAX_MONOMERS];
double  engy[MAX_MONOMERS],ebnd[2];
int     nbnd[2];
double  *xmm,*ymm,*zmm,*xbox,*ybox;
double  SSX,inv_SSX,SSY,SHY,inv_SHY,alpha,alpha_io,ctg_alpha,cosa2,system_size_z
,
        CSX,CSY,cell_size_z ;
double  hvar,volvar,alphvar,alphvar_io,lj_safety,temp,w0,pressure;
```

MC.c 2/30

```c
double  bond_length,bond_limit_quadrat,bond_min,bond_max ;
double  sprng,bond_min_quadrat,bond_max_quadrat;
double  cospr,costry1,costry2,costry3;
double  *cospot,*energy,*newcpot,*newenrg;
double  *xn2t,*yn2t,*zn2t,*xendt,*yendt,*zendt;
double *xax,*yax,*zax,*ncth,*nphi;
char    **swtch,heads;
unsigned long *histoxy,*histoz,*histothe,*histophi,*histodst,*historpvf;

int    cfg_save_flag,cfg_save_counter;

double x0ver[MAX_MONOMERS],y0ver[MAX_MONOMERS],z0ver[MAX_MONOMERS];
double d_diff_max,d_diff_max2,rad_verlet,skin_verlet,skin;
int    tab_verlet[MAX_MONOMERS][MAX_LJWW],upper_verlet[MAX_MONOMERS];
int    verlet_update_counter;

double *xtrans,*ytrans;   /* trial coordinates for volume moves */
#ifdef VH_Flag
double *xtrans_box,*ytrans_box; /* trial box coords. for VH-moves */
#endif

FILE *Pro;
int info,WEITER,XCPU;

#ifdef kollektiv
 double **akz_engy;
#endif


#ifdef SIGHAND
/* ---------------------------------------------------------------- */
/* Ignoriere Signal, setze aber glob. Variable XCPU:=sig            */
/*                                                                  */
/* Der Befehl "signal(sig, handler)" im Hauptprogramm bewirkt,      */
/* dass bei einem sig-Signal ans laufenden Programm die Funktion    */
/* "handler()" aufgerufen wird - letztlich XCPU:=sig gesetzt wird.  */
/* Hierzu ist keine staendige Abfrage noetig, nur der Zustand von   */
/* XCPU muss natuerlich regelmaessig ueberprueft werden.            */

void handler(int sig)
{
  signal(sig, SIG_IGN);
  XCPU = sig;
}
#endif


/* ---------------------------------------------------------------- */
/* Speichert die Konfiguration und Boxgeometrie in Binaerformat     */

void bin_save_cfg()
{
        FILE *file_1;
        char fname[30];
        double cfg_save_dummy[40];
        size_t n_data;
        int i;

        for (i=0; i<40; i++) cfg_save_dummy[i]=0.0;

        cfg_save_dummy[0]  = (double) dim;
        cfg_save_dummy[1]  = (double) npm;
        cfg_save_dummy[2]  = (double) lpm;
```

```
        cfg_save_dummy[3]  = mcsmax;
        cfg_save_dummy[4]  = temp;
        cfg_save_dummy[5]  = pressure;
        cfg_save_dummy[6]  = SSX;
        cfg_save_dummy[7]  = SHY;
        cfg_save_dummy[8]  = alpha_io;
        cfg_save_dummy[9]  = bond_min;
        cfg_save_dummy[10] = bond_max;
        cfg_save_dummy[11] = hvar;
        cfg_save_dummy[12] = volvar;
        cfg_save_dummy[13] = alphvar_io;
        cfg_save_dummy[14] = u0;
        cfg_save_dummy[15] = sprng;
        cfg_save_dummy[16] = cospr;
        cfg_save_dummy[17] = ljmin;
        cfg_save_dummy[18] = ljcut;
        cfg_save_dummy[19] = u0_hh;
        cfg_save_dummy[20] = ljmin_hh;
        cfg_save_dummy[21] = ljcut_hh;
        cfg_save_dummy[22] = u0_hm;
        cfg_save_dummy[23] = ljmin_hm;
        cfg_save_dummy[24] = ljcut_hm;
        cfg_save_dummy[25] = (double) heads;
        cfg_save_dummy[26] = mcs;
        cfg_save_dummy[27] = mcs_aequi;

        cfg_save_counter++;

        sprintf(fname,"%s_%04d.bcnf",swtch[1],cfg_save_counter);
        file_1=fopen(fname,"w");

 n_data = (size_t) 40;
        fwrite(cfg_save_dummy,sizeof(double),n_data,file_1);

 n_data = (size_t) ntot;
        fwrite(xmm,sizeof(double),n_data,file_1);
        fwrite(ymm,sizeof(double),n_data,file_1);
        fwrite(zmm,sizeof(double),n_data,file_1);

        fclose(file_1);
}

/* ---------------------------------------------------------------- */
/* Speichert alle Parameter und 3dim Orte aller Monomere            */

void save_configuration(int j)
{
        int  ii;
        FILE *file_1;
        char fname[20];

        /* j=0 entspr. Startkonfiguration */
        /* j=1 entspr. Endkonfiguration   */

        sprintf(fname,"%s_%02d.cnf",swtch[1],j);
        file_1=fopen(fname,"w");

    fprintf(file_1,"CONFIGURATION %s CREATED BY MC.exe\n",fname);
    fprintf(file_1,"--------------------------------------------\n");
    fprintf(file_1,"dim       npm       lpm       mcsmax    temp      pressure\n
");
    fprintf(file_1,"%-9d %-9d %-9d %-9ld %-9.6f %-9.6f\n"
                   ,dim,npm,lpm,(long)mcsmax,temp,pressure);
```



```
 alpha_io   = 180.0*alpha/M_PI;
 alphvar_io = 180.0*alphvar/M_PI;

    fprintf(file_1,"SSX       SHY       alpha     bnd_min  bnd_max  hvar    volv
ar  alphvar\n");
    fprintf(file_1,"%.13f %.13f %.13f %-8.3f %-8.3f %-7.5f %-7.5f %-7.5f\n"
                  ,SSX,SHY,alpha_io,bond_min,bond_max,hvar,volvar,alphvar_io);

    fprintf(file_1,"u0 (LJ)   sprng     cospr     ljmin     ljcut\n");
    fprintf(file_1,"%-9.3f %-9.3f %-9.3f %-9.3f %-9.3f\n"
            ,u0,sprng,cospr,ljmin,ljcut);
    fprintf(file_1,"u0_hh    ljmin_hh     ljcut_hh   << Kopf-Kopf-WW\n");
    fprintf(file_1,"%-9.3f %-9.3f %-9.3f\n",u0_hh,ljmin_hh,ljcut_hh);
    fprintf(file_1,"u0_hm    ljmin_hm     ljcut_hm   << Kopf-Monomer-WW\n");
    fprintf(file_1,"%-9.3f %-9.3f %-9.3f\n",u0_hm,ljmin_hm,ljcut_hm);

    fprintf(file_1,"%c : headgroups fixed (0) or movable (1) \n",heads);
    fprintf(file_1,"%14.6f : mcs at save time \n",mcs);
    fprintf(file_1,"%14.6f : start measurements at \n",mcs_aequi);

    for (ii=0; ii<ntot; ii++)
      {
        fprintf(file_1,"%.13f %.13f %.13f %.13f %.13f\n",
        xmm[ii],ymm[ii],zmm[ii],xbox[ii],ybox[ii]);
      }

    fclose(file_1);

    if (cfg_save_flag) bin_save_cfg();
}

/* ---------------------------------------------------------------- */
/* neue Initialisierung der naechsten und uebernaechsten Nachbar-    */
/* zellen in  x- und y-Richtung (vergleiche allocate_mermory)        */

void redefine_neighbor_cells()
{
     int i1;

        for (i1=0; i1<number_of_cells_x; i1++)
           { jpx[i1] = (i1+1+3*number_of_cells_x)%number_of_cells_x;
             jmx[i1] = (i1-1+3*number_of_cells_x)%number_of_cells_x; }

        for (i1=0; i1<number_of_cells_y; i1++)
           { jpy[i1] = (i1+1+3*number_of_cells_y)%number_of_cells_y;
             jmy[i1] = (i1-1+3*number_of_cells_y)%number_of_cells_y; }

        for (i1=0; i1<number_of_cells_z; i1++)
           { jpz[i1] = (i1+1+3*number_of_cells_z)%number_of_cells_z;
             jmz[i1] = (i1-1+3*number_of_cells_z)%number_of_cells_z; }
}

/* ---------------------------------------------------------------- */

void allocate_memory()
{
        int i1,i2,i3;
        size_t s_ntot,s_ntot2,s_x,s_y,s_z;

        s_ntot  = (size_t) ntot;
        s_ntot2 = (size_t) (ntot+2);
```

MC.c 5/30

```
        /* Allocierung fuer die Wechselwirkungstafeln                */
        /* bei dynamischer Allocierung passieren seltsame Dinge !    */
        WW1 = &WW1_fix[0];
        WW2 = &WW2_fix[0];

        /* Pointer in die Abschnitte der Wechselwirkungstafeln       */
        WW1_in  = (double **) calloc(s_ntot,sizeof(double *));
        WW2_in  = (double **) calloc(s_ntot,sizeof(double *));

        cospot = (double *) calloc(s_ntot2,sizeof(double));  /* Potentiale  */
        energy = (double *) calloc(s_ntot,sizeof(double));
       newcpot = (double *) calloc(s_ntot2,sizeof(double));
       newenrg = (double *) calloc(s_ntot,sizeof(double));
        xmm    = (double *) calloc(s_ntot,sizeof(double));  /* Koordinaten */
        ymm    = (double *) calloc(s_ntot,sizeof(double));
        zmm    = (double *) calloc(s_ntot,sizeof(double));
        xbox   = (double *) calloc(s_ntot,sizeof(double));
        ybox   = (double *) calloc(s_ntot,sizeof(double));
        xtrans = (double *) calloc(s_ntot,sizeof(double));
        ytrans = (double *) calloc(s_ntot,sizeof(double));
#ifdef VH_Flag
    xtrans_box = (double *) calloc(s_ntot,sizeof(double));
    ytrans_box = (double *) calloc(s_ntot,sizeof(double));
    if (xtrans_box == NULL)    printf("ALLOCATION ERROR \n");
    if (ytrans_box == NULL)    printf("ALLOCATION ERROR \n");
#endif

        xcell  = (int *) calloc(s_ntot,sizeof(int));     /* Zellnummern */
        ycell  = (int *) calloc(s_ntot,sizeof(int));     /* d. Monomere */
        zcell  = (int *) calloc(s_ntot,sizeof(int));
        list = (struct part *) calloc(s_ntot,sizeof(struct part));

     /* allocate thrice as much memory for j... and cell as needed
        for the present systemsizes  in xyz-direction             */

     /* um bei Volumen-Moves auf der sicheren Seite zu sein !     */

     /* (seq.) Zellnummern der naechsten und uebernaechsten Zellen
        in den verschiedenen Raumrichtungen                       */

        ncx_alloc = 3*number_of_cells_x;
        ncy_alloc = 3*number_of_cells_y;
        ncz_alloc = 3*number_of_cells_z;

        s_ntot = (size_t) (3);
        s_x = (size_t) (ncx_alloc);
        s_y = (size_t) (ncy_alloc);
        s_z = (size_t) (ncz_alloc);

        jmx     = (int *) calloc(s_x,sizeof(int));
        jpx     = (int *) calloc(s_x,sizeof(int));
        jmy     = (int *) calloc(s_y,sizeof(int));
        jpy     = (int *) calloc(s_y,sizeof(int));
        jmz     = (int *) calloc(s_z,sizeof(int));
        jpz     = (int *) calloc(s_z,sizeof(int));

     /* matrix_alloc in  < matrix.c >                            */
     /* belegt Platz fuer <3>-dim array mit Laengen number_of_cell_...
        mit Elementen der Groesse sizeof(..)
        und zusaetzlich Pointern auf Zeilen und Spalten
        ( werden bei Zeigerdarstellung eines Arrays benoetigt )   */

        cell = (struct part ****)
```

MC.c 6/30

```
                matrix_alloc(s_ntot,sizeof(struct part),s_x,s_y,s_z);

#ifdef kollektiv
        /* Speicherung der einzelnen LJ-Beitraege des akzeptierten Bonds   */
        /* fuer die versch. Segmente bei kollektiven Moves (PT)            */

        s_ntot = (size_t) (2);
        s_x    = (size_t) lpm;
        s_y    = (size_t) ntot;

        akz_engy = (double **)
                   matrix_alloc(s_ntot,sizeof(double),s_x,s_y);

        if (akz_engy == NULL) printf("ALLOCATION ERROR \n");
#endif

        if (WW1 == NULL)    printf("ALLOCATION ERROR \n");
        if (WW2 == NULL)    printf("ALLOCATION ERROR \n");
        if (WW1_in == NULL) printf("ALLOCATION ERROR \n");
        if (WW2_in == NULL) printf("ALLOCATION ERROR \n");

        if (energy == NULL)  printf("ALLOCATION ERROR \n");
        if (cospot == NULL)  printf("ALLOCATION ERROR \n");
        if (newcpot == NULL) printf("ALLOCATION ERROR \n");
        if (newenrg == NULL) printf("ALLOCATION ERROR \n");
        if (xmm == NULL)     printf("ALLOCATION ERROR \n");
        if (ymm == NULL)     printf("ALLOCATION ERROR \n");
        if (zmm == NULL)     printf("ALLOCATION ERROR \n");
        if (xbox == NULL)    printf("ALLOCATION ERROR \n");
        if (ybox == NULL)    printf("ALLOCATION ERROR \n");
        if (xtrans == NULL)    printf("ALLOCATION ERROR \n");
        if (ytrans == NULL)    printf("ALLOCATION ERROR \n");

        if (xcell == NULL)   printf("ALLOCATION ERROR \n");
        if (ycell == NULL)   printf("ALLOCATION ERROR \n");
        if (zcell == NULL)   printf("ALLOCATION ERROR \n");
        if (jmx == NULL)     printf("ALLOCATION ERROR \n");
        if (jpx == NULL)     printf("ALLOCATION ERROR \n");
        if (jmy == NULL)     printf("ALLOCATION ERROR \n");
        if (jpy == NULL)     printf("ALLOCATION ERROR \n");
        if (jmz == NULL)     printf("ALLOCATION ERROR \n");
        if (jpz == NULL)     printf("ALLOCATION ERROR \n");
        if (list == NULL)    printf("ALLOCATION ERROR \n");
        fflush(stdout);

     /* da gelegentlich auf cospot[ip+-1] zugegriffen wird, koennte dies */
     /* fuer ip=0 oder ip=ntot zu "wilden" Werten fuehren                */

        cospot[0] = 0.0;  cospot[ntot+1] = 0.0;  cospot++;
        newcpot[0] = 0.0; newcpot[ntot+1] = 0.0; newcpot++;

     /* initialize pointer arrays and energy */
     /* fuer Einsortieren mit "addp2cell"    */

        for (i1=0;i1<ntot;i1++) {
                list[i1].pn=i1;
                list[i1].next=NULL;
                energy[i1] = 0.;
        }

        for (i1=0; i1<ncx_alloc; i1++)
           for (i2=0; i2<ncy_alloc; i2++)
              for (i3=0; i3<ncz_alloc; i3++)
```

```
                { cell[i1][i2][i3]=NULL; }

        redefine_neighbor_cells();
}

/* ---------------------------------------------------------------- */
/* Hinzufuegen der Monomere zur entspr. Zelle                       */

/* Vorbesetzt ist :  list[i].pn   = i        ( in allocate_memory )
                     list[i].next = NULL
                    cell[x][y][z] = NULL                             */

/*  list[i] ist eine Variable vom Typ struct_part,
    cell[x][y][z] ist ein Pointer auf ein struct_part !              */

void addp2cell(ip,ix,iy,iz)
int ip,ix,iy,iz;
{
        struct part *tmp;

     /* Zelle noch leer, erstes Monomer wird eingetragen            */

        if (cell[ix][iy][iz]==NULL)
          { cell[ix][iy][iz] = &list[ip]; }

     /* Zelle schon einmal belegt, Monomer wird angefuegt
        ( eigentlich wird die Zelle ans Monomer gehaengt )          */

     /* tmp zeigt auf den bish. Listenanfang von cell[ix][iy][iz]
        dieser wird auf das ip.te Monomer verlegt,
        dessen Anschluss .next auf den alten Listenanfang zeigt     */

        else {
                tmp = cell[ix][iy][iz];
                cell[ix][iy][iz] = &list[ip];
                cell[ix][iy][iz]->next = tmp;
             }
}

/* ---------------------------------------------------------------- */
/* >> ungenutzt <<                                                  */
/* gibt fuer ganze verkettete Liste jeweils die Monomer-Nummern aus */

void print_cell(CL)
struct part *CL;
{
        for(;CL!=NULL;CL=CL->next)
                printf("%i ",CL->pn);
}

/* ---------------------------------------------------------------- */
/* Bei Monomer-Move hat Monomer alte Zelle verlassen, muss daher    */
/* aus alter entfernt und in neuen eingeordnet werden               */
/* ix,iy,iz : Indizes der neuen Zelle des Monomers ip               */

void update_pointers(ip,ix,iy,iz)
int ip,ix,iy,iz;
{
        struct part *CL;
        int ixold,iyold,izold,iprev;

        ixold = xcell[ip];                /* Indizes der alten Zelle */
        iyold = ycell[ip];
```



```
        izold = zcell[ip];

        /* Monomer war in alter Zelle am Listen-Anfang (einfacher)    */
        /* tritt beim hin-und-her-wechseln wohl oefter auf            */
        /* Alter Zelle wird Anfang abgeschnitten,                     */
        /* Monomer wird Anfang seiner neuen Zelle                     */
        if (cell[ixold][iyold][izold]->pn == ip){
                /* Listen-Anfang auf bisheriges 2. Element setzen     */
                cell[ixold][iyold][izold] = cell[ixold][iyold][izold]->next;
                /* Monomer-Listen-Next auf bisheriges 1. Element der  */
                /* neuen Zelle setzen                                 */
                list[ip].next = cell[ix][iy][iz];
                /* Zell-Listen-Anfang auf ip.tes Monomer in Monomer-  */
                /* Liste 'list' setzen                                */
                cell[ix][iy][iz] = &list[ip];
                xcell[ip] = ix; /* Neue Indizes verbuchen             */
                ycell[ip] = iy;
                zcell[ip] = iz;
                return;
        }

        /* Monomer war nicht am Listen-Anfang seiner alten Zelle      */
        /* Monomer wird in alter Liste jetzt uebersprungen und        */
        /* wird zum Anfang der Liste der neuen Zelle                  */

        /* setze CL auf 2. Element der Zell-Liste                     */
        CL = cell[ixold][iyold][izold]->next;
        /* iprev := Nummer des vorhergehenden Monomers in der Zelle   */
        iprev = cell[ixold][iyold][izold]->pn;

        /* Schleife bis zum Zell-Listen-Ende                          */
        for (;CL!=NULL;CL=CL->next){
                if (CL->pn == ip){         /* wenn Monomer gefunden : */

                        /* vorheriges Monomer zeige auf uebernaechst. */
                        list[iprev].next = CL->next;
                        /* Monomer zeige auf bish. Anfang neuer Zelle */
                        list[ip].next = cell[ix][iy][iz];
                        /* neuer Anfang neuer Zelle sei das Monomer   */
                        cell[ix][iy][iz] = &list[ip];
                        xcell[ip] = ix; /* Neue Indizes verbuchen     */
                        ycell[ip] = iy;
                        zcell[ip] = iz;
                        return;
                }
                iprev = CL->pn;
        }

        /* sollte "normal" nie erreicht werden, denn dann war Mono-   */
        /* mer ip gar nicht in Zelle (xcell[ip],ycell[ip],zcell[ip])  */

        printf("ERROR in update_pointers\n");
        fflush(stdout);
}

/* ---------------------------------------------------------------- */
/* verbuchen der geaenderten WW-Energien (Bond und LJ) auch bei den */
/* anderen Monomeren !                                              */

void update_energies(int ip)
{ int i0,i1,i2,i3;
  double dummy;
```

MC.c 9/30

```
  i0 = ip%lpm;

  /* unterer Bond-Nachbar                                          */
  if (i0!=0)
    { energy[ip-1]   += engy[ip-1] - WW1_in[ip-1][0];
      WW1_in[ip-1][0] = engy[ip-1];
    }
  /* oberer Bond-Nachbar                                           */
  if (i0!=(lpm-1))
    { energy[ip+1] += engy[ip+1] - WW1_in[ip][0];
      WW1_in[ip][0] = engy[ip+1];
    }

  /* echte LJ-Partner (ohne ip selbst und seine Bond-Nachbarn)      */
  /* Verlet-Tabelle hat zuerst "hoehere" dann "niedrigere" Partner  */

  for (i1=1; i1<=upper_verlet[ip]; i1++)
    { i2=tab_verlet[ip][i1];
      dummy               = engy[i2] - WW1_in[ip][i2-ip-1];
      WW1_in[ip][i2-ip-1] = engy[i2];
      energy[i2] += dummy; }

  for (i3=upper_verlet[ip]+1; i3<=tab_verlet[ip][0]; i3++)
    { i2=tab_verlet[ip][i3];
      dummy               = engy[i2] - WW1_in[i2][ip-i2-1];
      WW1_in[i2][ip-i2-1] = engy[i2];
      energy[i2] += dummy; }

}

/* ---------------------------------------------------------------- */
/* Entscheidet entgueltig ueber Akzeptanz eines Monomer-Moves       */

/* Aufruf mit 'metropo_update(hitm,xtry,ytry,ztry)', d.h. mit       */
/* hitm : Index des Monomers, r_try : neue Versuchskoordinaten      */

/* return-Wert : 0 --> Monomer-Move akzeptiert, 1 --> abgelehnt     */

int metropo_update(ip,x,y,z)
int ip;
double x,y,z;
{
    int ix,iy,iz,i1;
    double ediff,etry,xb,yb,diff;

        /* Periodische Koordinaten zu r_try                         */
        shear_period(&xb,&yb,x,y,inv_SSX,inv_SHY,ctg_alpha);

        ix = itrunc(xb*number_of_cells_x); /* Zellindex neuer Posit. */
        iy = itrunc(yb*number_of_cells_y);
        iz = itrunc(z/cell_size_z);

        /* steht in 'MC_e.c'                                        */
        /* wertet LJ-WW mit Monomeren in der Verlet-Tabelle aus und  */
        /* summiert die Energien                                    */
        /*  etry     : sum of new  bond and LJ-potentials           */

        etry  = calculate_energy(ip,xb,yb,z);

        /* Ergaenzung mit Bond-Laengen-WWs                          */
        for (i1=0; i1<nsav; i1++)
          { engy[nbnd[i1]] = ebnd[i1];  etry = etry + ebnd[i1]; }
```

MC.c 10/30

```
        /* Vervollstaendigung der neuen Energie durch Bond-Winkel-WW */
        /* und Vergleich mit bisherigen Energien                    */
        /* Achtung : hier wird nur Energiedifferenz fuer ip betracht.*/
        /*           da nicht ueber alle Monomers summiert wurde,   */
        /*           muss auch nicht durch 2 dividiert werden.      */
        ediff = etry - energy[ip]
               + costry1      + costry2    + costry3
               - cospot[ip-1] - cospot[ip] - cospot[ip+1];

        /* Entscheidung ueber Akzeptanz nach Metropolis-Algorithmus  */
        /* Schritt wird immer akzeptiert, wenn Energie-Absenkung,    */
        /* bei Energieanhebung nur, falls Zufall < Boltzmannfaktor   */
        /* d.h. je groesser ediff, desto kleiner exp(-ediff/temp),   */
        /* desto unwahrscheinlicher, dass 'Zufall' noch kleiner ist. */

        /* sonst UNDERFLOW; Energiezuwachs zu gross, immer abgelehnt */
        if (ediff/temp > 70.) return(1);

        if (ediff > 0.)
          if (double_r250() > exp(-ediff/temp)) return(1);

        /* ........... Monomer-Move wurde akzeptiert .............  */

        /* Wenn Monomer alte Zelle verlassen hat --> update_pointers */
        if ((ix!=xcell[ip])||(iy!=ycell[ip])||(iz!=zcell[ip]))
                update_pointers(ip,ix,iy,iz);

        /* Neue Energie incl. benachbarter Bond-Winkel verbuchen     */
        energy[ip] = etry;
        cospot[ip] = costry1;
        cospot[ip-1]= costry2;
        cospot[ip+1]= costry3;

        /* Neue Energien auch bei WW-Partnern verbuchen              */
        update_energies(ip);

        xmm[ip]  = x;      /* Neue Koordinaten             verbuchen */
        ymm[ip]  = y;
        zmm[ip]  = z;
        xbox[ip] = xb;     /* Neue periodsche Koordinaten verbuchen */
        ybox[ip] = yb;

        /* evtl. Verlet-Tabelle updaten (nur komplett, denn andere   */
        /* Monomere wurden vermutl. auch schon bewegt !)             */
        diff = (x-x0ver[ip])*(x-x0ver[ip]) +
               (y-y0ver[ip])*(y-y0ver[ip]) +
               (z-z0ver[ip])*(z-z0ver[ip]);
        if (diff>d_diff_max2)
          { if (diff>d_diff_max)
                { d_diff_max2 = d_diff_max;
                  d_diff_max  = diff;
                }
            else
                d_diff_max2= diff;
          /* Verlet-Tabelle wird upgedatet wenn die beiden groessten  */
          /* Monomer-Drifts seit letztem Update zusammen die Schicht- */
          /* dicke  skin_verlet  ueberschreiten.                      */
            if ((sqrt(d_diff_max2)+sqrt(d_diff_max))>=skin_verlet)
              verlet_update();
          }

        return(0); /* Mono-Move akzeptiert, ausgefuehrt u. verbucht  */
}
```

```c
/* ---------------------------------------------------------------- */
/* Versucht ein einzelnes Monomer zu bewegen, Erfolg -> 0, sonst 1   */

int move()
{
    int hitm,hit0;
    double  xdel1,ydel1,zdel1,xdel2,ydel2,zdel2,xdel3,ydel3,zdel3;
    double  xdel4,ydel4,zdel4,xtry,ytry,ztry,ldist,rdist,dist3,dist4;
    double  rnd;

     /* choose monomer                                              */
        hitm = itrunc(double_r250()*ntot);
        hit0 = hitm % lpm;  /* Modulo--> hit0 gibt Position in Kette */

    /* move the chosen monomer randomly                             */
    /* headmonomers  fixed at z=1.                                  */
    /* Zufallsbewegung in z-Richtung nur falls kein Kopf-Monomer    */
    /* falls Kopf-Monomer fixiert, gleich return(1)                 */

        if (hit0 == 0)
          { if (heads == '0') return (1); /* Flag f. fixierte Koepfe  */
            ztry = 1.; }
        else
          { rnd = 2.*double_r250()-1.;   /* Zufallszahl zw. -1 und +1 */
            ztry = zmm[hitm] + rnd*hvar; }

        rnd = 2.*double_r250()-1.;   /* Zufallsbewegung in x-Richtung */
        xtry = xmm[hitm] + rnd*hvar;

        if (dim == 3)
          { rnd = 2.*double_r250()-1.; /* in y-Richt. nur falls 3-dim */
            ytry = ymm[hitm] + rnd*hvar; }
        else ytry = ymm[hitm];

    /* return if z < 1. (i.e. monomer moving under the surface)      */
        if (ztry < 1.) return(1);

    /* return if bondlengths to nearest neighbours are out of range   */
    /* calculate bond interaction energy                              */
        nsav = 0;
        if (hit0 < lpm-1) {            /* Nur wenn kein End-Monomer : */
                xdel1 = xmm[hitm+1]-xtry;  /* Abstand zum oberen Mono */
                ydel1 = ymm[hitm+1]-ytry;
                zdel1 = zmm[hitm+1]-ztry;
                rdist = xdel1*xdel1+ydel1*ydel1+zdel1*zdel1;
                if (rdist <= bond_min_quadrat) return(1);
                if (rdist >= bond_max_quadrat) return(1);
                rdist = sqrt(rdist);
                nbnd[nsav] = hitm+1;
                ebnd[nsav] = bond_laengen_potential(rdist);
                nsav = nsav + 1;
        }

        if (hit0 > 0) {                /* Nur wenn kein Kopf-Monomer : */
                xdel2 = -xmm[hitm-1]+xtry; /* Abstand zum unteren Mono */
                ydel2 = -ymm[hitm-1]+ytry;
                zdel2 = -zmm[hitm-1]+ztry;
                ldist = xdel2*xdel2+ydel2*ydel2+zdel2*zdel2;
                if (ldist <= bond_min_quadrat) return(1);
                if (ldist >= bond_max_quadrat) return(1);
                ldist = sqrt(ldist);
                nbnd[nsav] = hitm-1;
```



```c
                ebnd[nsav] = bond_laengen_potential(ldist);
                nsav = nsav + 1;
        }

        /* calculate bondangle energies                                */
        /* Achtung : Die Bewegung eines einzigen Monomers aendert bis  */
        /*           zu drei umliegende Bondwinkel !                   */
         costry1= 0.;
         costry2= 0.;
         costry3= 0.;
         if (nsav == 2)                            /* Innerer Winkel  */
           { costry1 = cospr*(1.0-
                     (xdel1*xdel2 +ydel1*ydel2 +zdel1*zdel2)/(ldist*rdist)); }

         if (hit0 > 1) {
              xdel3 = xmm[hitm-1]-xmm[hitm-2];      /* unterer Winkel  */
              ydel3 = ymm[hitm-1]-ymm[hitm-2];
              zdel3 = zmm[hitm-1]-zmm[hitm-2];
              dist3 = xdel3*xdel3 + ydel3*ydel3 + zdel3*zdel3;
              dist3 = sqrt(dist3);
              costry2 = cospr*(1.0-
                        (xdel2*xdel3 + ydel2*ydel3 + zdel2*zdel3)/(dist3*ldist))
;
         }
         if (hit0 < lpm - 2) {
              xdel4 = xmm[hitm+2]-xmm[hitm+1];      /* oberer Winkel   */
              ydel4 = ymm[hitm+2]-ymm[hitm+1];
              zdel4 = zmm[hitm+2]-zmm[hitm+1];
              dist4 = xdel4*xdel4 + ydel4*ydel4 + zdel4*zdel4;
              dist4 = sqrt(dist4);
              costry3 = cospr*(1.0-
                        (xdel1*xdel4 + ydel1*ydel4 + zdel1*zdel4)/(dist4*rdist))
;
         }

         /* bisher wurden nur die BW- und BL-Energien beruecksichtigt  */
        if (metropo_update(hitm,xtry,ytry,ztry) == 1) return(1);

        return(0);
}

#ifndef NVT_Flag
#ifndef VH_Flag

/* ---------------------------------------------------------------- */
/* (NpT)-Ensemble : Aenderung der Box-Grundflaeche in x-Richtung    */
/*                                                                  */
/*                  Parinello-Rahman-System                         */
/*                                                                  */
/*       ____________________......                                 */
/*      /                   /    ;   relativ Koordinaten xbox,ybox  */
/*     /                   /    ;    unveraendert                  */
/*    /                   /    ;     ymm,SHY,SSY,alpha ebenfalls   */
/*   /                   /    ;      unveraendert                  */
/*  /___________________/......                                     */
/*                                                                  */
/* im Gegensatz zu proportionale Trafo  x -> lamda*x                */
/*                                                                  */
/*       ____________________.........                              */
/*      /  ..               /      ..  wobei sich auch SSY und der  */
/*     / ..                /     ..    Winkel alpha aendern        */
/*    /..                 /    ..                                  */
/*   /.                  /    .                                    */
```

```
/*  /_________________/.....                                          */
/*                                                                     */
/*                                                                     */
/* ------------------------------------------------------------------ */
/* Transformation fuer jedes Monomer bei einem volume_move_ssx der    */
/* gescherten Box, wenn Relativkoordinaten erhalten und ausschl.      */
/* SSX geaendert wird.  Berechnung erfolgt aus xmm,ymm damit tats.    */
/* Teilchenbewegung nachvollziehbar, mit xbox,ybox wuerden die        */
/* Teilchen in die Box gezwungen.                                     */

void shear_x_trans(lambda,xc,yc)
double lambda,*xc,*yc;
{ int i1;
  double fy;

  fy = (1.0-lambda)*ctg_alpha;

  /* Transformation wird mit nicht-period. Koordinaten berechnet !    */
  for (i1=0; i1<ntot; i1++)
    { xtrans[i1] = lambda*xc[i1] + fy*yc[i1];
      ytrans[i1] = yc[i1];
    }
}

int volume_move_ssx()
{
    int    i1,ncx,nxcell;
    double trysyx,bx,enedif,cosdif,ediff,diff;
    double *tmpe,**tmpp;

    /* Neue Systemlaenge in x-Richtung auswuerfeln                    */
    trysyx = SSX + volvar*(2.*double_r250() - 1.);
    bx     = trysyx/SSX;

    /* sicherstellen, dass Verletkugel noch in neue Box passt !       */
    /* impiziert zusaetzliches Potential, ist aber zur Auswertung     */
    /* der periodischen Randbedingungen noetig, sonst WW mit mehr     */
    /* als einem Bild eines Teilchens moeglich.                       */
    ncx    = itrunc(trysyx*sin(alpha)/(ljcut+skin_verlet));
    if(ncx < 2) return(1);

    /* Transformation der kartesischen Absolutkoordinaten             */
    shear_x_trans(bx,xmm,ymm);

    /* Werte Bond- und LJ-Wechselwirkungen in deformierter Box  aus   */
    svm_initialize_energy(trysyx,SSY,cosa2,newenrg);     /* <MC_e.c> */

    /* Werte Bond-Winkelpotentiale in deformierter Box aus            */
    svm_initialize_ang_energies(newcpot);                /* <MC_e.c> */

    enedif= 0.0;
    cosdif= 0.0;
    for (i1=0; i1<ntot; i1++)
      { enedif = enedif - energy[i1] + newenrg[i1];
        cosdif = cosdif - cospot[i1] + newcpot[i1];
      }

    /* Auswerten der Energie-Aenderung durch den Volumen-Move         */
    /* im Ggs. zu den Bond-Winkel-Potentialen wurden die 2-Monos-WW   */
    /* (Bond und LJ) doppelt gezaehlt, da beiden Monos zugeordnet     */
    /* 'enedif' muss daher noch halbiert werden !                     */

    ediff = /* Bond-Laengen-, LJ- und Bond-Winkel-Energien            */
```



```
            0.5*enedif + cosdif +
            /* Volumenaenderungsarbeit bei geg. Druck : dW = p * dV  */
            /* bzw.          Flaechenaenderungsarbeit : dW = P * dA  */
            /* Flaeche  A = SSX * SSY * sin(alpha)  =  SSX * SHY     */
            pressure*(trysyx - SSX)*SHY -
            /* vergleiche [Haas, (A.5), S. 96]                       */
            /* -N * k_B * T * ln( A_neu / A_alt )                    */
            /* eigentlich nur aus Trafo des Differentials d^{3N}r    */
            ntot*log(bx)*temp;


    /* Metropolis-Entscheidung ueber Annahme des Volumen-Moves       */

    /* sonst UNDERFLOW; Energiezuwachs zu gross, immer abgelehnt */
    if (ediff/temp > 70.) return(1);

    if (ediff > 0.)
       if (double_r250() > exp(-ediff/temp)) return(1);

    /* ..................... accept move ........................... */

    /*  Periodische, gescherte Relativkoordinaten xbox,ybox aendern  */
    /*  sich nicht.  Kartesische Absolutkoordinaten xtrans,ytrans    */
    /*  als xmm,ymm akzeptieren                                      */
    tmpe   = xmm;
    xmm    = xtrans;
    xtrans = tmpe;

    tmpe   = ymm;
    ymm    = ytrans;
    ytrans = tmpe;

    /* change energy pointers                                        */
    tmpe    = energy;
    energy  = newenrg; /* Beim naechsten Volumen-Move werden die     */
    newenrg = tmpe;    /* vorletzten Energien ueberschrieben !       */

    tmpe    = cospot;
    cospot  = newcpot;
    newcpot = tmpe;

    /* WW-Tafel WW2 uebernimmt Rolle von WW1 und umgekehrt           */
    tmpe    = WW1;
    WW1     = WW2;
    WW2     = tmpe; /* Warnung ! WW2 enthaelt jetzt falsche Werte !! */

    /* Pointer WW1_in und WW2_in ebenfalls tauschen                  */
    tmpp    = WW1_in;
    WW1_in  = WW2_in;
    WW2_in  = tmpp;

    SSX     = trysyx;
    inv_SSX = 1.0/SSX;

    /* wenn sich Zahl der Zellen in x-Richtung geaendert hat,        */
    /* sind die Indizes der Nachbarzellen neu zu bestimmen           */
    if (ncx != number_of_cells_x)
      {
        if(ncx < 2){
              printf("SSX too low!!! ==> program stopped\n");
              exit(1);
        }
        if(ncx > ncx_alloc){
```

```
            printf("SSX too big!!! ==> program stopped\n");
            fflush(stdout);
            exit(1);
        }
        number_of_cells_x = ncx;
        redefine_neighbor_cells();

        /* wird fuer Relativkoord. benutzt, daher nicht  SSX/number  */
        CSX = 1.0/(double) number_of_cells_x;

        for (i1=0; i1<ntot; i1++)
          { nxcell   = itrunc(xbox[i1]*number_of_cells_x);

          /* Moeglichkeit, dass Monomer mit neuen Koordinaten in   */
          /* andere Zelle wechselt erfordert 'update_pointers'     */
            if (nxcell != xcell[i1])
                update_pointers(i1,nxcell,ycell[i1],zcell[i1]);
          }
      }

    /* evtl. Verlet-Tabelle updaten (nur komplett, denn andere   */
    /* Monomere wurden vermutl. auch schon bewegt !)             */
    d_diff_max = d_diff_max2 = 0.0;
    for (i1=0; i1<ntot; i1++)
      { diff = (xmm[i1]-x0ver[i1])*(xmm[i1]-x0ver[i1]) +
               (ymm[i1]-y0ver[i1])*(ymm[i1]-y0ver[i1]) +
               (zmm[i1]-z0ver[i1])*(zmm[i1]-z0ver[i1]);
        if (diff>d_diff_max2)
          if (diff>d_diff_max)
            { d_diff_max2 = d_diff_max;
              d_diff_max  = diff;
            }
          else
             d_diff_max2= diff;
      }

    /* Verlet-Tabelle wird upgedatet wenn die beiden groessten  */
    /* Monomer-Drifts seit letztem Update zusammen die Schicht- */
    /* dicke  skin_verlet  ueberschreiten.                      */
    if ((sqrt(d_diff_max2)+sqrt(d_diff_max))>=skin_verlet)
      verlet_update();

    return(0);
}

/* ---------------------------------------------------------------- */
/* (NpT)-Ensemble : Aenderung der Box-Grundflaeche in SSY-Richtung  */
/*                                                                  */
/*                  Parinello-Rahman-System                         */
/*                                                                  */
/*          ....................                                    */
/*        .                    .                                    */
/*       .                    .                                     */
/*      .___________________.                                       */
/*     /                   /       relativ Koordinaten xbox,ybox    */
/*    /                   /        sowie SSX und alpha unveraendert */
/*   /                   /                                          */
/*  /                   /                                           */
/* /___________________/                                            */
/*                                                                  */
/* im Gegensatz zu proportionale Trafo  y -> lamda*y    x -> x      */
/*                                                                  */
/*                                                                  */
```



```
/*      ....................                                        */
/*     .                    .                                      */
/*     ;                    ;                                      */
/*    .  _________________.__                                      */
/*    ; /                  ; /         wobei sich auch alpha aendert */
/*   . /                  . /                                      */
/*   ;/                   ;/                                       */
/*  ./                   ./                                        */
/*  /___________________/                                          */
/*                                                                 */
/*                                                                 */
/* --------------------------------------------------------------- */
/* Transformation fuer jedes Monomer bei einem volume_move_ssy der */
/* gescherten Box, wenn Relativkoordinaten erhalten und ausschl.   */
/* SSY geaendert wird.  Berechnung erfolgt aus xmm,ymm damit tats. */
/* Teilchenbewegung nachvollziehbar, mit xbox,ybox wuerden die     */
/* Teilchen in die Box gezwungen.                                  */

void shear_y_trans(lambda,xc,yc)
double lambda,*xc,*yc;
{ int i1;
  double fx;

  fx = (lambda-1.0)*ctg_alpha;

  /* Transformation wird mit nicht-period. Koordinaten berechnet !   */
  for (i1=0; i1<ntot; i1++)
    { xtrans[i1] = xc[i1] + fx*yc[i1];
      ytrans[i1] = lambda*yc[i1];
    }
}

/* --------------------------------------------------------------- */
/* voellig analog zu 'volume_move_ssx()', Kommentare siehe oben    */

int volume_move_ssy()
{
    int    i1,ncy,nycell;
    double tryshy,trysyy,by,enedif,cosdif,ediff,diff;
    double *tmpe,**tmpp;

    trysyy = SSY + volvar*(2.*double_r250() - 1.);
    tryshy = trysyy*sin(alpha);
    by     = trysyy/SSY;

    ncy    = itrunc(tryshy/(ljcut+skin_verlet));
    if(ncy < 2) return(1);

    shear_y_trans(by,xmm,ymm);

    svm_initialize_energy(SSX,trysyy,cosa2,newenrg);

    svm_initialize_ang_energies(newcpot);

    enedif= 0.0;
    cosdif= 0.0;
    for (i1=0; i1<ntot; i1++)
      { enedif = enedif - energy[i1] + newenrg[i1];
        cosdif = cosdif - cospot[i1] + newcpot[i1];
      }

    /* Auswerten der Energie-Aenderung durch den Volumen-Move        */
```

```
ediff = 0.5*enedif + cosdif +
        pressure*(tryshy - SHY)*SSX -
        ntot*log(by)*temp;

/* Metropolis-Entscheidung ueber Annahme des Volumen-Moves       */

if (ediff/temp > 70.) return(1);

if (ediff > 0.)
   if (double_r250() > exp(-ediff/temp)) return(1);

/* ..................... accept move ........................... */

tmpe   = xmm;
xmm    = xtrans;
xtrans = tmpe;

tmpe   = ymm;
ymm    = ytrans;
ytrans = tmpe;

tmpe    = energy;
energy  = newenrg;
newenrg = tmpe;

tmpe    = cospot;
cospot  = newcpot;
newcpot = tmpe;

tmpe    = WW1;
WW1     = WW2;
WW2     = tmpe;

tmpp    = WW1_in;
WW1_in  = WW2_in;
WW2_in  = tmpp;

SSY     = trysyy;
SHY     = tryshy;
inv_SHY = 1.0/SHY;

if (ncy != number_of_cells_y)
  {
    if(ncy < 2){
          printf("SSY too low!!! ==> program stopped\n");
          exit(1);
    }
    if(ncy > ncy_alloc){
          printf("SSY too big!!! ==> program stopped\n");
          fflush(stdout);
          exit(1);
    }
    number_of_cells_y = ncy;
    redefine_neighbor_cells();

    /* wird fuer Relativkoord. benutzt, daher nicht SSY/number  */
    CSY = 1.0/(double) number_of_cells_y;

    for (i1=0; i1<ntot; i1++)
      { nycell   = itrunc(ybox[i1]*number_of_cells_y);

        if (nycell != ycell[i1])
           update_pointers(i1,xcell[i1],nycell,zcell[i1]);
```



```
          }
      }

    d_diff_max = d_diff_max2 = 0.0;
    for (i1=0; i1<ntot; i1++)
      { diff = (xmm[i1]-x0ver[i1])*(xmm[i1]-x0ver[i1]) +
               (ymm[i1]-y0ver[i1])*(ymm[i1]-y0ver[i1]) +
               (zmm[i1]-z0ver[i1])*(zmm[i1]-z0ver[i1]);
        if (diff>d_diff_max2)
          if (diff>d_diff_max)
            { d_diff_max2 = d_diff_max;
              d_diff_max  = diff;
            }
          else
             d_diff_max2= diff;
      }

    if ((sqrt(d_diff_max2)+sqrt(d_diff_max))>=skin_verlet)
      verlet_update();

    return(0);
}

/* ------------------------------------------------------------------- */

#else   /* ifndef VH_Flag ... */
#include "vh.c"
#endif
#endif /* ifndef NVT_Flag */

#ifdef SHEAR

/* ------------------------------------------------------------------- */
/* (NpT)-Ensemble : Aenderung des Scherwinkels alpha durch Scherung  */
/*                                                                   */
/*                  Parinello-Rahman-System                          */
/*                                                                   */
/*      ....__________________                                       */
/*     ;   /               ;   / -ymm const.                         */
/*     ;  /                ;  /  -SSX und SHY, sowie xbox und ybox  */
/*    ;  /                ;  /    erhalten.                          */
/*    ; /                 ; /    -Flaeche ebenfalls unveraendert     */
/*   ; /                 ; /     -Neben alpha aendert sich auch      */
/*   ;/                  ;/       SSY                               */
/*  ;/                  ;/       -Zelleinteilung geht via SHY        */
/*  /__________________/         daher keine Aenderung moeglich      */
/*                                                                   */
/* im Gegensatz zu reiner alpha-Aenderung mit SSX und SSY const.     */
/*                                                                   */
/*       ....................                                        */
/*      ;   _________________;___                                    */
/*     ;   /               ;   /                                     */
/*     ;  /                ;  /                                      */
/*    ;  /                ;  /   -Aenderung von SHY und Flaeche      */
/*    ; /                 ; /                                       */
/*   ; /                 ; /                                        */
/*   ;/                  ;/                                         */
/*  ;/                  ;/                                          */
/*  /__________________/                                            */
/*                                                                   */
/* ------------------------------------------------------------------- */
/* Transformation fuer jedes Monomer bei einem volume_move_ssy der   */
/* gescherten Box, wenn relativkoordinaten erhalten und ausschl.     */
```

```c
/* SSY geaendert wird.  Berechnung erfolgt aus xmm,ymm damit tats.   */
/* Teilchenbewegung nachvollziehbar, mit xbox,ybox wuerden die       */
/* Teilchen in die Box gezwungen.                                    */

void shear_alpha_trans(fx,xc,yc)
double fx,*xc,*yc;
{ int i1;

  /* Transformation wird mit nicht-period. Koordinaten berechnet !   */
  for (i1=0; i1<ntot; i1++)
    { xtrans[i1] = xc[i1] + fx*yc[i1];
      ytrans[i1] = yc[i1];
    }
}

/* ---------------------------------------------------------------- */
/* voellig analog zu 'volume_move_ssx()', Kommentare siehe oben      */

int shear_box()
{
    int    i1,ncx,nxcell;
    double tryalf,tryctg,trysin,try_cosa2,trysyy,by,enedif,cosdif,ediff,diff;
    double *tmpe,**tmpp;

    /* neuen Scherwinkel auswuerfeln                                 */
    tryalf = alpha + alphvar*(2.*double_r250() - 1.);
    trysin = sin(tryalf);
    trysyy = SHY/trysin;   /*  neue Box-Hoehe, analog SSY-Move  */
    by     = trysyy/SSY;

    /* sicherstellen, dass Verletkugel noch in neue Box passt !      */
    /* impiziert zusaetzliches Potential, ist aber zur Auswertung    */
    /* der periodischen Randbedingungen noetig, sonst WW mit mehr    */
    /* als einem Bild eines Teilchens moeglich.                      */
    ncx    = itrunc(SSX*trysin/(ljcut+skin_verlet));
    if(ncx < 2) return(1);

    tryctg    = cos(tryalf)/trysin;
    try_cosa2 = 2.0*cos(tryalf);

    shear_alpha_trans(tryctg-ctg_alpha,xmm,ymm);

    svm_initialize_energy(SSX,trysyy,try_cosa2,newenrg);

    svm_initialize_ang_energies(newcpot);

    enedif= 0.0;
    cosdif= 0.0;
    for (i1=0; i1<ntot; i1++)
      { enedif = enedif - energy[i1] + newenrg[i1];
        cosdif = cosdif - cospot[i1] + newcpot[i1];
      }

    /* Auswerten der Energie-Aenderung durch den Scher-Move          */
    /* Da die Flaeche unveraendert, keine Terme  P*dA - NkT ln(A'/A) */
    ediff = 0.5*enedif + cosdif;

    /* Metropolis-Entscheidung ueber Annahme des Scher-Moves         */

    if (ediff/temp > 70.) return(1);

    if (ediff > 0.)
       if (double_r250() > exp(-ediff/temp)) return(1);
```



```c
    /* ..................... accept move ........................... */

    tmpe   = xmm;
    xmm    = xtrans;
    xtrans = tmpe;

    tmpe   = ymm;
    ymm    = ytrans;
    ytrans = tmpe;

    tmpe    = energy;
    energy  = newenrg;
    newenrg = tmpe;

    tmpe    = cospot;
    cospot  = newcpot;
    newcpot = tmpe;

    tmpe    = WW1;
    WW1     = WW2;
    WW2     = tmpe;

    tmpp    = WW1_in;
    WW1_in  = WW2_in;
    WW2_in  = tmpp;

    /* Verbuchung der akzeptierten neuen Boxgeometrie                */
    SSY       = trysyy;
    alpha     = tryalf;
    alpha_io  = 180.0*alpha/M_PI;
    ctg_alpha = tryctg;
    cosa2     = try_cosa2;

    /* wenn sich Zahl der Zellen in x-Richtung geaendert hat,        */
    /* sind die Indizes der Nachbarzellen neu zu bestimmen           */
    if (ncx != number_of_cells_x)
      {
        if(ncx < 2){
              printf("alpha too extreme, ncx = %d ! ==> program stopped\n",ncx);
              exit(1);
        }
        if(ncx > ncx_alloc){
              printf("alpha too extreme, ncx = %d ! ==> program stopped\n",ncx);
              exit(1);
        }
        number_of_cells_x = ncx;
        redefine_neighbor_cells();

        /* wird fuer Relativkoord. benutzt, daher nicht  SSX/number  */
        CSX = 1.0/(double) number_of_cells_x;

        for (i1=0; i1<ntot; i1++)
          { nxcell   = itrunc(xbox[i1]*number_of_cells_x);

            /* Moeglichkeit, dass Monomer mit neuen Koordinaten in   */
            /* andere Zelle wechselt erfordert 'update_pointers'     */
              if (nxcell != xcell[i1])
                 update_pointers(i1,nxcell,ycell[i1],zcell[i1]);
          }
      }

    d_diff_max = d_diff_max2 = 0.0;
```

```
    for (i1=0; i1<ntot; i1++)
      { diff = (xmm[i1]-x0ver[i1])*(xmm[i1]-x0ver[i1]) +
               (ymm[i1]-y0ver[i1])*(ymm[i1]-y0ver[i1]) +
               (zmm[i1]-z0ver[i1])*(zmm[i1]-z0ver[i1]);
        if (diff>d_diff_max2)
          if (diff>d_diff_max)
            { d_diff_max2 = d_diff_max;
              d_diff_max  = diff;
            }
          else
             d_diff_max2= diff;
      }

    if ((sqrt(d_diff_max2)+sqrt(d_diff_max))>=skin_verlet)
      verlet_update();

    return(0);
}

#endif

/* ------------------------------------------------------------------- */

#ifdef PT_Flag
#include "pt.c"   /* enthaelt die Ein-Ketten-Translations-Moves         */
#endif

/* ------------------------------------------------------------------- */
/* Setzt die Wechselwirkungs-Matrix auf Null                           */

void zero_WW()
{ int i,j,k;

  k = 0;
  for (i=0; i<ntot; i++)
    { WW1_in[i] = &WW1[k];
      WW2_in[i] = &WW2[k];
      for (j=i+1; j<ntot; j++)
        { WW1[k] = WW2[k] = 0.0; k++;
          if (k>WW_triangle)
            { printf(" > [WW_triangle] too small !\n");
              exit(1);
            }
        }
    }
  WW_max = k;
}

/* ................................................................... */
#ifdef DIAGNOSE
#include "diagnostic.c"
#endif
/* ................................................................... */

/* ------------------------------------------------------------------- */
/* Zentrale Definition des Bond-Laengen-Potentials (hier logarithmisch) */

/* Kruemmung wie bei parabolischem Potential, aber stetige Divergenz in */
/* den minimalen und maximalen Bondlaengen                              */

double bond_laengen_potential(dist)
double dist;
{ return(-sprng*bond_limit_quadrat*
```



```
        log(1.0-(dist-bond_length)*(dist-bond_length)/bond_limit_quadrat));
}


/* ------------------------------------------------------------------- */
/* Umrechnung der kartesischen allg. Koordinaten in die periodischen   */
/* Koordinaten der gescherten Box                                      */
/*       ____________________________                                  */
/*      /                            /                                 */
/*     /                            /                                  */
/*    /...xb*SSX....o              /     ==>  0 <= xb < 1             */
/*   /             /              /                                   */
/*  /             yb*SSY          /       ==>  0 <= yb < 1             */
/* /______________/________________/                                   */
/*                                                                     */
/* Abstandsberechnung ist korrekt, wenn Abstand kleiner als Durchmesser */
/* der groessten Kugel, die noch in Box passt -> Verletkugel.          */
/* Andernfalls ist Abstand evtl. falsch, aber immer groesser als Kugel- */
/* durchmesser und damit als der LJ-Abschneide-Radius !                */
/* [Allen, Tildesley, "Computer Simulation in Chemical Physics"]       */

void shear_period(xb,yb,xm,ym,inv_ssx,inv_shy,ctg_alfa)
double *xb,*yb,xm,ym,inv_ssx,inv_shy,ctg_alfa;
{
  *xb = fmod((xm-ctg_alfa*ym)*inv_ssx+32.0,1.0);
  *yb = fmod( ym             *inv_shy+32.0,1.0);
}

/* ------------------------------------------------------------------- */
/* Optimierung der Skin-Dicke ueber CPU-Zeit/Savestep                  */

void optimize_skin(double T_so_alt, double T_so_neu)
#define INCMAX 1.10
#define DECMAX 0.90
{ static double fact = INCMAX;
  static int sdc = 0;
  double skin_try,fact_old;
  int ncx,ncy,ncz,i1,nx,ny,nz,sdc_old;

  sdc_old = sdc;  fact_old = fact;

  if (T_so_neu>T_so_alt)
    { /* Programm wurde durch letzte Skin-Aenderung langsamer !        */
      /* 2. Richtung der Aenderung umkehren                            */
      /* 1. Naehe des Optimums ==> Schrittweite verkleinern            */
      sdc = 0;
      if ((fact>1.005)||(fact<0.995)) fact = sqrt(fact);

      fact = 1.0/fact;
    }
  else
    { sdc++;
      /* dreimal in gleiche Richtung --> Vergroesserung des Faktors    */
      if (sdc>2)
        if ((fact<INCMAX)&&(fact>DECMAX)) fact = fact*fact;
    }

  /* Neue Skin-Dicke berechnen                                         */
  skin_try = skin_verlet*fact;
  ncx = floor(SSX*sin(alpha)/(ljcut_max+skin_try));
  ncy = floor(SHY           /(ljcut_max+skin_try));
  ncz = floor(system_size_z/(ljcut_max+skin_try));
```

```
  if ((ncx>=2)&&(ncy>=2)&&(ncx<ncx_alloc)&&(ncy<ncy_alloc))
    { /* Neue Skin-Dicke verarbeiten                                  */
      if ((ncx != number_of_cells_x)||
          (ncy != number_of_cells_y)||
          (ncz != number_of_cells_z))
        { number_of_cells_x = ncx;
          number_of_cells_y = ncy;
          number_of_cells_z = ncz;
          redefine_neighbor_cells();

          CSX         = 1.0/(double)ncx;
          CSY         = 1.0/(double)ncy;
          cell_size_z = system_size_z/(double)ncz;
          for (i1 = 0; i1<ntot; i1++)
             { nx = itrunc(xbox[i1]*ncx);
               ny = itrunc(ybox[i1]*ncy);
               nz = itrunc(zmm[i1]/cell_size_z);
               if ((nx != xcell[i1])||(ny != ycell[i1])||(nz != zcell[i1]))
                 update_pointers(i1,nx,ny,nz);
        }
        }

      skin_verlet = lj_safety = skin_try;
      /* printf("%lf %lf %lf %lf\n",mcs,T_so_neu,skin_verlet,fact); */

     verlet_update();
     verlet_update_counter--;
    }
  else { fact = fact_old;  sdc = sdc_old; }

}

/* ------------------------------------------------------------------- */
/* PROGRAM SWITCHES for main                                           */
/*                                                                     */
/* swtch[1]        : prefix for filenames                              */
/* swtch[2][0] = f : parameters from input file                        */
/*             = c : parameters from existing configuration file       */
/* swtch[2][1] = c : start from existing configuration file            */
/*             = s : start from stretched chains                       */
/*             = o : start with orthorhombic lattice for headmonomers  */
/*                   in upright position                               */
/*             = g : start from approximate groundstate                */
/* swtch[2][2] = m : make measurements                                 */
/*                                                                     */
/* FILES                                                               */
/* <prefix>.ein    : input parameters                                  */
/* <prefix>_00.cnf : start configuration                               */
/* <prefix>_01.cnf : final configuration                               */
/* <prefix>_<l>.dat: data files                                        */
/* ------------------------------------------------------------------- */


void main(int argc, char *argv[])
{
     int    autoskin,sig,i,idir_start,no_save_yet,isamp;
     long   itime,tmax,icount,ivmx,ivmy,ivmz,iptm,ishr;
     long   jcount,jptm,jvm;
     double accpt,vaccptx,vaccpty,vaccptz,shracpt,mcs0,mcsave,savestep;
     double ptaccpt,wall_secs,CPU_secs,CPU_limit,CPU_old;
     time_t t_start,t_end;

     struct tms u_start,u_end;
```



```
     struct tms so_start,so_end;
     double T_so_alt,T_so_neu;

     XCPU = 0;
#ifdef SIGHAND
     /* Wenn Programm von aussen (DQS) ein sig-Signal erhaelt, wird -   */
     /* unabhaengig vom momentanen Programm-Zustand ! - die Funktion    */
     /* "handler" aufgerufen. Diese setzt einfach XCPU:=sig.            */

     for (sig=1; sig<=31; sig++)
         if ((sig!=SIGCONT))
            signal(sig,handler);
#endif

     time(&t_start);        /* Systemzeit bei Simulationsbeginn speichern  */
     times(&u_start);
     times(&so_start); T_so_alt = 0.0;

     Pro = fopen("CPU.time","r");
     if (Pro==NULL)
       { CPU_old = 0.0;
         CPU_limit = 3.6e10;
       }
     else
       { fscanf(Pro,"%lf",&CPU_old);
         fscanf(Pro,"%lf",&CPU_limit);
         fclose(Pro);
       }
     printf("\nCPU_old : %0.2f,   CPU_limit : %0.2f sec\n",CPU_old,CPU_limit);

     Pro = fopen("skin","r");
     fscanf(Pro,"%lf",&skin);
     fscanf(Pro,"%d",&autoskin);
     fclose(Pro);
     printf(" > Verlet skin set to   : %7.5f",skin);
     if (autoskin) printf("  automatic\n");
        else       printf("  fixed\n");


     swtch = argv; /* Switches aus Argumentliste uebernehmen        */
                   /* swtch[1] = Aufrufname =: Filename fuer xx.cfg */

     if (argc==1) {
         swtch[1]="y4y";
         swtch[2]="ccn";
         printf("No args given ==> old config in y4y_01.cnf continued\n");
     }

     printf(" > cmd line arguments   :");
     for (i=1; i<argc; i++)
       { printf(" %s",swtch[i]); }
     printf("\n");
     fflush(stdout);

   /* sollen auch alle Zwischen-Konfigurationen aufgehoben werden ?        */
      cfg_save_flag = (swtch[2][2]=='M');
      if (cfg_save_flag) { swtch[2][2]='m';  cfg_save_counter=-1; }

   /* (swtch[2][0]=='f') --> lese Parameter vom input-File                 */
      if (swtch[2][0]=='f')
        { read_parameters(); mcs0=0.0; }                    /* < MC_s.c > */

   /* (swtch[2][1]=='c') --> starte aus existierendem Konfigurations-File  */
```

```
   if ((swtch[2][1]=='c')&&(swtch[2][0]!='c'))
       read_config_sysi();                              /* < MC_s.c > */

/* (swtch[2][0]=='c') --> lese Parameter aus exist. Konfigurations-File */
   if (swtch[2][0]=='c') read_from_config(&mcs0);        /* < MC_s.c > */

/* (immer) benoetigten Speicherplatz belegen         */
   allocate_memory();

/* fuer Messungen  benoetigten Speicherplatz belegen -  steht in MC_m.c */
   if (swtch[2][2]=='m') allocate_msr_memory();

   isamp = 0;
   ivmx = 0;
   ivmy = 0;
   iptm = 0;
   ishr = 0;
   itime = 0;

/* ....... Bestimme die Startposition ................................ */
/* zug. start_from_...-Funktionen sind alle in MC_s.c                 */

/* 'c' : aus existierendem Konfigurationsfile
   'O' : orthorhombisches Kopfgitter mit senkrechten Ketten
   's' : aus approximiertem Grundzustand NN-geneigt
   'S' : aus NNN-geneigtem Grundzustand

   fuer Verwendung mit gescherter Box weniger geeignet, alte Versionen :


   'o' : orthorhombisches Kopfgitter mit senkrechten Ketten
   'g' : aus approximiertem Grundzustand NN-geneigt
   'G' : aus NNN-geneigtem Grundzustand                              */

   if  (swtch[2][1]=='c')
     { dim =3; start_from_config(1); }

   if  (swtch[2][1]=='o')
      { dim =3; start_from_ortho();  }
   if  (swtch[2][1]=='O')
      { dim =3; start_from_shear_ortho();  }

   if (swtch[2][1]=='g')
      { dim =3; start_from_ground_nnn(); }
   if (swtch[2][1]=='G')
      { dim =3; start_from_ground_nn(); }

   if (argc!=4) { idir_start = 0; }
   else
     { sscanf(swtch[3],"%d",&idir_start);
       if ((idir_start<0)||(idir_start>5))
         { printf(">>> start direction out of bounds ! %d , using 0\n",
                  idir_start);
           fflush(stdout);
           idir_start = 0;
         }
     }

   if (swtch[2][1]=='s')
      { dim =3; start_from_shear_nnn(idir_start); }
   if (swtch[2][1]=='S')
      { dim =3; start_from_shear_nn(idir_start); }
/* ................................................................... */
```



```
  /* weitere Vorbereitung fuer Lauf mit durchzufuehrenden Messungen        */
  /* alle in MC_m.c                                                        */

     if (swtch[2][2]=='m')
        if (swtch[2][0]!='c')
          { prepare_msr_files();             /* *.dat-Files initialisieren */
            initialize_histo();              /* Histogrammzaehler nullen   */
          }
        else
          { read_histo(&isamp);             /* alte Histogramme fortsetzen */
          }

  /* Wechselwirkungs-Matrix mit Nullen vorbesetzen                         */
     zero_WW();

  /* initialize_... in MC_s.c                                              */

  /* tabellarisiere LJ-Potential(e)                                        */
     initialize_LJ_table();

  /* Einordnen der Monomere in die Verlet-Tabelle                          */
     verlet_update();
     verlet_update_counter = 0;

  /* Werte Bond- und LJ-Wechselwirkungen aus                               */
     initialize_energy(energy);

  /* Werte Bond-Winkelpotentiale aus                                       */
     initialize_ang_energies(cospot);

  /* steht in MC_m.c; bestimmt Gesamtenergie der Konfiguration             */
  /* ( zumindest Bond-, LJ- und Bond-Winkel-Energien )                     */

     printf("initial energy : %f\n",msr_energy());

     if (lpm>=MAX_LPM)
       { printf("\n > Chains are too long !  Change MAX_LPM in MC.h \n");
         exit(1);
       }

     if (ntot>=MAX_MONOMERS)
       { printf("\n > Too many Monomers,  resize WW_triangle in MC.h !\n");
         exit(1);
       }

/*  - - - - - - Hauptschleife ueber die Simulations-Schritte - - - - - - - */
/* Ein 'mcs' entspricht 'ntot' akzeptierter Monomermoves, diese werden     */
/* durch Aufruf von 'move' bewirkt, return=0 entspricht akzeptiert.        */
/* Nach einem ganzen 'mcs' werden Volumen-Moves versucht                   */

     printf("starting at %12.5f MCS, measuring from %7.0f MCS, ending at %7.0f M
CS\n",
             mcs0,mcs_aequi,mcsmax); fflush(stdout);

     tmax = (long) ((mcsmax-mcs0)*ntot);  /* Zahl der verbleibenden MM-Moves */

     savestep = 500.0;      /* Abstand der zu speichernden Konfigurationen */
     mcsave = ((int)(mcs0/savestep)+1)*savestep;
                            /* mcs-Nummer der naechsten Speicherung        */
     icount = 0;            /* Zaehler fuer gelungene Moves                */
     ivmx   = 0;            /* Zaehler fuer gel. Volumenmoves in x-Richt.  */
     ivmy   = 0;            /*                                   y-        */
```

```
     ivmz   = 0;            /*                                    z-        */
     iptm   = 0;            /* Zaehler kollektiver Ketten-Translationen    */
     ishr   = 0;            /* Zaehler fuer Box-Scher-Versuche             */

     jcount = 0;
     jptm   = 0;
     jvm    = 0;

     WEITER = TRUE;
     no_save_yet = TRUE;

     mcs = mcs0; save_configuration(0);

     for (itime=1; ( (itime<=tmax) && (WEITER) ); itime++) {

        mcs = mcs0 + (double)itime/(double)ntot;

        jcount++;
        if (move()==0) icount++;             /* Aufruf eines Monomer-Moves */

       if (itime%(lpm)==0)
          { jptm++;
#ifdef PT_Flag
            if (poly_trans_move()==0) iptm++;
            /*    bei diesen ist Moves ist ggf. ueber die Definition des   */
            /*    Monte-Carlo-Schrittes nachzudenken.                      */
#endif
          }

        if (itime%(ntot)==0){
                jvm++;
                /* volume_move_ssx/y : hom. Reskalierung in einer Richtung */
                /*                                                         */
                /* vol_hom_trans     : hom. Reskalierung des Kopfgruppen-  */
                /*                     gitter in BEIDEN Raumrichtungen !   */
                /*                     Translation der Ketten darueber,    */
                /*                     Scherwinkel unveraendert            */

#ifndef NVT_Flag
#ifndef VH_Flag /* ..................................................... */
                if (volume_move_ssx()==0) ivmx++;  /* Aufruf Volumen-Moves */
                if (volume_move_ssy()==0) ivmy++;
#else           /* ..................................................... */
                if (vol_hom_trans()==0) { ivmx++; ivmy++; }
#endif          /* ..................................................... */
#endif

#ifdef SHEAR
                if (shear_box()==0) ishr++;        /* Aufruf Shear-Moves   */
#endif

#ifdef DIAGNOSE
                if (check_WW(mcs)>0) check_verlet(mcs);
#endif
        }

        /* wenn Messungen gewuenscht werden und mcsave Schritte weiter     */
        /* entwickelt worden ist.                                          */
        if (mcs >= mcsave)
          {  /* CPU-Zeitmessung fuer Skin-Optimierung bei jedem Savestep   */
             times(&so_end);
             T_so_neu = (so_end.tms_utime - so_start.tms_utime);
             if (T_so_alt==0.0) T_so_alt = T_so_neu;
```



```
        if ((swtch[2][2]=='m')&&(mcs >= mcs_aequi))
           { /* Akzeptanzrate = #akz. Monomoves / #vers. Monomoves         */
             if (itime==1)
               { accpt = ptaccpt = vaccptx = vaccpty = vaccptz = shracpt = 0.0;
}
             else
               { accpt = (double) icount/(double) jcount;
                 /* Akzeptanzrate fuer kollektive Ketten-Translationen     */
                 ptaccpt = (double) iptm/(double) jptm;
                 vaccptx = (double) ivmx/(double) jvm; /* Akz.raten fuer   */
                 vaccpty = (double) ivmy/(double) jvm; /* Volumen-Moves    */
                 vaccptz = (double) ivmz/(double) jvm;
                 shracpt = (double) ishr/(double) jvm; /* fuer Scherungen  */
               }

             icount = jcount = 0;
             iptm   = jptm   = 0;
             ivmx = ivmy = ivmz = ishr = jvm = 0;

             /* zentrale Funktion zur Ausfuehrung von Messungen         */
             /* steht in <MC_m.c>                                    */
             measurement(accpt,vaccptx,vaccpty,skin_verlet,shracpt,ptaccpt);

             isamp++;
             /* Histographie der Ketten-Achsen und vert. Schichtung    */
             /* steht in <MC_m.c>                                      */
             add_to_histo();
             if (mcs >= 1.0)
                save_configuration(1);

             no_save_yet = FALSE;
           }
             /* Berechnung der neuen skin-Dicke und entspr. Umstellung  */
             if (autoskin) optimize_skin(T_so_alt,T_so_neu);
             T_so_alt = T_so_neu;
             times(&so_start);

             mcsave = mcsave + savestep;

             times(&u_end);
             CPU_secs  = (u_end.tms_utime - u_start.tms_utime)/
                         (double) sysconf(_SC_CLK_TCK);
             WEITER = ( ((CPU_old+CPU_secs)<CPU_limit) && WEITER );
          }

        /* in jedem istep, d.h. ntot-mal pro MCS wird XCPU ausgewertet  */
        /* d.h. groesste Totzeit sind 2xVolumen-Move + 1xMeasurement    */
        /*                                                              */
        /* Programm schreibt nur noch Histogramme raus und endet dann.  */
        /* Geht schnell und verliert hoechsten einen savestep.          */
        /* Letzter Eintrag in Histogramme war bei letztem savestep !    */

          WEITER=( (XCPU==0) && WEITER );
        }  /* next itime */

/* - - - - - - - - - - Hauptschleifen-Ende  - - - - - - - - - - - - - - -  */

        if (! WEITER)
           printf(">>>  forced early ending at %f MCS, caught signal %d  <<<\n",
mcs,XCPU);

        time(&t_end);                    /* Systemzeit bei Schleifen-Ende */
```

MC.c **29/30**

```
        wall_secs  = (t_end-t_start);     /* Zeit fuer Schleifendurchlauf  */

        times(&u_end);
        CPU_secs  = (u_end.tms_utime - u_start.tms_utime)/
                    (double) sysconf(_SC_CLK_TCK);
        printf(" > main                  :");
        printf(" %-10.2f CPU seconds to execute\n",CPU_secs);
        printf(" > main                  :");
        printf(" %-10.2f attempts / CPU second\n",itime/CPU_secs);

        printf(" > main                  :");
        printf(" %-10.2f wall seconds to execute\n",wall_secs);
        printf(" > main                  :");
        printf(" %-10.2f attempts / wall second\n",itime/wall_secs);

        printf("%5d Verlet updates in %5d MCSs --> rate %15.6f\n",
                 verlet_update_counter,(int) (itime/ntot),
                 (double) (verlet_update_counter*ntot)/itime);

        /* i.A. sollte bei Abbruch von aussen keine neue Konfiguration     */
        /* geschrieben werden, sondern die vom letzten Savestep als Start- */
        /* konfiguration bei der Fortsetzung dienen                        */
        if ((swtch[2][2]!='m')||(no_save_yet))
          { save_configuration(1); }

        if (swtch[2][2]=='m')
          { /* Fortschreibung der Histogramme                             */
            /* steht in <MC_m.c>                                       */
            print_histoxy(isamp);
            print_histoz(isamp);
            print_histoang(isamp);
            print_histodst(isamp);
            print_historpvf(isamp);
          }

        /* durch Aufruf des Zufallsgenerators wird neuer Wert fuer iseed   */
        /* erzeugt                                                         */
        /* [tmp] write_seed(r250_random()); */

        /* ueberprueft abschliessend Speicherallocation und gibt dann den  */
        /* reservierten Speicherplatz wieder frei                          */
        /* definiert als :  >>  mallprot(a,-1,a); free((char *) a);  <<    */
#ifdef kollektiv
        mfree(akz_engy);
#endif
        mfree(cell);

        /* aktuelle, optimierte Skindicke rausschreiben - Startwert        */
        if (autoskin)
          { Pro = fopen("skin.new","w");
            fprintf(Pro,"%f\n",skin_verlet);
            fprintf(Pro,"%d\n",autoskin);
            fclose(Pro);
          }

        /* gesamte verbrauchte CPU-Zeit und CPU-Limit (in sec) ausgeben    */
        Pro = fopen("CPU.time","w");
        fprintf(Pro,"%f\n",CPU_old+CPU_secs);
        fprintf(Pro,"%f\n",CPU_limit);
        fclose(Pro);

        if (itime>tmax) system("date > finished");
}
```

MC.c **30/30**

"Man hat in jedem Programm bisher nur den vorletzten Fehler entdeckt."

```c
/* ======================================================== */
/* Dieses Include-File enthaelt read_* und init_*-Routinen */
/* sowie "verlet_update"                                    */
/* ======================================================== */

#include "switchboard.h"

#include <malloc.h>
#include <stdio.h>
#include <math.h>
#include <time.h>
#include "r250.h"
#include "matrix.h"
#include "MC.h"
#include <stdlib.h>

#define c_2 ((char) 2)
#define c_1 ((char) 1)
#define c_0 ((char) 0)

/* ------------------------------------------------------------------- */
/* Liest aus "seed.rnd" den Initialisierungswert des Zufallsgenerators */

int read_seed()
{
        int iseed;
        FILE *file_1;
        char zeile[80];

        file_1 = fopen("seed.rnd","r");
        if (file_1==NULL)
                printf(" > read_seed           : ERROR seed.rnd not found\n");
        fgets(zeile, 80, file_1);
        sscanf(zeile,"%d ",&iseed);
        fclose(file_1);
        return(iseed);
}

/* ------------------------------------------------------------------- */
/* Schreibt iseed auf File "seed.rnd"                                  */

void write_seed(int iseed)
{
        FILE *file_1;

        file_1=fopen("seed.rnd","w");
        fprintf(file_1,"%d\n",iseed);
        fclose(file_1);
}

/* ------------------------------------------------------------------- */

void read_parameters()
{
        int  iseed,idummy;
        FILE *file_1;
        char zeile[80],fname[20];
        double l6,l12,l6_hh,l12_hh,l6_hm,l12_hm,rlc,rlc_hh,rlc_hm;

        sprintf(fname,"%s.ein",swtch[1]);
        file_1=fopen(fname,"r");
        if (file_1 == NULL) printf("ERROR opening %s\n",fname);
```



```c
        fgets(zeile, 80, file_1);
        sscanf(zeile,"%d ",&npm);             /* Anz. Ketten                 */
        fgets(zeile,80,file_1);
        sscanf(zeile,"%d ",&lpm);             /* Anz. Knoten pro Ketten      */
        fgets(zeile,80,file_1);
        idummy = sscanf(zeile,"%lf %lf",&mcsmax,&mcs_aequi);
        if (idummy!=2) mcs_aequi = 0.0;
                                        /* max. Anz. MC-Schritte, Mess-Beginn */
        fgets(zeile,80,file_1);
        sscanf(zeile,"%lf ",&SSX);               /* Box-X-Ausdehnung         */
        fgets(zeile,80,file_1);
        sscanf(zeile,"%lf %lf",&SHY,&alpha_io);  /* Box-Hoehe, Scherwinkel   */
 alpha     = M_PI*alpha_io/180.0;
 ctg_alpha = cos(alpha)/sin(alpha);
 cosa2     = 2.0*cos(alpha);
 SSY       = SHY/sin(alpha);
 inv_SSX   = 1.0/SSX;
 inv_SHY   = 1.0/SHY;

        fgets(zeile,80,file_1);
        sscanf(zeile,"%lf ",&temp);           /* Temperatur                  */
        fgets(zeile,80,file_1);
        sscanf(zeile,"%lf ",&pressure);       /* Spreizdruck                 */
        fgets(zeile,80,file_1);
        sscanf(zeile,"%lf ",&u0);             /* LJ-Staerke-Vorfaktor        */
        fgets(zeile,80,file_1);
        sscanf(zeile,"%lf ",&ljmin);          /* LJ-Minimum-Lage <sigma>     */
        fgets(zeile,80,file_1);
        sscanf(zeile,"%lf ",&ljcut);          /* LJ-Reichweite <d_LJ>        */
        fgets(zeile,80,file_1);
        sscanf(zeile,"%lf ",&u0_hh);          /* entspr. fuer Kopf-Kopf-WW   */
        fgets(zeile,80,file_1);
        sscanf(zeile,"%lf ",&ljmin_hh);
        fgets(zeile,80,file_1);
        sscanf(zeile,"%lf ",&ljcut_hh);
        fgets(zeile,80,file_1);
        sscanf(zeile,"%lf ",&u0_hm);          /* entspr.fuer Kopf-Monomer-WW */
        fgets(zeile,80,file_1);
        sscanf(zeile,"%lf ",&ljmin_hm);
        fgets(zeile,80,file_1);
        sscanf(zeile,"%lf ",&ljcut_hm);

 ljcut_max = max(ljcut,max(ljcut_hh,ljcut_hm));

        fgets(zeile,80,file_1);
        sscanf(zeile,"%lf ",&hvar);           /* Weite der Monomer-Moves   */
        fgets(zeile,80,file_1);
        sscanf(zeile,"%lf %lf",&volvar,&alphvar_io); /* Weite der Volumen- u. Wi
nkel-Moves */
 lj_safety = skin;
 alphvar   = M_PI*alphvar_io/180.0;

        fgets(zeile,80,file_1);
        sscanf(zeile,"%lf ",&sprng);       /* Federkonst.(rad) zw. Monomeren */
        fgets(zeile,80,file_1);
        sscanf(zeile,"%lf ",&cospr);       /* Federkonst.(ang) zw. Monomeren */
        fgets(zeile,80,file_1);
        sscanf(zeile,"%lf ",&bond_min);    /* unt. Anschlag (rad) Monomere   */
        fgets(zeile,80,file_1);
        sscanf(zeile,"%lf ",&bond_max);    /* ob.  Anschlag (rad) Monomere   */
        fgets(zeile,80,file_1);
        sscanf(zeile,"%i ",&dim);          /* Dimensionalitaet {2;3}         */
```

```
   fgets(zeile,80,file_1);
   sscanf(zeile,"%c ",&heads);        /* Kopf-Param  0:fest 1:bewegl.   */
   fclose(file_1);

/* Liest aus "seed.rnd" randomize-Wert iseed                            */
   iseed = read_seed();
   printf(" > read_from_config     : iseed = %i read from file\n",iseed);

/* Bondlaenge (Gleichgewicht) als Mitte zwischen bond_min und bond_max */
   bond_length = 0.5*(bond_max+bond_min);

   bond_limit_quadrat = (bond_max - bond_min)*(bond_max - bond_min)/4.0;
   bond_min_quadrat = bond_min*bond_min;
   bond_max_quadrat = bond_max*bond_max;

/* Zellgroesse entspr. Reichweite der LJ-Wechselwirkung + safety */
   cell_size_z = ljcut_max+lj_safety;
   number_of_cells_z  = ceil((1+(lpm-1)*bond_max)/cell_size_z +5.);
   system_size_z = number_of_cells_z * cell_size_z ;
/* +5, damit keine WW zwischen Kopfmonomeren und Kettenenden auftritt */

/* [old] CSX(Y) > ljcut_max because of volume moves             */
/* [old]  volvar/number_of_cells_x(y) < volvar/3. !             */

/* [new] cell_size > ljcut_max because cells are used for       */
/* [new] updating the verlet table                              */
   number_of_cells_x  = floor(SSX*sin(alpha) / (ljcut_max+lj_safety));
   CSX = 1.0 / number_of_cells_x;
   number_of_cells_y  = floor(SHY            / (ljcut_max+lj_safety));
   CSY = 1.0 / number_of_cells_y;


/* Skaliertes Lennard-Jones-Potential                           */
/* Auswertung der LJ-Potentiale am Abschneideradius, um dort    */
/* auf Null zu verschieben                                      */
   rlc    = ljmin    / ljcut;
   rlc_hh = ljmin_hh / ljcut_hh;
   rlc_hm = ljmin_hm / ljcut_hm;

   l6     = rlc*rlc*rlc*rlc*rlc*rlc;
   l12    = l6*l6;
   l6_hh  = rlc_hh*rlc_hh*rlc_hh*rlc_hh*rlc_hh*rlc_hh;
   l12_hh = l6_hh*l6_hh;
   l6_hm  = rlc_hm*rlc_hm*rlc_hm*rlc_hm*rlc_hm*rlc_hm;
   l12_hm = l6_hm*l6_hm;

   lj0    = l12   -2.0*l6;
   lj0_hh = l12_hh-2.0*l6_hh;
   lj0_hm = l12_hm-2.0*l6_hm;

   ljcut_quadrat    = ljcut * ljcut ;
   ljcut_hh_quadrat = ljcut_hh * ljcut_hh ;
   ljcut_hm_quadrat = ljcut_hm * ljcut_hm ;

   lj_scala    = LJ_grid/ljcut_quadrat;
   lj_hh_scala = LJ_hh_grid/ljcut_hh_quadrat;
   lj_hm_scala = LJ_hm_grid/ljcut_hm_quadrat;

/* Zahl Monomere = Zahl Ketten * Zahl Monomere pro Kette */
   ntot = npm*lpm;

/* Initialisiere den Zufallszahlengenerator */
   r250_srandom(iseed);
```



```
}

/* ---------------------------------------------------------------- */
/* Entnimmt bestehender Konfiguration einige Parameter, die sonst
   read_parameters() aus *.ein lesen wuerde                          */
/* Erklaerungen siehe auch dort ...                                  */

void read_from_config(double *pmcs0)
{
        int iseed;
        FILE *file_1;
        char zeile[160],fname[20];
        double l6,l12,l6_hh,l12_hh,l6_hm,l12_hm,rlc,rlc_hh,rlc_hm;

        sprintf(fname,"%s_01.cnf",swtch[1]);
        file_1=fopen(fname,"r");
        if (file_1 == NULL) printf("ERROR opening %s\n",fname);

        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        sscanf(zeile,"%d %d %d %lf %lf %lf",
                      &dim,&npm,&lpm,&mcsmax,&temp,&pressure);
        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        sscanf(zeile,"%lf %lf %lf %lf %lf %lf %lf %lf",
                      &SSX,&SHY,&alpha_io,
                      &bond_min,&bond_max,&hvar,&volvar,&alphvar_io);

 alpha     = M_PI*alpha_io/180.0;
 ctg_alpha = cos(alpha)/sin(alpha);
 cosa2     = 2.0*cos(alpha);
 alphvar   = M_PI*alphvar_io/180.0;
 SSY       = SHY/sin(alpha);
 inv_SSX   = 1.0/SSX;
 inv_SHY   = 1.0/SHY;

        lj_safety = skin;
        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        sscanf(zeile,"%lf %lf %lf %lf %lf",
               &u0,&sprng,&cospr,&ljmin,&ljcut);
        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        sscanf(zeile,"%lf %lf %lf",
                      &u0_hh,&ljmin_hh,&ljcut_hh);
        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        sscanf(zeile,"%lf %lf %lf",
                      &u0_hm,&ljmin_hm,&ljcut_hm);

 ljcut_max = max(ljcut,max(ljcut_hh,ljcut_hm));

        fgets(zeile,160,file_1);
        sscanf(zeile,"%c",
                      &heads);
        fgets(zeile,160,file_1);
        sscanf(zeile,"%lf",
                      pmcs0);
        ntot = npm*lpm;
        *pmcs0 =   ((long)((*pmcs0 * ntot)+0.5))/(double)ntot;
```

```
fgets(zeile,160,file_1);
sscanf(zeile,"%lf",
               &mcs_aequi);

fclose(file_1);


iseed = read_seed();
printf(" > read_from_config     : iseed = %i read from file\n",iseed);

bond_length = 0.5*(bond_max+bond_min);
bond_min_quadrat = bond_min*bond_min;
bond_max_quadrat = bond_max*bond_max;
bond_limit_quadrat = (bond_max - bond_min)*(bond_max - bond_min)/4.0;

cell_size_z = ljcut_max + lj_safety;
number_of_cells_z = ceil((1+(lpm-1)*bond_max)/cell_size_z+ 5.);
system_size_z = number_of_cells_z * cell_size_z ;

/* +++++++++++++++++++++++  ACHTUNG ++++++++++++++++++++++++++ */
/*                                                             */
/* kleiner Trick : z-Richtung ist natuerlich nicht period.     */
/* fuer            wird aber periodisch verarbeitet (um sich   */
/* linked cells    Fallunterscheidungen zu sparen)             */
/*                 Dazu ist es noetig, die z-Laenge der Box    */
/*                 soweit zu vergroessern, dass auf der        */
/*                 gegenueberliegenden Seite die Zellen        */
/*                 zuverlaessig leer sind !  --> "+5"          */
/*                 Denn keine WW reicht ueber 5 Zellen hinweg  */
/*                                                             */
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */

number_of_cells_x  = floor(SSX*sin(alpha) / (ljcut_max+lj_safety));
CSX = 1.0 / number_of_cells_x;
number_of_cells_y  = floor(SHY            / (ljcut_max+lj_safety));
CSY = 1.0 / number_of_cells_y;

rlc    = ljmin    / ljcut;
rlc_hh = ljmin_hh / ljcut_hh;
rlc_hm = ljmin_hm / ljcut_hm;

l6     = rlc*rlc*rlc*rlc*rlc*rlc;
l12    = l6*l6;
l6_hh  = rlc_hh*rlc_hh*rlc_hh*rlc_hh*rlc_hh*rlc_hh;
l12_hh = l6_hh*l6_hh;
l6_hm  = rlc_hm*rlc_hm*rlc_hm*rlc_hm*rlc_hm*rlc_hm;
l12_hm = l6_hm*l6_hm;

lj0    = l12   -2.0*l6;
lj0_hh = l12_hh-2.0*l6_hh;
lj0_hm = l12_hm-2.0*l6_hm;

ljcut_quadrat    = ljcut    * ljcut ;
ljcut_hh_quadrat = ljcut_hh * ljcut_hh ;
ljcut_hm_quadrat = ljcut_hm * ljcut_hm ;

lj_scala    = LJ_grid/ljcut_quadrat;
lj_hh_scala = LJ_hh_grid/ljcut_hh_quadrat;
lj_hm_scala = LJ_hm_grid/ljcut_hm_quadrat;

r250_srandom(iseed);
```



```
}

/* ------------------------------------------------------------------ */
/* Entnimmt bestehender Konfiguration einige Parameter, die sonst
   read_parameters() aus *.ein lesen wuerde                            */
/* Erklaerungen siehe auch dort ...                                    */

void read_config_sysi()
{
        FILE *file_1;
        char zeile[160],fname[20];

        sprintf(fname,"%s_01.cnf",swtch[1]);
        file_1=fopen(fname,"r");
        if (file_1 == NULL) printf("ERROR opening %s\n",fname);

        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);

        sscanf(zeile,"%lf %lf %lf",&SSX,&SHY,&alpha_io);

 alpha     = M_PI*alpha_io/180.0;
 ctg_alpha = cos(alpha)/sin(alpha);
 cosa2     = 2.0*cos(alpha);
 SSY       = SHY/sin(alpha);
 inv_SSX   = 1.0/SSX;
 inv_SHY   = 1.0/SHY;

 lj_safety = skin; /* [old] max(dum,volvar/3.0); */

        fclose(file_1);

        cell_size_z = ljcut_max + lj_safety;
        number_of_cells_z = ceil((1+(lpm-1)*bond_max)/cell_size_z+ 5.);
        system_size_z = number_of_cells_z * cell_size_z ;

        number_of_cells_x  = floor(SSX*sin(alpha) / (ljcut_max+lj_safety));
        CSX = 1.0 / number_of_cells_x;
        number_of_cells_y  = floor(SHY            / (ljcut_max+lj_safety));
        CSY = 1.0 / number_of_cells_y;

}

/* ------------------------------------------------------------------ */
/* Sortiert die periodischen Koordinaten aller Monomere in die zug.
   Zellen ein                                                          */

void setup_pointers()
{
        int ii,ix,iy,iz;

        for (ii=0; ii<ntot; ii++)
          {
                ix = itrunc(xbox[ii]*number_of_cells_x);
                iy = itrunc(ybox[ii]*number_of_cells_y);
                iz = itrunc(zmm[ii]/cell_size_z);

                /* Fuegt das Monomer ii in die Zelle [ix,iy,iz] ein    */
                addp2cell(ii,ix,iy,iz);                    /*  <MC.c>  */
```

```
                xcell[ii] = ix;
                ycell[ii] = iy;
                zcell[ii] = iz;
        }
}

/* ------------------------------------------------------------------- */
/* Start mit orthorhombischen Kopfgitter und senkrechten Ketten        */

void start_from_ortho()                                /* Aufruf  "o" */
{
    int indx,jndx,kndx,nside;
    double distx,disty;

    nside = (int) ceil(sqrt((double) npm));
    distx = (double) SSX /nside;
    disty = (double) SHY /nside;
    /* setup coordinates     */

    /* Aufsetzen in rechteckiger (!) Box der Groesse SSX*SHY           */
    /* und nachtraegliche Scherung auf gewuenschten Winkel alpha       */

    for (indx = 0; (indx < nside) && (indx*nside < npm); indx++)
      {
        xmm[indx*lpm*nside] = distx*indx;
        ymm[indx*lpm*nside] = 0.5*disty;  /* [Haas] ymm[indx*lpm*nside] = 0.5; *
/
        zmm[indx*lpm*nside] = 1.;
        for (kndx =indx*lpm*nside+1; kndx < (indx*nside+1)*lpm; kndx++)
          {
            xmm[kndx]=xmm[kndx-1];
            ymm[kndx]=ymm[kndx-1];
            zmm[kndx]=zmm[kndx-1]+bond_length;
          }
        for (jndx = indx*nside+1; (jndx < (indx+1)*nside)&&(jndx < npm); jndx++)

          {
            xmm[jndx*lpm]  = xmm[(jndx-1)*lpm] + distx/2.;
            ymm[jndx*lpm]  = ymm[(jndx-1)*lpm] + disty;
            zmm[jndx*lpm]  = 1.;
            for (kndx =jndx*lpm+1; kndx < (jndx +1)*lpm; kndx++)
              {
                xmm[kndx]=xmm[kndx-1];
                ymm[kndx]=ymm[kndx-1];
                zmm[kndx]=zmm[kndx-1]+bond_length;
              }
          }
      }

    /* Auswertung der periodischen, gescherten Relativkoordinaten     */
    /* Diese werden von der folgenden Scherung nicht veraendert       */

    for ( indx=0; indx<ntot; indx++)
      {
        xbox[indx] = fmod(xmm[indx]/SSX +32.0, 1.0);
        ybox[indx] = fmod(ymm[indx]/SHY +32.0, 1.0);
      }

    /* Scherung der kartesischen Absolutkoordinaten auf Winkel alpha  */

    for ( indx=0; indx<ntot; indx++)
      {
```



```
        xmm[indx] += ctg_alpha*ymm[indx];
      }

    /* Einordnen der Monomere in die Zellen                            */
    setup_pointers();
}

/* ------------------------------------------------------------------- */
/* Start mit deutlich gescherter Box und senkrechten Ketten            */

void start_from_shear_ortho()                          /* Aufruf  "O" */
{
    int indx,jndx,kndx,nside;
    double distx,disty;

    nside = (int) ceil(sqrt((double) npm));
    distx = (double) 1.0 /nside;
    disty = (double) 1.0 /nside;

    /* Aufsetzen in gescherter Box, zuerst in Relativkoordinaten       */

    for (indx = 0; (indx < nside) && (indx*nside < npm); indx++)
      {
        xbox[indx*lpm*nside] = distx*(indx+0.5);
        ybox[indx*lpm*nside] = disty*0.5;
        zmm[indx*lpm*nside] = 1.;
        for (kndx =indx*lpm*nside+1; kndx < (indx*nside+1)*lpm; kndx++)
          {
            xbox[kndx]=xbox[kndx-1];
            ybox[kndx]=ybox[kndx-1];
            zmm[kndx]=zmm[kndx-1]+bond_length;
          }
        for (jndx = indx*nside+1; (jndx < (indx+1)*nside)&&(jndx < npm); jndx++)

          {
            xbox[jndx*lpm] = xbox[(jndx-1)*lpm];
            ybox[jndx*lpm] = ybox[(jndx-1)*lpm] + disty;
            zmm[jndx*lpm]  = 1.;
            for (kndx =jndx*lpm+1; kndx < (jndx +1)*lpm; kndx++)
              {
                xbox[kndx]=xbox[kndx-1];
                ybox[kndx]=ybox[kndx-1];
                zmm[kndx]=zmm[kndx-1]+bond_length;
              }
          }
      }

    /* Auswertung der kartesischen Absolutkoordinaten                  */
    /* NEU ! auch alle xmm,ymm sind jetzt in der Box !                 */

    for ( indx=0; indx<ntot; indx++)
      {
        ymm[indx] = SHY*ybox[indx];
        xmm[indx] = SSX*xbox[indx] + ctg_alpha*ymm[indx];
      }

    /* Einordnen der Monomere in die Zellen                            */
    setup_pointers();
}

/* ------------------------------------------------------------------- */
/* Start aus naeherungsweisem Grundzustand NNN-geneigt                 */
/* LJ-Koepfe und Softcore-Koepfe < 1.08 sind im GZ in NNN-geneigt      */
```

```c
void start_from_ground_nnn()                              /* Aufruf  "g" */
{
    int indx,jndx,kndx,nside;
    double distx,disty;
    double bond_length,thetagr,phigr;

 /* Grundzustandswerte fuer Bondabstand, Neigungswinkel und -richtung  */
 /* unter der Annahme gestreckter Ketten aber mit Beruecksichtigung    */
 /* der LJ-WW innerhalb einer Kette                                    */

    if (fabs(ljmin_hh-ljmin)<0.001)
      { /* Parameter fuer LJ-Koepfe und BW=100.0                       */

        /* Werte wurden von [H. Lange] in seiner Diplomarbeit durch    */
        /* Energieminimierung des Systems bestimmt.                    */
        /* (keine einfache analytische Angabe "1/sqrt(3)..." moeglich) */

        bond_length = 0.6259995;    /* mit starrer Bond-Laenge = 0.700 */
        thetagr     = 0.3653694596; /* mit starrer Bond-Laenge = 0.615 */
        phigr       = 0.0;
      }
    else
      { /* Parameter fuer Softcore [MM=1.0,HM=1.05,HH=1.10] BW=100.0   */

        bond_length = 0.6528;       /* mit starrer Bond-Laenge = 0.700 */
        thetagr     = 0.4322;       /* mit starrer Bond-Laenge = 0.615 */
        phigr       = 0.0;
      }

    nside = (int) ceil(sqrt((double) npm));
    distx = (double) SSX / nside;
    disty = (double) SHY / nside;

    /* Aufsetzen in rechteckiger (!) Box der Groesse SSX*SHY           */
    /* und nachtraegliche Scherung auf gewuenschten Winkel alpha       */

    for (indx = 0; (indx < nside) && (indx*nside < npm); indx++)
    {
      /* Setzen der Kopfgruppen, unterste Reihe aequidistant           */
      /* Indizes {0 *7*12; 1 *7*12; ... ; 11 *7*12}                    */

      xmm[indx*lpm*nside] = distx*indx;
      ymm[indx*lpm*nside] = disty*0.5;  /* [Haas] ymm[indx*lpm*nside] = 0.5; */
      zmm[indx*lpm*nside] = 1.;

      /* Aufbauen der Monomere der ersten (geneigten) Ketten-Reihe    */

      for (kndx =indx*lpm*nside+1; kndx < (indx*nside+1)*lpm; kndx++)
        {
          xmm[kndx]=xmm[kndx-1]+bond_length*sin(thetagr)*sin(phigr);
          ymm[kndx]=ymm[kndx-1]+bond_length*sin(thetagr)*cos(phigr);
          zmm[kndx]=zmm[kndx-1]+bond_length*cos(thetagr);
        }

      for (jndx = indx*nside+1; jndx < (indx+1)*nside && jndx < npm; jndx++)
       {
     /* restliche Reihen der KOPF-Monomere                                */
     /* Aufbau in einem hexagonalen Gitter : nicht senkrecht uebereinander, */
     /* sondern um distx/2 versetzt !   X X X           X X X            */
     /*                                X X X   anstatt  X X X            */
     /*                               X X X             X X X            */
        xmm[jndx*lpm]  = xmm[(jndx-1)*lpm] + distx/2.;
```



```c
        ymm[jndx*lpm]  = ymm[(jndx-1)*lpm] + disty;
        zmm[jndx*lpm]  = 1.;

        /* (geneigter) Aufbau der darueberliegenden Monomerschichten       */

        for (kndx =jndx*lpm+1; kndx < (jndx +1)*lpm; kndx++)
           {
             xmm[kndx]=xmm[kndx-1]+bond_length*sin(thetagr)*sin(phigr) ;
             ymm[kndx]=ymm[kndx-1]+bond_length*sin(thetagr)*cos(phigr) ;
             zmm[kndx]=zmm[kndx-1]+bond_length*cos(thetagr) ;
           }
       } /* next jndx */
    } /* next indx */

    /* Auswertung der periodischen, gescherten Relativkoordinaten     */
    /* Diese werden von der folgenden Scherung nicht veraendert       */

    for ( indx=0; indx<ntot; indx++)
      {
        xbox[indx] = fmod(xmm[indx]/SSX +32.0, 1.0);
        ybox[indx] = fmod(ymm[indx]/SHY +32.0, 1.0);
      }

    /* Scherung der kartesischen Absolutkoordinaten auf Winkel alpha  */

    for ( indx=0; indx<ntot; indx++)
      {
        xmm[indx] += ctg_alpha*ymm[indx];
      }

     /* Einordnen der Monomere in die Zellen                           */
        setup_pointers();
}
/* ------------------------------------------------------------------- */
/* Start aus naeherungsweisem Grundzustand NNN-geneigt                 */
/* LJ-Koepfe und Softcore-Koepfe < 1.08 sind im GZ in NNN-geneigt      */

void start_from_shear_nnn(idir_start)                    /* Aufruf  "s" */
int idir_start;
{ /* NNN-Richtung ist 6-fach entartet, idir_start waehlt aus :         */
  /* Richtung in Grad = 30 + idir_start * 60                            */

    int indx,jndx,nside,ii,jj;
    double distx,disty,x_aim,y_aim;
    double KSI_x,KSI_y,ETA_x,ETA_y;
    double bond_length,thetagr,phigr;
    double fx,fy,fz;

 /* Grundzustandswerte fuer Bondabstand, Neigungswinkel und -richtung  */
 /* unter der Annahme gestreckter Ketten aber mit Beruecksichtigung    */
 /* der LJ-WW innerhalb einer Kette                                    */

    if (fabs(ljmin_hh-ljmin)<0.001)
      { /* Parameter fuer LJ-Koepfe und BW=100.0                       */

        /* Werte wurden von [H. Lange] in seiner Diplomarbeit durch    */
        /* Energieminimierung des Systems bestimmt.                    */
        /* (keine einfache analytische Angabe "1/sqrt(3)..." moeglich) */

        bond_length = 0.6259995;    /* mit starrer Bond-Laenge = 0.700 */
        thetagr     = 0.3653694596; /* mit starrer Bond-Laenge = 0.615 */
      }
    else
```

```
      { /* Parameter fuer Softcore [MM=1.0,HM=1.05,HH=1.10] BW=100.0   */

        bond_length = 0.6528;       /* mit starrer Bond-Laenge = 0.700 */
        thetagr     = 0.4322;       /* mit starrer Bond-Laenge = 0.615 */
      }

    nside = (int) ceil(sqrt((double) npm));
    distx = (double) 1.0 / nside;
    disty = (double) 1.0 / nside;

    /* Kopfgruppengitter in periodischen Koordinaten                    */

    for (indx = 0; (indx < nside) && (indx*nside < npm); indx++)
      {
        xbox[indx*lpm*nside] = distx*(indx+0.5);
        ybox[indx*lpm*nside] = disty*0.5;
        zmm[indx*lpm*nside] = 1.;

        for (jndx = indx*nside+1; (jndx < (indx+1)*nside)&&(jndx < npm); jndx++)

          {
            xbox[jndx*lpm] = xbox[(jndx-1)*lpm];
            ybox[jndx*lpm] = ybox[(jndx-1)*lpm] + disty;
            zmm[jndx*lpm]  = 1.;
          }
      }

    /* Auswertung der kartesischen Absolutkoordinaten der Koepfe      */

    for ( jndx=0; jndx<npm; jndx++)
      { indx = lpm*jndx;
        ymm[indx] = SHY*ybox[indx];
        xmm[indx] = SSX*xbox[indx] + ctg_alpha*ymm[indx];
      }

    /* Ausrichtung der Ketten auf nn-Nachbarn                           */

    /* Basisvektoren des gescherten Gitters                             */
    KSI_x = xmm[lpm*nside] - xmm[0];   ETA_x = xmm[lpm*1] - xmm[0];
    KSI_y = ymm[lpm*nside] - ymm[0];   ETA_y = ymm[lpm*1] - ymm[0];

    if (idir_start==0) { x_aim = 1.0*KSI_x + 1.0*ETA_x;   y_aim = 1.0*KSI_y + 1.
0*ETA_y; }
    if (idir_start==1) { x_aim =-1.0*KSI_x + 2.0*ETA_x;   y_aim =-1.0*KSI_y + 2.
0*ETA_y; }
    if (idir_start==2) { x_aim =-2.0*KSI_x + 1.0*ETA_x;   y_aim =-2.0*KSI_y + 1.
0*ETA_y; }
    if (idir_start==3) { x_aim =-1.0*KSI_x - 1.0*ETA_x;   y_aim =-1.0*KSI_y - 1.
0*ETA_y; }
    if (idir_start==4) { x_aim = 1.0*KSI_x - 2.0*ETA_x;   y_aim = 1.0*KSI_y - 2.
0*ETA_y; }
    if (idir_start==5) { x_aim = 2.0*KSI_x - 1.0*ETA_x;   y_aim = 2.0*KSI_y - 1.
0*ETA_y; }

    phigr = atan2(y_aim,x_aim);

    fx = bond_length*sin(thetagr)*cos(phigr);
    fy = bond_length*sin(thetagr)*sin(phigr);
    fz = bond_length*cos(thetagr);

    /* Aufbau der Ketten auf ihren Koepfen in kart. Absolutkoord.      */

    for (ii=0; ii<npm; ii++)
```



```
      { indx = ii*lpm;
        for (jj=1; jj<lpm; jj++)
        { jndx = indx+jj;
          xmm[jndx]=xmm[indx] + fx*jj;
          ymm[jndx]=ymm[indx] + fy*jj;
          zmm[jndx]=zmm[indx] + fz*jj;
        }
      }

    for (ii=0; ii<npm; ii++)
      { indx = ii*lpm;
        for (jj=1; jj<lpm; jj++)
        { jndx = indx+jj;
          xbox[jndx] = fmod((xmm[jndx]-ctg_alpha*ymm[jndx])/SSX +32.0, 1.0);
          ybox[jndx] = fmod(ymm[jndx]/SHY +32.0, 1.0);

        }
      }

     /* Einordnen der Monomere in die Zellen                            */
        setup_pointers();
}
/* ------------------------------------------------------------------- */
/* Start aus naeherungsweisem Grundzustand NN-geneigt                  */
/* Softcore-Koepfe > 1.08 sind im GZ in Richtung NN geneigt !          */

void start_from_ground_nn()                            /* Aufruf  "G" */
{
    int indx,jndx,kndx,nside;
    double distx,disty;
    double bond_length,thetagr,phigr;

 /* Grundzustandswerte fuer Bondabstand, Neigungswinkel und -richtung  */
 /* unter der Annahme gestreckter Ketten aber mit Beruecksichtigung    */
 /* der LJ-WW innerhalb einer Kette                                    */

    /* Parameter fuer MM=1.0, HH=1.10, HM=1.05, BW=100.0 BL=10.0 */

    bond_length = 0.65;   /* mit reiner Bond-Laengen-WW  = 0.700 */
    thetagr     = 0.47;   /* mit reiner Bond-Laengen-WW  = 0.615 */
    phigr       = M_PI_2;

    nside = (int) ceil(sqrt((double) npm));
    distx = (double) SSX / nside;
    disty = (double) SHY / nside;

    /* Aufsetzen in rechteckiger (!) Box der Groesse SSX*SHY           */
    /* und nachtraegliche Scherung auf gewuenschten Winkel alpha       */

    for (indx = 0; (indx < nside) && (indx*nside < npm); indx++)
    {
   /* Setzen der Kopfgruppen, unterste Reihe aequidistant
      Indizes {0 *7*12; 1 *7*12; ... ; 11 *7*12}                        */

      xmm[indx*lpm*nside] = distx*indx;
      ymm[indx*lpm*nside] = disty*0.5;  /* [Haas] ymm[indx*lpm*nside] = 0.5; */
      zmm[indx*lpm*nside] = 1.;

  /* Aufbauen der Monomere der ersten (geneigten) Ketten-Reihe          */

     for (kndx =indx*lpm*nside+1; kndx < (indx*nside+1)*lpm; kndx++)
        {
          xmm[kndx]=xmm[kndx-1]+bond_length*sin(thetagr)*sin(phigr);
```

```
          ymm[kndx]=ymm[kndx-1]+bond_length*sin(thetagr)*cos(phigr);
          zmm[kndx]=zmm[kndx-1]+bond_length*cos(thetagr);
        }

      for (jndx = indx*nside+1; jndx < (indx+1)*nside && jndx < npm; jndx++)
        {
     /* restliche Reihen der KOPF-Monomere                                  */
     /* Aufbau in einem hexagonalen Gitter : nicht senkrecht uebereinander, */
     /* sondern um distx/2 versetzt !   X X X           X X X               */
     /*                                  X X X   anstatt  X X X               */
     /*                                X X X             X X X               */

        xmm[jndx*lpm]  = xmm[(jndx-1)*lpm] + distx/2.;
        ymm[jndx*lpm]  = ymm[(jndx-1)*lpm] + disty;
        zmm[jndx*lpm]  = 1.;

     /* (geneigter) Aufbau der darueberliegenden Monomerschichten           */

        for (kndx =jndx*lpm+1; kndx < (jndx +1)*lpm; kndx++)
           {
             xmm[kndx]=xmm[kndx-1]+bond_length*sin(thetagr)*sin(phigr) ;
             ymm[kndx]=ymm[kndx-1]+bond_length*sin(thetagr)*cos(phigr) ;
             zmm[kndx]=zmm[kndx-1]+bond_length*cos(thetagr) ;
           }

      } /* next jndx */
    } /* next indx */

    /* Auswertung der periodischen, gescherten Relativkoordinaten     */
    /* Diese werden von der folgenden Scherung nicht veraendert       */

    for ( indx=0; indx<ntot; indx++)
      {
        xbox[indx] = fmod(xmm[indx]/SSX +32.0, 1.0);
        ybox[indx] = fmod(ymm[indx]/SHY +32.0, 1.0);
      }

    /* Scherung der kartesischen Absolutkoordinaten auf Winkel alpha  */

    for ( indx=0; indx<ntot; indx++)
      {
        xmm[indx] += ctg_alpha*ymm[indx];
      }

     /* Einordnen der Monomere in die Zellen                           */
        setup_pointers();
}
/* ------------------------------------------------------------------- */
/* Start aus naeherungsweisem Grundzustand NN-geneigt                  */
/* Softcore-Koepfe > 1.08 sind im GZ in Richtung NN geneigt !          */

void start_from_shear_nn(idir_start)                    /* Aufruf  "S" */
int idir_start;
{ /* NN-Richtung ist 6-fach entartet, idir_start waehlt aus :          */
  /* Richtung in Grad = idir_start * 60                                */
    int indx,jndx,nside,ii,jj;
    double distx,disty,x_aim,y_aim;
    double KSI_x,KSI_y,ETA_x,ETA_y;
    double bond_length,thetagr,phigr;
    double fx,fy,fz;

 /* Grundzustandswerte fuer Bondabstand, Neigungswinkel und -richtung  */
 /* unter der Annahme gestreckter Ketten aber mit Beruecksichtigung    */
```



```
 /* der LJ-WW innerhalb einer Kette                                    */

    bond_length = 0.65;   /* mit reiner Bond-Laengen-WW  = 0.700 */
    thetagr     = 0.47;   /* mit reiner Bond-Laengen-WW  = 0.615 */

    nside = (int) ceil(sqrt((double) npm));
    distx = (double) 1.0 / nside;
    disty = (double) 1.0 / nside;

    /* Kopfgruppengitter in periodischen Koordinaten                   */

    for (indx = 0; (indx < nside) && (indx*nside < npm); indx++)
      {
        xbox[indx*lpm*nside] = distx*(indx+0.5);
        ybox[indx*lpm*nside] = disty*0.5;
        zmm[indx*lpm*nside] = 1.;

        for (jndx = indx*nside+1; (jndx < (indx+1)*nside)&&(jndx < npm); jndx++)

          {
            xbox[jndx*lpm] = xbox[(jndx-1)*lpm];
            ybox[jndx*lpm] = ybox[(jndx-1)*lpm] + disty;
            zmm[jndx*lpm]  = 1.;
          }
      }

    /* Auswertung der kartesischen Absolutkoordinaten der Koepfe      */

    for ( jndx=0; jndx<npm; jndx++)
      { indx = lpm*jndx;
        ymm[indx] = SHY*ybox[indx];
        xmm[indx] = SSX*xbox[indx] + ctg_alpha*ymm[indx];
      }

    /* Ausrichtung der Kette auf  n-Nachbarn                           */

    /* Basisvektoren des gescherten Gitters                            */
    KSI_x = xmm[lpm*nside] - xmm[0];   ETA_x = xmm[lpm*1] - xmm[0];
    KSI_y = ymm[lpm*nside] - ymm[0];   ETA_y = ymm[lpm*1] - ymm[0];

    if (idir_start==0) { x_aim = 1.0*KSI_x + 0.0*ETA_x;   y_aim = 1.0*KSI_y + 0.
0*ETA_y; }
    if (idir_start==1) { x_aim = 0.0*KSI_x + 1.0*ETA_x;   y_aim = 0.0*KSI_y + 1.
0*ETA_y; }
    if (idir_start==2) { x_aim =-1.0*KSI_x + 1.0*ETA_x;   y_aim =-1.0*KSI_y + 1.
0*ETA_y; }
    if (idir_start==3) { x_aim =-1.0*KSI_x - 0.0*ETA_x;   y_aim =-1.0*KSI_y - 0.
0*ETA_y; }
    if (idir_start==4) { x_aim = 0.0*KSI_x - 1.0*ETA_x;   y_aim = 0.0*KSI_y - 1.
0*ETA_y; }
    if (idir_start==5) { x_aim = 1.0*KSI_x - 1.0*ETA_x;   y_aim = 1.0*KSI_y - 1.
0*ETA_y; }

    phigr = atan2(y_aim,x_aim);

    fx = bond_length*sin(thetagr)*cos(phigr);
    fy = bond_length*sin(thetagr)*sin(phigr);
    fz = bond_length*cos(thetagr);

    /* Aufbau der Ketten auf ihren Koepfen in kart. Absolutkoord.      */

    for (ii=0; ii<npm; ii++)
      { indx = ii*lpm;
```

MC_s.c 15/21

```
        for (jj=1; jj<lpm; jj++)
        { jndx = indx+jj;
          xmm[jndx]=xmm[indx] + fx*jj;
          ymm[jndx]=ymm[indx] + fy*jj;
          zmm[jndx]=zmm[indx] + fz*jj;
        }
      }

    for (ii=0; ii<npm; ii++)
      { indx = ii*lpm;
        for (jj=1; jj<lpm; jj++)
        { jndx = indx+jj;
          xbox[jndx] = fmod((xmm[jndx]-ctg_alpha*ymm[jndx])/SSX +32.0, 1.0);
          ybox[jndx] = fmod(ymm[jndx]/SHY +32.0, 1.0);

        }
      }

     /* Einordnen der Monomere in die Zellen                          */
        setup_pointers();
}

/* ------------------------------------------------------------------ */
/* Start aus existierender Konfigurations-Datei                       */
/* 1. Lesen des Alters der Konfiguration und
      der Koordinaten der Monomere
   2. Setzen der Pointer                                              */

void start_from_config(int ifile)                        /* Aufruf  "c" */
{
        int ii;
        FILE *file_1;
        char zeile[160],fname[20];

     /* Regelfall : ifile = 1 --> Endung _01.cnf  entspr.
                                  "final configuration"-File           */

        sprintf(fname,"%s_%02d.cnf",swtch[1],ifile);
        file_1=fopen(fname,"r");
        if (file_1 == NULL) printf("ERROR opening %s\n",fname);

        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);
        fgets(zeile,160,file_1);

        for (ii=0; ii < ntot;ii++)
          {
            fgets(zeile,160,file_1);
            sscanf(zeile,"%lf %lf %lf %lf %lf",
                         &xmm[ii],&ymm[ii],&zmm[ii],&xbox[ii],&ybox[ii]);
          }
```

MC_s.c 16/21

```
        fclose(file_1);


     /* Einordnen der Monomere in die Zellen                          */
        printf(" > start_from_config    : pointers ...");
        if (ifile==1)
           { setup_pointers(); printf("all set\n"); }
}


/* ------------------------------------------------------------------ */
/* Tabellarisierung des Lennard-Jones-Potentials in LJ_grid Schritten  */
/* Schritte in r^2 --> unguenstig wegen Potentialverlaufs !            */
/*                                                                     */
/* Da LJ-Potential immer nur in der Form min(enrg,LJ_clip) ausgewertet */
/* wurde ist es schneller, es gleich so abzuspeichern !                */

void initialize_LJ_table()
{
        double dist6,delta,dist,enrg;
        int ii;

     /* LJ-Potential zwischen normalen Monomeren                 */

        delta = ljcut_quadrat/(double) LJ_grid;
        delta = delta / (ljmin*ljmin);

        for (ii=0;ii<LJ_grid;ii++)
           {    dist  = (ii+1)*delta;
                dist6 = dist*dist*dist;
                enrg  = u0*(-lj0-(2./dist6)+(1./(dist6*dist6)));
                LJ[ii] =min(enrg,LJ_clip);
           }
        LJ[LJ_grid]=0.;

     /* LJ-Potential zwischen Kopfmonomeren                      */

        delta = ljcut_hh_quadrat/(double) LJ_hh_grid;
        delta = delta / (ljmin_hh*ljmin_hh);

        for (ii=0;ii<LJ_hh_grid;ii++)
           {    dist  = (ii+1)*delta;
                dist6 = dist*dist*dist;
                enrg  = u0_hh*(-lj0_hh-(2./dist6)+(1./(dist6*dist6)));
                LJ_hh[ii] = min(enrg,LJ_clip);
           }
        LJ_hh[LJ_hh_grid]=0.;

     /* LJ-Potential zwischen Kopf und normalem Monomer          */

        delta = ljcut_hm_quadrat/(double) LJ_hm_grid;
        delta = delta / (ljmin_hm*ljmin_hm);

        for (ii=0;ii<LJ_hm_grid;ii++)
           {    dist  = (ii+1)*delta;
                dist6 = dist*dist*dist;
                enrg  = u0_hm*(-lj0_hm-(2./dist6)+(1./(dist6*dist6)));
                LJ_hm[ii] = min(enrg,LJ_clip);
           }
        LJ_hm[LJ_hm_grid]=0.;

  pLJ[0] = LJ;    ljcq[0] = ljcut_quadrat;    ljsc[0] = lj_scala;
```

```
  pLJ[1] = LJ_hm; ljcq[1] = ljcut_hm_quadrat; ljsc[1] = lj_hm_scala;
  pLJ[2] = LJ_hh; ljcq[2] = ljcut_hh_quadrat; ljsc[2] = lj_hh_scala;

  lju0[0] = u0;    ljmq[0] = ljmin*ljmin;
  lju0[1] = u0_hm; ljmq[1] = ljmin_hm*ljmin_hm;
  lju0[2] = u0_hh; ljmq[2] = ljmin_hh*ljmin_hh;
}

/* ------------------------------------------------------------------- */
/* Auswertung aller Bondwinkel-Potentiale der bestehenden Konfigurat.  */

void initialize_ang_energies(copo)
double *copo;
{
        int i1,i2,i3;
        double dist1,dist2,x1,x2,y1,y2,z1,z2;

        for (i1=0; i1<ntot; i1++)
           { copo[i1]=0.0; }  /*  auf Null vorbesetzen */

/* copo[i] : energie of bond angle between bonds connecting monomers
               (i-1 and i) and (i and i+1)                          */

        for (i1=0; i1<npm; i1++){     /* Schleife ueber alle Ketten     */
           for (i2=1; i2<lpm-1; i2++){/* Schleife ueber innere Monomere */
             i3 = i1*lpm+i2;
             x1= xmm[i3]- xmm[i3-1];
             y1= ymm[i3]- ymm[i3-1]; /* Vektor nach unten */
             z1= zmm[i3]- zmm[i3-1];
             x2= xmm[i3+1]- xmm[i3];
             y2= ymm[i3+1]- ymm[i3]; /* Vektor nach oben  */
             z2= zmm[i3+1]- zmm[i3];
             dist1 = x1*x1 + y1*y1 + z1*z1;
             dist2 = x2*x2 + y2*y2 + z2*z2;

          /* a_vec * b_vec  =  |a| * |b| * cos(a,b)                  */
          /* Bond-Winkelpotential = cospr * ( 1-cos(Bondwinkel) )     */
          /* wird im copo-Array jeweils dem Knick-Monomer zugeordnet  */

             copo[i3]   = cospr*(1-(x1*x2+y1*y2+z1*z2)/sqrt(dist1*dist2));
           }
        }
}

/* ------------------------------------------------------------------- */
/* Auswertung aller Bondpotentiale der Ketten   einer best. Konfig.    */
/* und Aufruf zur Auswertung der LJ-Potentiale                         */
/* Diese Version wird nur beim Start der Simulation aufgerufen.        */
/* Hier werden die WW-Tafeln WW1 und WW2 gleichgesetzt !               */
/* Die Volumen-Moves benutzen vm_initialize_energy.                    */

void initialize_energy(egy)
double *egy;
{
        int i1,i2,i3,i4,ip,jp,hit0;
        double dx,dy,dz,rdist,e_tmp;

 /* Typisierung der LJ-WWs  (nur einmal, bei Programmstart)              */

 for (i1=0; i1<npm; i1++)
    for (i2=0; i2<lpm; i2++)
      { ip = lpm*i1 + i2;
        for (i3=i1; i3<npm; i3++)
```



```
          for (i4=0; i4<lpm; i4++)
            { jp = lpm*i3 + i4;
              if ((i2==0)&&(i4==0))
                 lj_type[ip][jp] = lj_type[jp][ip] = c_2;    /* HH */
              else
                 if ((i2>0)&&(i4>0))
                    lj_type[ip][jp] = lj_type[jp][ip] = c_0; /* MM */
                 else
                    lj_type[ip][jp] = lj_type[jp][ip] = c_1; /* HM */
            }
      }

        for (i1=0; i1<ntot; i1++) egy[i1]=0.0;

        for (i1=0; i1<ntot; i1++)        /* Schleife ueber alle Monomere    */
           { hit0 = i1%lpm;              /* Modulo --> Position im Monomer  */

             if (hit0<lpm-1)              /* >>> alle ausser Endmonomer <<< */
               { dx    = xmm[i1+1]-xmm[i1]; /* Abstd. zum Bondnachbarn aus  */
                 dy    = ymm[i1+1]-ymm[i1]; /* nicht-period. Koordianten !  */
                 dz    = zmm[i1+1]-zmm[i1];
                 rdist = sqrt(dx*dx+dy*dy+dz*dz);
                 if ((rdist>bond_max)||(rdist<bond_min))
                   { e_tmp = LJ_clip; }
                 else
                   { e_tmp = bond_laengen_potential(rdist); }
                 *WW1_in[i1] = e_tmp;
                 egy[i1] += e_tmp;  egy[i1+1] += e_tmp;
               }

           } /* next i1 , next Monomer */

        /* Auswertung aller LJ-WW zwischen allen Monomeren,                */
        /* Speicherung in WW-Tafel WW1,                                    */
        /* Summation der Monomer-Gesamtenergien egy, incl. Bondlaengen     */

        calc_all_lj(egy);                                   /*  <MC_e.c>  */

        /* Vorgefundene Energien nach WW2 uebernehmen                      */
        for (i1=0; i1<WW_max; i1++)
          { WW2[i1] = WW1[i1]; }

  return;
}


/* ------------------------------------------------------------------- */
/* vollst. Neuberechnung der Verlet-Tabelle aus den linked-cells unter */
/* Ausnutzung der paarweisen Symmetrie, vgl. calc_all_lj               */

void verlet_update()

{ char tmp[WW_triangle];
  char *tmp_in[MAX_MONOMERS],*tmp_ptr;
  int ix,iy,iz,ip,jp,kp;
  int ii,jj,kk,x_array[14],y_array[14],z_array[14];
  struct part *CL,*CLa;
  double x,y,z;
  double dist,distx,disty,dx,dy,dz;
  int count_lower[MAX_MONOMERS],*end_ptr[MAX_MONOMERS];

  verlet_update_counter++;
```

MC_s.c 19/21

```
d_diff_max = d_diff_max2 = 0.0;
skin_verlet= lj_safety;
rad_verlet = (ljcut_max + skin_verlet)*(ljcut_max + skin_verlet);

/* Monomerposition zum Zeitpunkt des Verlet-Updates, um nachher aus */
/* der Drift die Notwendigkeit eines neuen Updates zu bestimmen     */
for (ii=0; ii<ntot; ii++)
  { x0ver[ii] = xmm[ii];  y0ver[ii] = ymm[ii];  z0ver[ii] = zmm[ii]; }

/* 0-Element enthaelt Zahl der Partner in der Kugel, wird auf Null  */
/* vorbesetzt                                                       */
for (ii=0; ii<ntot; ii++)
  { tab_verlet[ii][0] = 0; }

/* Logisches Hilfsfeld der Paare mit Abstand<rad_verlet vorbesetzen */
kk=0;
for (ii=0; ii<ntot; ii++)
  { tmp_in[ii] = &tmp[kk];  kk = kk + ntot-1 - ii; }
for (ii=0; ii<WW_triangle; ii++)
  { tmp[ii] = c_0; }

for (ix=0; ix<number_of_cells_x; ix++)
 for (iy=0; iy<number_of_cells_y; iy++)
  for (iz=0; iz<number_of_cells_z; iz++)
    { CLa = cell[ix][iy][iz];

      /* die Nachbarzellen in den anderen Richtungen werden */
      /* im Schleifenverlauf beruecksichtigt !              */
      x_array[ 0] = ix;        y_array[ 0] = iy;       z_array[ 0] = iz;
      x_array[ 1] = ix;        y_array[ 1] = iy;       z_array[ 1] = jpz[iz];
      x_array[ 2] = ix;        y_array[ 2] = jpy[iy];  z_array[ 2] = jpz[iz];
      x_array[ 3] = ix;        y_array[ 3] = jpy[iy];  z_array[ 3] = iz;
      x_array[ 4] = ix;        y_array[ 4] = jpy[iy];  z_array[ 4] = jmz[iz];
      x_array[ 5] = jpx[ix];   y_array[ 5] = jmy[iy];  z_array[ 5] = jpz[iz];
      x_array[ 6] = jpx[ix];   y_array[ 6] = iy;       z_array[ 6] = jpz[iz];
      x_array[ 7] = jpx[ix];   y_array[ 7] = jpy[iy];  z_array[ 7] = jpz[iz];
      x_array[ 8] = jpx[ix];   y_array[ 8] = jmy[iy];  z_array[ 8] = iz;
      x_array[ 9] = jpx[ix];   y_array[ 9] = iy;       z_array[ 9] = iz;
      x_array[10] = jpx[ix];   y_array[10] = jpy[iy];  z_array[10] = iz;
      x_array[11] = jpx[ix];   y_array[11] = jmy[iy];  z_array[11] = jmz[iz];
      x_array[12] = jpx[ix];   y_array[12] = iy;       z_array[12] = jmz[iz];
      x_array[13] = jpx[ix];   y_array[13] = jpy[iy];  z_array[13] = jmz[iz];

      for ( ;  CLa!=NULL; CLa=CLa->next)
        { /* ......aeussere Schleife ueber alle Monomere...... */
          ip      = CLa->pn;
          tmp_ptr = tmp_in[ip];
          x = xbox[ip];
          y = ybox[ip];
          z = zmm[ip];

          { ii=0;

              CL = cell[x_array[ii]][y_array[ii]][z_array[ii]];

              for ( ; CL!=NULL; CL=CL->next)
                { /* ...innere Schleife ueber alle Partner... */
                  jp = CL->pn;
                  if (jp!=ip)
                    { /* nur falls wirklich LJ-WW moeglich    */

                      /* Abstandsbestimmung im period. System */
                      distx = xbox[jp]-x;
```

MC_s.c 20/21

```
                      dx    = SSX*(distx-floor(distx+0.5));
                      disty = ybox[jp]-y;
                      dy    = SSY*(disty-floor(disty+0.5));
                      dz    = zmm[jp]-z;

                      dist  = dx*dx + dy*dy + cosa2*dx*dy + dz*dz;

                      if (dist<rad_verlet)
                        { kp = jp-ip;
                          if (kp>0)
                            { tmp_ptr[kp-1]     = c_1; }
                          else
                            { tmp_in[jp][-kp-1] = c_1; }
                        }
                    }
                } /* ..................................... */
            } /* ii=0 */


          for (ii=1; ii<14; ii++)
            { CL = cell[x_array[ii]][y_array[ii]][z_array[ii]];

              for ( ; CL!=NULL; CL=CL->next)
                { /* ...innere Schleife ueber alle Partner... */
                  jp = CL->pn;

                      /* Abstandsbestimmung im period. System */
                      distx = xbox[jp]-x;
                      dx    = SSX*(distx-floor(distx+0.5));
                      disty = ybox[jp]-y;
                      dy    = SSY*(disty-floor(disty+0.5));
                      dz    = zmm[jp]-z;

                      dist  = dx*dx + dy*dy + cosa2*dx*dy + dz*dz;

                      if (dist<rad_verlet)
                        { kp = jp-ip;
                          if (kp>0)
                            { tmp_ptr[kp-1]     = c_1; }
                          else
                            { tmp_in[jp][-kp-1] = c_1; }
                        }
                    } /* ..................................... */
            }
        }
    }

/* Nachtraegliche Entfernung der Bondnachbarn                  */
/* Bei hohen Temperaturen koennen Schwanz von i-1 und          */
/* Kopf von i durchaus wechselwirken !                         */
for (ii=0; ii<ntot; ii++)
  if ((ii%lpm)!=(lpm-1)) tmp_in[ii][0] = c_0;

/* In jeder Verlet-Kugel kommen zuerst die Monomere mit        */
/* hoeherem Index danach die mit niedrigerem Index als das     */
/* Monomer in der Kugelmitte                                   */
kk=0;
for (ii=0; ii<ntot; ii++)
  { kp = 0;
    for (jj=ii+1; jj<ntot; jj++)
      { if (tmp[kk]) { kp++; tab_verlet[ii][kp]=jj; }
        kk++;
      }
```

```
      upper_verlet[ii]=kp;
    }

  for (ii=0; ii<ntot; ii++)
    { count_lower[ii] = 0;
      end_ptr[ii]     = &tab_verlet[ii][1+upper_verlet[ii]];
    }

  for (ii=0; ii<ntot; ii++)
    { for (jj=1; jj<=upper_verlet[ii]; jj++)
        { kk=tab_verlet[ii][jj];
          *end_ptr[kk]=ii;
          end_ptr[kk]++; count_lower[kk]++;
        }
    }

  for (ii=0; ii<ntot; ii++)
    { tab_verlet[ii][0] = upper_verlet[ii] + count_lower[ii]; }

}
```

MC_e.c 1/5

```c
/* ================================================================ */
/* Enthaelt Funktionen die direkt mit dem LJ-Potential zu tun haben  */
/* ================================================================ */

#include "switchboard.h"

#include <malloc.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "MC.h"


/* ---------------------------------------------------------------- */
/* berechnet LJ-Energie Monomer-Monomer                              */
double se_mono(ip,x,y,z,ljcut_quad,lj_scal,L)
int ip;                        /* Nummer des zu vergleichenden Monomers */
double x,y,z;              /* period. Koord. des zu untersuchenden Monos. */
double ljcut_quad,lj_scal,*L;
{
        double dist,distx,disty,dx,dy,dz;
        int idist;

     /* Abstandsbestimmung im periodischen relativen System           */

        distx = xbox[ip]-x;
        if (fabs(distx)<=0.5) { dx = SSX*distx; }
        else if (distx>0.0)   { dx = (distx-1.0)*SSX; }
             else             { dx = (distx+1.0)*SSX; }
        disty = ybox[ip]-y;
        if (fabs(disty)<=0.5) { dy = SSY*disty; }
        else if (disty>0.0)   { dy = (disty-1.0)*SSY; }
             else             { dy = (disty+1.0)*SSY; }

        dz    = zmm[ip]-z;

        dist  = dx*dx + dy*dy + cosa2*dx*dy + dz*dz;

        /* Abst. > LJ-Abschneideradius ?                              */
        if (dist > ljcut_quad) return(0.);

        /* auf naechsten Rasterpunkt ! keine weitere Interpolation     */
        idist = (int) (dist*lj_scal);
        return(L[idist]);
}

/* ---------------------------------------------------------------- */
/* Wird nur noch fuer normale Monomer-Moves benutzt !                */
/* routine returns interaction energy of monomer ip having trial     */
/* coordinates x,y,z with all particles in the verlet table          */
/* x,y,z    : periodische,relative Monomerkoordinaten                */
/*                                                                   */
/* WARNUNG  : Da durch tab_verlet die Indizes aller WW-Partner bekannt */
/*            sind werden die Energien mit den anderen Monomeren NICHT */
/*            mehr genullt ! (inkompatibel zu aelteren Versionen)    */

double calculate_energy(ip,x,y,z)
int ip;
double x,y,z;
{
  int i1,i2,i3;
  double enrgy;
```

MC_e.c 2/5

```c
 enrgy = 0.0;

 /* die Verlet-Tabelle enthaelt nur die Monomere mit denen wirklich  */
 /* LJ-WW moeglich, also weder ip selbst, noch seine Bondnachbarn !  */

 for (i1=1; i1<=tab_verlet[ip][0]; i1++)
   { i2 = tab_verlet[ip][i1];
     i3 = (int) lj_type[ip][i2];
     enrgy += ( engy[i2] =  se_mono(i2,x,y,z,ljcq[i3],ljsc[i3],pLJ[i3]) );
   }

 return(enrgy);
}

/* ---------------------------------------------------------------- */
/* Auswertung aller LJ-WW zwischen allen Monomeren nach Verlet-Tab.  */
/* beim Simulationsstart                                             */
/* Speicherung in WW-Tafel WW1,                                      */
/* Summation der Monomer-Gesamtenergien egy, incl. Bondlaengen       */
/* diese sind beim Aufruf bereits in WW1 eingetragen                 */

void calc_all_lj(e_field)
double *e_field;

{ int jj,ip,jp,kp;
  double *WW1_ptr;
  double x,y,z,e_tmp;

  for (ip=0; ip<ntot-1; ip++)
  { /* ........aeussere Schleife ueber alle Monomere........ */
    WW1_ptr = WW1_in[ip];
    x = xbox[ip];
    y = ybox[ip];
    z = zmm[ip];

    for (jj=1; jj<=upper_verlet[ip]; jj++)
      { jp = tab_verlet[ip][jj];
        kp = (int) lj_type[ip][jp];
        /* Die Paare mit jp<ip wurden schon ausgewertet !    */
        e_tmp = se_mono(jp,x,y,z,ljcq[kp],ljsc[kp],pLJ[kp]);
        WW1_ptr[jp-ip-1] = e_tmp;
        e_field[ip] += e_tmp;
        e_field[jp] += e_tmp;
      }
  } /* ............................................... */

}

#ifndef VH_Flag

/* ---------------------------------------------------------------- */
/* berechnet LJ-Energie Monomer-Monomer bei einem kont. Volumen-Move  */
double svm_e_mono(ip,x,y,z,new_ssx,new_ssy,new_cosa2,ljcut_quad,lj_scal,L)
int ip;
double x,y,z,new_ssx,new_ssy,new_cosa2;
double ljcut_quad,lj_scal,*L;
{
        double dist,distx,disty,dx,dy,dz;
        int idist;

     /* Abstandsbestimmung mit gescherten periodischen Randbegingungen */
        distx = xbox[ip]-x;
        if (fabs(distx)<=0.5) { dx = new_ssx*distx; }
```

```
        else if (distx>0.0)   { dx = (distx-1.0)*new_ssx; }
             else             { dx = (distx+1.0)*new_ssx; }
        disty = ybox[ip]-y;
        if (fabs(disty)<=0.5) { dy = new_ssy*disty; }
        else if (disty>0.0)   { dy = (disty-1.0)*new_ssy; }
             else             { dy = (disty+1.0)*new_ssy; }
        dz    = zmm[ip]-z;

        dist  = dx*dx + dy*dy + new_cosa2*dx*dy +dz*dz;

        if (dist > ljcut_quad) return(0.0);
        /* auf naechsten Rasterpunkt ! keine weitere Interpolation     */
        idist = (int) (dist*lj_scal);

        return(L[idist]);
}


/* ------------------------------------------------------------------- */
/* Version fuer kontinuierliche Volumen-Moves                          */
/* Auswertung aller LJ-WW zwischen allen Monomeren,                    */
/* Speicherung in WW-Tafel WW2,                                        */
/* Summation der Monomer-Gesamtenergien egy, incl. Bondlaengen         */

void svm_calc_all_lj(e_field,new_ssx,new_ssy,new_cosa2)
double *e_field,new_ssx,new_ssy,new_cosa2;
{
  int ip,jp,kp;
  double *WW2_ptr;
  int jj;
  double x,y,z,e_tmp;

  for (ip=0; ip<ntot-1; ip++)
  { /* ......aeussere Schleife ueber alle Monomere....... */
    WW2_ptr = WW2_in[ip];
    x = xbox[ip];
    y = ybox[ip];
    z = zmm[ip];

    for (jj=1; jj<=upper_verlet[ip]; jj++)
      { jp = tab_verlet[ip][jj];
        kp = (char) lj_type[ip][jp];
        /* Die Paare mit jp<ip wurden schon ausgewertet !    */
        e_tmp = svm_e_mono(jp,x,y,z,new_ssx,new_ssy,new_cosa2,
                           ljcq[kp],ljsc[kp],pLJ[kp]);
        WW2_ptr[jp-ip-1] = e_tmp;
        e_field[ip] += e_tmp;
        e_field[jp] += e_tmp;
      }

  } /* ................................................ */

}

/* ------------------------------------------------------------------- */
/* Auswertung aller Bondwinkel-Potentiale der bestehenden Konfigurat.  */
/* Wird nur von kontinuierlichen Volumen-Moves aufgerufen.             */

void svm_initialize_ang_energies(copo)
double *copo;
{
        int i1,i2,i3;
        double ubx,uby,ubz,ubd;
```



```
        double lbx,lby,lbz,lbd;

        for (i1=0; i1<ntot; i1++)
           { copo[i1]=0.; }                /* alle auf Null vorbesetzen  */

        for (i1=0; i1<npm; i1++)          /* Schleife ueber alle Ketten  */
          { i3 = i1*lpm;
            lbx = xtrans[i3+1]-xtrans[i3];  /* unterer Bond              */
            lby = ytrans[i3+1]-ytrans[i3];
            lbz = zmm[i3+1]-zmm[i3];
            lbd = sqrt(lbx*lbx + lby*lby + lbz*lbz);

            for (i2=1; i2<lpm-1; i2++) /* Schleife ueber innere Monomere */
              { i3 = i1*lpm+i2;
                ubx = xtrans[i3+1] - xtrans[i3];
                uby = ytrans[i3+1] - ytrans[i3]; /* oberer Bond          */
                ubz = zmm[i3+1] - zmm[i3];
                ubd = sqrt(ubx*ubx + uby*uby + ubz*ubz);

               /* a_vec * b_vec  =  |a| * |b| * cos(a,b)                 */
               /* Bond-Winkelpotential = cospr * ( 1-cos(Bondwinkel) )    */
               /* wird im copo-Array jeweils dem Knick-Monomer zugeordnet */

                copo[i3]=cospr*(1-(lbx*ubx + lby*uby + lbz*ubz)/(lbd*ubd));

                lbx = ubx;  lby = uby;  lbz = ubz;  lbd = ubd;
              }
        }
        return;
}

/* ------------------------------------------------------------------- */
/* Auswertung aller Bondpotentiale der Ketten   einer best. Konfig.    */
/* und Aufruf zur Auswertung der LJ-Potentiale                         */
/* Diese Version wird bei kontinuierlichen Volumen-Moves aufgerufen.   */
/* Hier werden die WWs in Tafel WW2 geschrieben, falls der Move akzep- */
/* tiert wird, werden nur die Pointer auf WW1 und WW2 vertauscht.      */

void svm_initialize_energy(new_ssx,new_ssy,new_cosa2,egy)
double new_ssx,new_ssy,new_cosa2,*egy;
{
  int i1,hit0;
  double dx,dy,dz,rdist,e_tmp;

  for (i1=0; i1<ntot; i1++) egy[i1]=0.0;

  for (i1=0; i1<ntot; i1++)       /* Schleife ueber alle    Monomere */
    { hit0 = i1%lpm;              /* Modulo --> Position im Monomer  */

      if (hit0<lpm-1)                   /* >>> alle ausser Endmonomer <<< */
        { dx    = xtrans[i1+1]-xtrans[i1]; /* Abstd. zum Bondnachbarn aus  */
          dy    = ytrans[i1+1]-ytrans[i1]; /* nicht-period. Koordinaten !  */
          dz    = zmm[i1+1]-zmm[i1];
          rdist = sqrt(dx*dx + dy*dy + dz*dz);
          if ((rdist>bond_max)||(rdist<bond_min))
            { e_tmp = LJ_clip; }
          else
            { e_tmp = bond_laengen_potential(rdist); }
          *WW2_in[i1] = e_tmp;
          egy[i1] += e_tmp;  egy[i1+1] += e_tmp;
        }

    } /* next i1 , next Monomer */
```

MC_e.c **5/5**

```
  /* Auswertung aller LJ-WW zwischen allen Monomeren,                */
  /* Speicherung in WW-Tafel WW2,                                    */
  /* Summation der Monomer-Gesamtenergien egy, incl. Bondlaengen     */

  svm_calc_all_lj(egy,new_ssx,new_ssy,new_cosa2);

  return;
}

#else

#include "vh_e.c"  /* falls Volumen-Moves in beide Richtungen gewollt  */

#endif
```

```c
/* Abweichend von der Version [Haas,Lange] wird jeder Kette ein axialer   */
/* Vektor (xax,yax,zax) zugeordnet und Neigungswinkel und -richtung       */
/* werden daraus zentral berechnet und in ncth,nphi gespeichert.          */
/*                                                                        */
/* Verwendet man als axialen Vektor einfach den Ende-Kopf-Vektor der      */
/* Ketten   erhaelt man das alte Ergebnis.                                */
/* Alternativ wird hier die "kleinste" Haupt-Traegheitsachse der Ketten   */
/* als Axial-Vektor benutzt.                                              */
/*                                                                        */
/* Die Routinen zur Hauptachsen-Transformation stammen aus                */
/* [Press, Flannery, Teukolsky, Vetterling]                               */
/* Numerical Recipes. The Art of Scientific Computing                     */

#include "switchboard.h"

#include <malloc.h>
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <stdlib.h>
#include "r250.h"
#include "matrix.h"
#include "MC.h"
#include "poly_axis.h"


void speriod_dist_min(pdx,pdy,dx,dy,inv_ssx,inv_shy,ctg_alfa)
double *pdx,*pdy,dx,dy,inv_ssx,inv_shy,ctg_alfa;
{
  double d_eta,d_ksi;

  d_eta = (dx-ctg_alfa*dy)*inv_ssx;  /* gescherte Relativkoordinaten */
  d_ksi =  dy             *inv_shy;

  d_eta = d_eta-floor(d_eta+0.5);
  d_ksi = d_ksi-floor(d_ksi+0.5);

  *pdy = SHY*d_ksi;
  *pdx = SSX*d_eta + ctg_alfa*SHY*d_ksi;
}

/* ------------------------------------------------------------------- */
/* Speicherplatz fuer die zum Messen benoetigten Pointer belegen       */

void allocate_msr_memory()
{
  int    hsize;
  size_t s_npm,s_npm2;

  s_npm   = (size_t) npm;
  s_npm2  = (size_t) (2*npm);

     /* allocation of memory for measurement arrays   */

        xn2t = (double *) calloc(s_npm,sizeof(double)); /* entsprechende    */
        yn2t = (double *) calloc(s_npm,sizeof(double)); /* Momentan-        */
        zn2t = (double *) calloc(s_npm,sizeof(double)); /* koordinaten      */
        xendt= (double *) calloc(s_npm2,sizeof(double));
        yendt= (double *) calloc(s_npm2,sizeof(double));
        zendt= (double *) calloc(s_npm2,sizeof(double));

        xax = (double *) calloc(s_npm,sizeof(double)); /* Axialer Vektor */
        yax = (double *) calloc(s_npm,sizeof(double)); /* des Kette      */
```



```c
        zax = (double *) calloc(s_npm,sizeof(double));

        ncth = (double *) calloc(s_npm,sizeof(double)); /* cos(Neigungswinkel)*/
        nphi = (double *) calloc(s_npm,sizeof(double)); /* Neigungsrichtung   */
 hsize = ceil(20.*bond_max*(lpm-1));

        histoxy = (unsigned long *)
                  calloc((size_t) (hsize*hsize),sizeof(unsigned long));

        histoz = (unsigned long *)
                  calloc((size_t) (lpm*hsize),sizeof(unsigned long));

        histophi = (unsigned long *)          /* 800 Kaestchen fuer 360 Grad */
                  calloc((size_t) 800,sizeof(unsigned long)); /* N.-Richtung */

        histothe = (unsigned long *)          /* 100 Kaestchen fuer  90 Grad */
                  calloc((size_t) 100,sizeof(unsigned long)); /* N.-Winkel   */

        histodst = (unsigned long *)
                  calloc((size_t) 1001,sizeof(unsigned long));

        historpvf = (unsigned long *)
                  calloc((size_t) 1001*3,sizeof(unsigned long));

        if (xn2t == NULL) printf("ALLOCATION ERROR in msr \n");
        if (yn2t == NULL) printf("ALLOCATION ERROR in msr \n");
        if (zn2t == NULL) printf("ALLOCATION ERROR in msr \n");
        if (xendt == NULL) printf("ALLOCATION ERROR in msr \n");
        if (yendt == NULL) printf("ALLOCATION ERROR in msr \n");
        if (zendt == NULL) printf("ALLOCATION ERROR in msr \n");
        if (xax == NULL) printf("ALLOCATION ERROR in msr \n");
        if (yax == NULL) printf("ALLOCATION ERROR in msr \n");
        if (zax == NULL) printf("ALLOCATION ERROR in msr \n");
        if (ncth == NULL) printf("ALLOCATION ERROR in msr \n");
        if (nphi == NULL) printf("ALLOCATION ERROR in msr \n");
        if (histoxy == NULL) printf("ALLOCATION ERROR in msr \n");
        if (histoz == NULL) printf("ALLOCATION ERROR in msr \n");
        if (histophi == NULL) printf("ALLOCATION ERROR in msr \n");
        if (histothe == NULL) printf("ALLOCATION ERROR in msr \n");
        if (histodst == NULL) printf("ALLOCATION ERROR in msr \n");
        if (historpvf == NULL) printf("ALLOCATION ERROR in msr \n");
}

/* ------------------------------------------------------------------- */
/* Schreibt den Kopf der Messungsdateien 1..13 analog zur Konfigurations-
   datei *.cnf  [die wichtigsten Parameter der Simulation]              */

void write_msr_header(f)
FILE *f;
{
   fprintf(f,"DATA FOR %s CREATED BY MC.exe\n",swtch[1]);
   fprintf(f,"----------------------------\n");
   fprintf(f,"dim       npm       lpm       mcsmax    temp      pressure\n");
   fprintf(f,"%-9d %-9d %-9d %-9ld %-9.5f %-9.5f\n",dim,npm,lpm,(long)mcsmax,tem
p,pressure);
   fprintf(f,"bond_min  bond_max  hvar      volvar  alphvar\n");
   fprintf(f,"%-9.3f %-9.3f %-9.5f %-9.5f %-9.5f\n",bond_min,bond_max,hvar,volva
r,alphvar_io);
   fprintf(f,"u0        sprng     cospr     ljmin     ljcut\n");
   fprintf(f,"%-9.3f %-9.3f %-9.3f %-9.3f %-9.3f\n"
            ,u0,sprng,cospr,ljmin,ljcut);
   fprintf(f,"u0_hh     ljmin_hh  ljcut_hh\n");
   fprintf(f,"%-9.3f %-9.3f %-9.3f\n"
```

MC_m.c 3/27

```
            ,u0_hh,ljmin_hh,ljcut_hh);
   fprintf(f,"u0_hm     ljmin_hm  ljcut_hm\n");
   fprintf(f,"%-9.3f %-9.3f %-9.3f\n"
            ,u0_hm,ljmin_hm,ljcut_hm);
   fprintf(f,"%c : headgroups fixed (0) or movable (1) \n",heads);
   fprintf(f," \n");
   fprintf(f," \n");
}

/* ------------------------------------------------------------------- */
/* Histogramm-Zaehler histoz, histoxy, histophi, histothe, histodst
   und historpvf durch Nullung initialisieren                          */

void initialize_histo()
{
    int i1,i2,hsize;

    hsize = ceil(20.*bond_max*(lpm-1));

    for (i1=0; i1<hsize; i1++)
       {
         for (i2=0; i2<lpm;   i2++) histoz[i1+i2*hsize]=0;
         for (i2=0; i2<hsize; i2++) histoxy[i1+i2*hsize]=0;
    }

    for(i1=0;i1<800;i1++) histophi[i1]=0;
    for(i1=0;i1<100;i1++) histothe[i1]=0;
    for(i1=0;i1<1000;i1++) histodst[i1]=0;

    for (i1=0; i1<3; i1++)
        for (i2=0; i2<=1000; i2++) historpvf[1001*i1+i2]=0;

}

/* ------------------------------------------------------------------- */
/* histographische Behandlung der Ketten-Achsen                        */
/* und der vertikalen Schichtung der Monomere                          */

void add_to_histo()
{
    int ii,jj,kk,jj_prt,kk_prt,imx,imy,imz,hsize,ibin,ii_offset;
    double mx,my,mz,x,y,distx,disty,dist,dx,dy,
           xcm_j,xcm_k,xct_j,xct_k,
           ycm_j,ycm_k,yct_j,yct_k,cut_plane,rel;

    /* bond_max*(lpm-1) ist nicht nur die maximale Laenge einer Kette  */
    /* sondern auch Maximalwert der Komponenten des Kopf-End-Vektors   */
    hsize = ceil(20.*bond_max*(lpm-1));
    /* Schleife ueber alle Ketten                                      */
    for (ii=0; ii<npm; ii++)
       { mx = xendt[2*ii] - xendt[2*ii+1]; /* von Ende nach Kopf         */
         my = yendt[2*ii] - yendt[2*ii+1];
         mz = zendt[2*ii+1] - zendt[2*ii]; /* von Kopf nach Ende !      */

         /* mx,my in [-bond_max*(lpm-1);+bond_max*(lpm-1)]              */
         /* Aufloesung in xy-Ebene : 10 Kaestchen/Bond_Laenge           */
         imx = ceil(10*(mx + bond_max*(lpm-1)));
         imy = ceil(10*(my + bond_max*(lpm-1)));
         /* 1-dim Index in der xy-Ebene                                 */
         histoxy[imx+imy*hsize]++;

         /* Kopfgruppen sowieso in z=1-Ebene fixiert, stattdessen       */
         /* Histogramm der Ketten-Hoehen !                              */
```

MC_m.c 4/27

```
         /* mz in [0;bond_max*(lpm-1)]                                  */
         imz = ceil(20*mz);
         histoz[imz]++;

         for (jj=1; jj<lpm; jj++)
            { mz = zmm[jj+ii*lpm] -1.;
              imz = ceil(20*mz);
              /* Offset 'jj*hsize' trennt die Monomer-Schichten         */
              histoz[imz+jj*hsize]++;
            }
       }

    /* ..................................................................... */
    /* radiale Paarverteilungsfunktion der Koepfe (2-dim)                  */
    ii_offset = 0;

    for (jj=0; jj<npm; jj++)
      {
        jj_prt = lpm*jj;
        x = xbox[jj_prt];  y = ybox[jj_prt];

        for (kk=jj+1; kk<npm; kk++)
          { kk_prt = lpm*kk;

            distx = xbox[kk_prt]-x;
            dx    = SSX*(distx-floor(distx+0.5));
            disty = ybox[kk_prt]-y;
            dy    = SSY*(disty-floor(disty+0.5));
            dist  = sqrt(dx*dx + dy*dy + cosa2*dx*dy);

            /* Kasten-System : 0 < r < 10,   100 Bins / LE  */

           ibin = min(1000,itrunc(100.0*dist));
           historpvf[ii_offset+ibin]++;
          }
      }

    /* ..................................................................... */
    /* radiale Paarverteilungsfunktion der Schwerpunkte (2-dim)            */
    ii_offset = 1001;

    for (jj=0; jj<npm; jj++)
      {
        jj_prt = lpm*jj;
        xcm_j = ycm_j = 0.0;

        for (ii=0; ii<lpm; ii++)
          { xcm_j += xmm[jj_prt+ii];  ycm_j += ymm[jj_prt+ii]; }

        xcm_j /= (double) lpm;  ycm_j /= (double) lpm;

        for (kk=jj+1; kk<npm; kk++)
          {
            kk_prt = lpm*kk;
            xcm_k = ycm_k = 0.0;

            for (ii=0; ii<lpm; ii++)
              { xcm_k += xmm[kk_prt+ii];  ycm_k += ymm[kk_prt+ii]; }

            xcm_k /= (double) lpm;  ycm_k /= (double) lpm;

            speriod_dist_min(&distx, &disty,
                             xcm_k-xcm_j, ycm_k-ycm_j,
```

```
                              inv_SSX, inv_SHY, ctg_alpha);
           dist  = sqrt(distx*distx + disty*disty);

           /* Kasten-System : 0 < r < 10,   100 Bins / LE  */

          ibin = min(1000,itrunc(100.0*dist));
          historpvf[ii_offset+ibin]++;
        }
      }

   /* ............................................................ */
   /* radiale Paarverteilungsfunktion der Schwerpunkte (2-dim)        */
   ii_offset = 2002;

 cut_plane = 3.0;

   for (jj=0; jj<npm; jj++)
     {
       jj_prt = lpm*jj;

       /* Annahme : In jeder Kette nur ein Schnittpunkt mit der Ebene */
       for (ii=1; ii<lpm; ii++)
           if (zmm[jj_prt+ii] > cut_plane) break;

       if (ii<lpm)
         { rel = (cut_plane      - zmm[jj_prt+ii-1]) /
                 (zmm[jj_prt+ii] - zmm[jj_prt+ii-1]);

           xct_j = rel * xmm[jj_prt+ii]  +  (1.0-rel) * xmm[jj_prt+ii-1];
           yct_j = rel * ymm[jj_prt+ii]  +  (1.0-rel) * ymm[jj_prt+ii-1];

           for (kk=jj+1; kk<npm; kk++)
             {
               kk_prt = lpm*kk;

               for (ii=1; ii<lpm; ii++)
                  if (zmm[kk_prt+ii] > cut_plane) break;

               if (ii<lpm)
                  { rel = (cut_plane      - zmm[kk_prt+ii-1]) /
                          (zmm[kk_prt+ii] - zmm[kk_prt+ii-1]);

                    xct_k = rel * xmm[kk_prt+ii]  +  (1.0-rel) * xmm[kk_prt+ii-
1];
                    yct_k = rel * ymm[kk_prt+ii]  +  (1.0-rel) * ymm[kk_prt+ii-
1];

                    speriod_dist_min(&distx, &disty,
                                     xct_k-xct_j, yct_k-yct_j,
                                     inv_SSX, inv_SHY, ctg_alpha);
                    dist  = sqrt(distx*distx + disty*disty);

                    /* Kasten-System : 0 < r < 10,   100 Bins / LE  */

                    ibin = min(1000,itrunc(100.0*dist));
                    historpvf[ii_offset+ibin]++;
                  }
             }
         }
     }
}

/* ---------------------------------------------------------------- */
```



```
/* Alte Histogramme einlesen und fortschreiben                        */

void read_histo(int *isamp)
{
    FILE *f1;
    char fname[30],zeile[160];
    int i,jj,ix,iy,iz,hsize,err_cnt,idummy;
    double x,y;

    /* in FORTRAN wuerde eine kleine implizite DO-Schleife genuegen... */
    unsigned long u[14];

    err_cnt = 0;

/* ............................................................ */

    sprintf(fname,"%s_histxy.dat",swtch[1]);
    f1 = fopen(fname,"r");
    if (f1 == NULL)
      { printf("ERROR opening %s\n",fname); err_cnt++; }
    else
      { for(i=0; i<15; i++) fgets(zeile,80,f1);

        fgets(zeile,160,f1);
        sscanf(zeile,"%d\n",&idummy);

        hsize = ceil(20.*bond_max*(lpm-1));
        for (ix=0; ix<hsize; ix++)
           for (iy=0; iy<hsize; iy++)
            { fgets(zeile,160,f1);
              sscanf(zeile,"%lf %lf %lu\n",&x,&y,&histoxy[ix+iy*hsize]);
            }
        fclose(f1);
      }

    *isamp = idummy;
/* ............................................................ */

    sprintf(fname,"%s_histz.dat",swtch[1]);
    f1 = fopen(fname,"r");
    if (f1 == NULL)
      { printf("ERROR opening %s\n",fname); err_cnt++; }
    else
      { for(i=0; i<16; i++) fgets(zeile,80,f1);

        hsize = ceil(20.*bond_max*(lpm-1));
        for (iz=0; iz<hsize; iz++)
         { fgets(zeile,160,f1);
           idummy = sscanf(zeile,
                     "%lf %lu %lu %lu %lu %lu %lu %lu %lu %lu %lu %lu %lu %l
u",
                     &x,&u[0],&u[1],&u[2],&u[3],&u[4],&u[5],&u[6],&u[7],
                     &u[8],&u[9],&u[10],&u[11],&u[12],&u[13]);
           if (idummy!=lpm+1) { err_cnt++; continue; }

           for (jj=0; jj<lpm; jj++) histoz[iz+jj*hsize] = u[jj];
         }

        fclose(f1);
      }
/* ............................................................ */

    sprintf(fname,"%s_histang.dat",swtch[1]);
```

MC_m.c 7/27

```
    f1 = fopen(fname,"r");
    if (f1 == NULL)
      { printf("ERROR opening %s\n",fname); err_cnt++; }
    else
      { for(i=0; i<16; i++) fgets(zeile,80,f1);

        for (jj=0; jj<100; jj++)
          { fgets(zeile,160,f1);

            sscanf(zeile,"%lf %lu %lu %lu %lu %lu %lu %lu %lf %lu",
                   &x,&histophi[jj+  0],&histophi[jj+100],&histophi[jj+200],
                   &histophi[jj+300],&histophi[jj+400],&histophi[jj+500],
                   &histophi[jj+600],&histophi[jj+700],&y,&histothe[jj]);
          }

        fclose(f1);
      }
/* ..................................................................... */

    sprintf(fname,"%s_histdst.dat",swtch[1]);
    f1 = fopen(fname,"r");
    if (f1 == NULL)
      { printf("ERROR opening %s\n",fname); err_cnt++; }
    else
      { for(i=0; i<16; i++) fgets(zeile,80,f1);

        for (jj=0; jj<1000; jj++)
          { fgets(zeile,160,f1);
            sscanf(zeile,"%lf %lu",&x,&histodst[jj]);
          }
        fclose(f1);
      }

/* ..................................................................... */

    sprintf(fname,"%s_histrpvf.dat",swtch[1]);
    f1 = fopen(fname,"r");
    if (f1 == NULL)
      { printf("ERROR opening %s\n",fname); err_cnt++; }
    else
      { for(i=0; i<16; i++) fgets(zeile,80,f1);

        for (jj=0; jj<1001; jj++)
          { fgets(zeile,160,f1);
            sscanf(zeile,"%lu %lu %lu %lu",&u[0],
                   &historpvf[jj],&historpvf[1001+jj],&historpvf[2002+jj]);
          }
        fclose(f1);
      }

    /* Fehler beim Einlesen der alten Histogramme, dann eben neue      */
    /* berechnen ...                                                   */
    if (err_cnt>0)
      { printf("Fehler beim Lesen der alten Histogramme !\n");
        fflush(stdout);
        *isamp = 0;
        initialize_histo();
      }
}

/* ------------------------------------------------------------------- */
/* Histogramm der xy-Ebene der  Ketten-Achsen                          */
```

MC_m.c 8/27

```
void print_histoxy(int isam)
{
    FILE *f1;
    char fname[20];
    int ix,iy,hsize;
    double x,y;

    sprintf(fname,"%s_histxy.dat",swtch[1]);
    f1 = fopen(fname,"w");                          /*  <-- append !    */
    write_msr_header(f1);

    fprintf(f1,"%d : number of samples\n",isam);
    hsize = ceil(20.*bond_max*(lpm-1));
    for (ix=0; ix<hsize; ix++)
       { for (iy=0; iy<hsize; iy++)
            { /* (x|y) Koordinaten des Histo-Kaestchens [ix+iy*hsize]  */
              x = 0.1*(ix - ceil(10.*bond_max*(lpm-1)));
              y = 0.1*(iy - ceil(10.*bond_max*(lpm-1)));
              fprintf(f1,"%f %f %lu\n",x,y,histoxy[ix+iy*hsize]);
            }
       }
    fclose(f1);
}

/* ------------------------------------------------------------------- */
/* Histogramm der  Ketten-Achsen-Hoehen und der vertikalen Schichtung  */

void print_histoz(int isam)
{
    FILE *f1;
    char fname[20];
    int jj,iz,hsize;
    double z;

    sprintf(fname,"%s_histz.dat",swtch[1]);
    f1 = fopen(fname,"w");
    write_msr_header(f1);

    fprintf(f1,"%d : number of samples\n",isam);
    hsize = ceil(20.*bond_max*(lpm-1));
    for (iz=0; iz<hsize; iz++)
       {  z = 0.05*iz;
          fprintf(f1,"%f ",z);
          for (jj=0; jj<lpm; jj++)
             { fprintf(f1,"%lu ",histoz[iz+jj*hsize]); }
          fprintf(f1,"\n");
       }
    fclose(f1);
}

/* ------------------------------------------------------------------- */
/* Histogramme der Neigungswinkel, -richtung und Bondwinkel            */
/* --->  MEANphi                                                       */

void print_histoang(int isam)
{
    FILE *f1;
    char fname[20];
    int jj,kk;
    double phi;

    sprintf(fname,"%s_histang.dat",swtch[1]);
    f1 = fopen(fname,"w");
```

```c
    write_msr_header(f1);

    fprintf(f1,"%d : number of samples\n",isam);
    for (jj=0; jj<100; jj++)
       { phi=jj*0.01*2.*M_PI;
         fprintf(f1,"%f ",phi);
         for (kk=0; kk<8; kk++)
            { fprintf(f1,"%lu ",histophi[jj+100*kk]); }
         phi=jj*0.01*M_PI/2.;
         fprintf(f1,"%f ",phi);
         fprintf(f1,"%lu ",histothe[jj]);
         fprintf(f1,"\n");
       }
    fclose(f1);
}

/* ------------------------------------------------------------------- */
/* Histogramm der Entfernung zu den 6 naechsten Nachbarn               */

void print_histodst(int isam)
{
    FILE *f1;
    char fname[20];
    int jj;
    double dst;

    sprintf(fname,"%s_histdst.dat",swtch[1]);
    f1 = fopen(fname,"w");
    write_msr_header(f1);

    fprintf(f1,"%d : number of samples\n",isam);
    for(jj=0;jj<1000;jj++){
       dst = jj*0.01;
       fprintf(f1,"%f %lu\n",dst,histodst[jj]);
       }
    fclose(f1);
}

/* ------------------------------------------------------------------- */
/* Histogramm zu den radialen Paarverteilungsfunktionen                */

void print_historpvf(int isam)
{
    FILE *f1;
    char fname[20];
    int jj;

    sprintf(fname,"%s_histrpvf.dat",swtch[1]);
    f1 = fopen(fname,"w");
    write_msr_header(f1);

    fprintf(f1,"%d : number of samples\n",isam);

    for (jj=0; jj<=1000; jj++)
        fprintf(f1,"%u %lu %lu %lu\n",jj,
                historpvf[jj],historpvf[1001+jj],historpvf[2002+jj]);

    fclose(f1);
}

/* ------------------------------------------------------------------- */
/* Initialisiert die .dat-Files, in die bei jedem Messungs-Schritt die */
/* zu bestimmenden Messwerte fortgeschrieben werden.                   */
```



```c
void prepare_msr_files()
{
        char fname[20];

    FILE *f1,*f2,*f3,*f4,*f5,*f6,*f8,*f9,*f14,*f15;

        sprintf(fname,"%s_b.dat",swtch[1]); /* <"Biegung">             */
     f1 = fopen(fname,"w");
        sprintf(fname,"%s_u.dat",swtch[1]); /* Energien : U, ...,H     */
     f2 = fopen(fname,"w");
        sprintf(fname,"%s_v.dat",swtch[1]); /* Box-Groesse, Fl.-Dichte */
     f3 = fopen(fname,"w");
        sprintf(fname,"%s_a.dat",swtch[1]); /* Akzeptanz-Raten         */
     f4 = fopen(fname,"w");
        sprintf(fname,"%s_p.dat",swtch[1]); /* <Ketten -Achsen>        */
     f5 = fopen(fname,"w");
        sprintf(fname,"%s_o.dat",swtch[1]); /* <R_xy, O>               */
     f14= fopen(fname,"w");
        sprintf(fname,"%s_t.dat",swtch[1]); /* <Neigung> "tilt"        */
     f6 = fopen(fname,"w");
        sprintf(fname,"%s_n.dat",swtch[1]); /* Phi_6 -Neigungsrichtung */
     f15= fopen(fname,"w");

    if (dim == 3)
      { sprintf(fname,"%s_l.dat",swtch[1]); /* Kopfgruppen-Gitter      */
        f8 = fopen(fname,"w");
        sprintf(fname,"%s_sprpr.dat",swtch[1]); /* Spreizdruck         */
        f9 = fopen(fname,"w");
      }

    write_msr_header(f1);
              fclose(f1);
    write_msr_header(f2);
              fclose(f2);
    write_msr_header(f3);
              fclose(f3);
    write_msr_header(f4);
              fclose(f4);
    write_msr_header(f5);
              fclose(f5);
    write_msr_header(f14);
              fclose(f14);
    write_msr_header(f6);
              fclose(f6);
    write_msr_header(f15);
              fclose(f15);

    if (dim == 3)
      { write_msr_header(f8);
                  fclose(f8);
        write_msr_header(f9);
                  fclose(f9);
      }
}

/* ------------------------------------------------------------------- */
/* Bestimmung der momentanen Lagen der Mittel-, Kopf- und End-Monomere */
/* und der Schwerpunkte der Monomere                                   */

void measure_all()
{       double lax;
        int n2,ii;
```

```
        n2 = lpm/2;

        for (ii=0; ii<npm; ii++){
                xendt[2*ii+1]   = xmm[ii*lpm+lpm-1];
                yendt[2*ii+1]   = ymm[ii*lpm+lpm-1]; /* der End-Monomere  */
                zendt[2*ii+1]   = zmm[ii*lpm+lpm-1];
        }

        for (ii=0; ii<npm; ii++){
                xendt[2*ii]     = xmm[ii*lpm];
                yendt[2*ii]     = ymm[ii*lpm];      /* der Kopf-Monomere  */
                zendt[2*ii]     = zmm[ii*lpm];
           }

        for (ii=0; ii<npm; ii++){              /* momentane Koordinaten : */
                xn2t[ii]        = xmm[ii*lpm+n2];
                yn2t[ii]        = ymm[ii*lpm+n2];  /* der mittleren Monos */
                zn2t[ii]        = zmm[ii*lpm+n2];
           }

  /* Bestimmung der  Ketten-Achsen                                    */
  for (ii=0; ii<npm; ii++)
    { poly_axis_eval(ii,lpm); }                       /* <poly_axis.c> */

  /* Haupttraegheitsrichtung mit unbest. Vorzeichen,  waehle z>0,     */
  /* sonst Problem bei Winkelbestimmung (theta > pi/2 sonst)          */
  for (ii=0; ii<npm; ii++)
    if (zax[ii]<0)
      { xax[ii]=-xax[ii]; yax[ii]=-yax[ii]; zax[ii]=-zax[ii]; }

  /* Alternativ : [Haas,Lange]-Version : End-Kopf-Vektor              */
  /* for (ii=0; ii<npm; ii++)
    { xax[ii] = xmm[ii*lpm+lpm-1] - xmm[ii*lpm];
      yax[ii] = ymm[ii*lpm+lpm-1] - ymm[ii*lpm];
      zax[ii] = zmm[ii*lpm+lpm-1] - zmm[ii*lpm];
    } */

  /* Bestimmung der Neigungswinkel und -richtungen                     */
  for (ii=0; ii<npm; ii++)
    { lax = sqrt(xax[ii]*xax[ii] + yax[ii]*yax[ii] + zax[ii]*zax[ii]);
      ncth[ii] = zax[ii]/lax;
      nphi[ii] = save_atan2(yax[ii],xax[ii]);
    }
}

/* ------------------------------------------------------------------ */
/* Mittelwert ueber die Durchbiegung der  Ketten                      */

double msr_bend()
{

    int ii,bonds,n2;
    double bend,fac1,fac2;
    double dx,dy,dz;

    n2 = lpm/2;
    bend= 0.0;
    bonds= lpm - 1;

    fac1 = ((double)      n2   ) / ((double) bonds);
    fac2 = ((double) (bonds-n2)) / ((double) bonds);
```



```
    /* fac1*Kopf + fac2*Ende --> dem Mittel-Monomer entsprechende Lage */
    /* auf der  Ketten-Achse; Abweichung voneinander entspr. Biegung   */
    for (ii=0;ii<npm;ii++) {

        dx = fac1*xendt[2*ii]+fac2*xendt[2*ii+1]-xn2t[ii];
        dy = fac1*yendt[2*ii]+fac2*yendt[2*ii+1]-yn2t[ii];
        dz = fac1*zendt[2*ii]+fac2*zendt[2*ii+1]-zn2t[ii];

        bend = bend + dx*dx + dy*dy + dz*dz;
    }

    bend = bend / (double) npm;

    return(bend);
}

/* ------------------------------------------------------------------ */
/* Normierung von R_xy zum Parameter O                                */

void msr_O(double *O,double *R_xy,double *nnr)
{
    int ii;
    double mx,my,mr,dx,dy;

    mx = my = mr = 0.0;
    for (ii=0; ii<npm; ii++)
      { dx = xendt[2*ii]-xendt[2*ii+1];
        dy = yendt[2*ii]-yendt[2*ii+1];
        mx += dx;
        my += dy;
        mr += sqrt(dx*dx + dy*dy);
      }
   *nnr = mr/npm;
    dx  = mx/npm;
    dy  = my/npm;
   *R_xy= sqrt(dx*dx + dy*dy);
   *O   = (*R_xy)/(*nnr);
}

/* ------------------------------------------------------------------ */
/* Mittelwert ueber die x-Komponenten der  Ketten-Achsen              */

double msr_x()
{
    int ii;
    double mx;

    mx= 0.;
    for (ii=0; ii<npm; ii++)
      { mx = mx + xendt[2*ii]-xendt[2*ii+1]; }

    mx = mx / npm;
    return(mx);
}

/* ------------------------------------------------------------------ */
/* Varianz "sigma" der x-Komponenten der  Ketten-Achsen               */

double msr_sigx()
{
    int ii;
    double sx,mx,dx;
```

```
    sx= 0.;
    mx= 0.;
    for (ii=0;ii<npm;ii++) {
        dx = xendt[2*ii]-xendt[2*ii+1];
        mx = mx + dx;
        sx = sx + dx*dx;
    }
    sx = sx/npm - mx*mx/(npm*npm);
    return(sx);
}

/* ------------------------------------------------------------------ */
/* Mittelwert ueber die y-Komponenten der  Ketten-Achsen              */

double msr_y()
{
    int ii;
    double my;

    my= 0.;
    for (ii=0;ii<npm;ii++) {
        my = my + yendt[2*ii]-yendt[2*ii+1];
    }
    my = my / npm;
    return(my);
}

/* ------------------------------------------------------------------ */
/* Mittelwert ueber die z-Komponenten der  Ketten-Achsen              */

double msr_z()
{
    int ii;
    double mz;

    mz= 0.;
    for (ii=0;ii<npm;ii++) {
        mz = mz + zendt[2*ii]-zendt[2*ii+1];
    }
    mz = mz / npm;
    return(mz);
}

/* ------------------------------------------------------------------ */
/* Mittelwert <Kosinus(theta)> der Neigungswinkel                     */

double msr_cos()
{
    int ii;
    double mcos;

    mcos  = 0.;
    for (ii=0; ii<npm; ii++)
      { mcos = mcos + ncth[ii]; }

    mcos = mcos / npm;
    return(mcos);
}

/* ------------ "Sichere" Version von atan2 : ------------------------ */

double save_atan2(double y,double x)
{
```



```
  double maxrel = 1.0e4;

  if ((x==0.0)&&(y==0.0)) return(0.0);

  if (fabs(y) > fabs(maxrel*x))
    { if ((y<0)&&(x<0))
        { return( -M_PI_2 - atan2(-x,-y) ); }
      else
        { return(  M_PI_2 - atan2(x,y) ); }
    }
  else
    { return(atan2(y,x)); }
}

/* ------------------------------------------------------------------ */
/* Mittelwert der Neigungsrichtungen <phi>                            */

double msr_phi()
{
    int ii;
    double phi;

    phi = 0;
    for (ii=0; ii<npm; ii++)
      { phi = phi + nphi[ii]; }

    phi = phi / npm;
    return(phi);
}

/* ------------------------------------------------------------------ */
/* Mittelwert des Neigungswinkels <theta>                             */
/* ist i.A. verschieden von arccos( <cos(theta)> )                    */

double msr_theta()
{
    int ii,ihist;
    double thadd,theta;

    theta = 0.;
    for (ii=0; ii<npm; ii++)
      { thadd = acos(ncth[ii]);
        theta = theta + thadd;

        ihist = (int) (100.*thadd/M_PI_2);   /* Histo-Kaestchen best.  */
        histothe[ihist]++;                 /* Einfuegen ins Histogramm */
    }
    theta = theta / npm;
    return(theta);
}

/* ------------------------------------------------------------------ */
/* Varianz sigma(theta) = <thata^2> - <theta>^2                       */

double msr_sigthe()
{
    int ii;
    double inv,the2,theta;

    the2  = 0.;
    theta = 0.;
    for (ii=0; ii<npm; ii++)
      { inv = acos(ncth[ii]);
```

MC_m.c 15/27

```c
        theta = theta + inv;
        the2  = the2 + inv*inv;
    }
    the2 = the2/npm - theta*theta/(npm*npm);
    return(the2);
}

/* ------------------------------------------------------------------- */
/* Mittelwerte der relativen Bondorientierungen gegen  Ketten-Achse    */

void msr_bondcos(double bcos[])
{
    int ii,jj;
    double dirx,diry,dirz,ldir,bx,by,bz,lb,Si;

    for(ii=0; ii<lpm-1; ii++) bcos[ii]=0.0;

    /* Schleife ueber alle Ketten                                      */
    for(jj=0; jj<npm; jj++){
        dirx = -xax[jj];
        diry = -yax[jj];
        dirz = -zax[jj];
        ldir = sqrt(dirx*dirx +diry*diry +dirz*dirz);

        /* Schleife ueber die Monomere einer Kette    (ausser Ende)    */
        for(ii=0;ii<lpm-1;ii++){
           bx = xmm[ii+jj*lpm] - xmm[ii+jj*lpm+1]; /* Vektor zwischen  */
           by = ymm[ii+jj*lpm] - ymm[ii+jj*lpm+1]; /* aufeinander fol- */
           bz = zmm[ii+jj*lpm] - zmm[ii+jj*lpm+1]; /* genden Monomeren */
           lb = sqrt(bx*bx + by*by + bz*bz);
           /* Kosinus des Winkels zwischen Bond-b und  Ketten-Haupt-   */
           /* Richtung-dir wird ausgerechnet,                          */
           /* bildet Mittelwert (ueber alle Ketten)   der relat. Bond- */
           /* richtungen                                               */

           /* ACHTUNG : vorher   < cos(abw) >                          */
           /*           jetzt    < 3*cos^2(abw)-1 >/2                  */

           Si = (dirx*bx +diry*by +dirz*bz)/(ldir*lb);
           Si = (3.0*Si*Si-1.0)/2.0;
           bcos[ii] = bcos[ii] + Si;
        }
    }

   for(ii=0;ii<lpm-1;ii++) bcos[ii]=bcos[ii]/(double) npm;
}

/* ------------------------------------------------------------------- */
/* Messung der Beitraege zum Spreizdruck                               */
/* Aufruf mit "msr_spread_press(ppr,blpr,bapr)"                        */
/* Theorie vgl. [Haas] Anhang B                                        */
/* Indizes [0..2] entspricht Tensorkomponenten {xx; yy; xy}            */

void msr_spread_press(double press[],double bolenpress[],double boangpress[],
                      double mcs)
{
    int ii,jj,ip,jp,i1,i2,k;
    double dist,sqdist,bln,dx,dy,dz,area,pd4,pd7,x,y,z,rdst;
    double dx1,dy1,dz1,dx2,dy2,dz2,dst1,dst2,d2st1,d2st2,d1d2;

    double ljpress_fac[3];

    for (k=0; k<3; k++) { ljpress_fac[k] = lju0[k]/ljmq[k]; }
```

MC_m.c 16/27

```c
    for (ii=0; ii<3; ii++)
       { press[ii]      =0.;
         boangpress[ii] =0.;
         bolenpress[ii] =0.;
       }
    area = SSX*SHY;

    /* Schleife ueber alle Monomere                                    */
    for (i1=0; i1<ntot; i1++)
       if ((i1%lpm)!=0)     /* Koepfe haben keinen unteren Bondpartner */
         { i2=i1-1;
           /* (dx,dy,dz) -->  Abstandsvektor                          */
           dx = xmm[i1] -xmm[i2];
           dy = ymm[i1] -ymm[i2];
           dz = zmm[i1] -zmm[i2];
           dist   = dx*dx + dy*dy + dz*dz;
           sqdist = sqrt(dist);
           bln    = (sqdist - bond_length);

                  /* (P)_ij * A  =  N* k_B * T  +                     */
                  /*   << Sum_{i1,i2} F_{i1,i2}^i * (r_i1 - r_i2)^j >> */

                  /* F_{i1,i2}^x = - d'V(d_vec)/d'dx                  */
                  /* mit (dx,dy,dz) := (r_i1 - r_i2)_vec              */

                  /* V_BL(d)     = bond_laengen_potential(d)          */
                  /* d'V_BL/d'dx = 2*sprng*(d-b0) / (1-(d-b0)^2/b0^2) */
                  /*               * dx / d                           */

                  /* (P_BL)_xy = Sum_{i1,i2} dy * d'V_BL/d'dx  / A    */

           bolenpress[0] -= dx*dx*bln/(1.- bln*bln/bond_limit_quadrat)/sqdist;
           bolenpress[1] -= dy*dy*bln/(1.- bln*bln/bond_limit_quadrat)/sqdist;
           bolenpress[2] -= dx*dy*bln/(1.- bln*bln/bond_limit_quadrat)/sqdist;
         }

    bolenpress[0] *= 2.0*sprng/area;
    bolenpress[1] *= 2.0*sprng/area;
    bolenpress[2] *= 2.0*sprng/area;

    /* Schleife ueber alle Monomere                                    */
    for (ip=0; ip<ntot-1; ip++)
    { /* ........aeussere Schleife ueber alle Monomere........ */
      x = xmm[ip];
      y = ymm[ip];
      z = zmm[ip];

      for (jj=1; jj<=upper_verlet[ip]; jj++)
      { /* nur ueber die LJ-Partner, d.h. Abstand < rad_verlet !   */
        jp = tab_verlet[ip][jj];
        k = lj_type[ip][jp];
        speriod_dist_min(&dx,&dy,xmm[jp]-x,ymm[jp]-y,inv_SSX,inv_SHY,ctg_alpha);

        dz = zmm[jp] -z;
        dist   = dx*dx + dy*dy + dz*dz;

        if (dist < ljcq[k])
          { rdst = ljmq[k]/dist;
                pd4  = rdst*rdst*rdst*rdst;
                pd7  = rdst*rdst*rdst*rdst*rdst*rdst*rdst;
                press[0] += dx*dx*(pd7-pd4) * ljpress_fac[k];
                press[1] += dy*dy*(pd7-pd4) * ljpress_fac[k];
```

```
                press[2] += dx*dy*(pd7-pd4) * ljpress_fac[k];
            }

        } /* next jj,jp */
    } /* next ip */

    press[0] *= 12.0/area;
    press[1] *= 12.0/area;
    press[2] *= 12.0/area;

    /* Schleife ueber alle Ketten                                    */

    for (i1=0; i1<npm; i1++)
         /* Schleife ueber alle Bondwinkel der Kette                 */
       { for (i2=0; i2<lpm-2; i2++)
            { /* unterer Bondvektor                                  */
              dx1 = xmm[i1*lpm+i2] -xmm[i1*lpm+i2+1];
              dy1 = ymm[i1*lpm+i2] -ymm[i1*lpm+i2+1];
              dz1 = zmm[i1*lpm+i2] -zmm[i1*lpm+i2+1];
              d2st1 = dx1*dx1 + dy1*dy1 + dz1*dz1;
              dst1  = sqrt(d2st1);
              /* oberer Bondvektor                                  */
              dx2 = xmm[i1*lpm+i2+1] -xmm[i1*lpm+i2+2];
              dy2 = ymm[i1*lpm+i2+1] -ymm[i1*lpm+i2+2];
              dz2 = zmm[i1*lpm+i2+1] -zmm[i1*lpm+i2+2];
              d2st2 = dx2*dx2 + dy2*dy2 + dz2*dz2;
              dst2  = sqrt(d2st2);
              d1d2  = dx1*dx2 + dy1*dy2 + dz1*dz2;

              /* (P_BW)_xy = -( Sum_{ijk} dy_ij * d'V_BW/d'dx_ij +      */
              /*                          dy_jk * d'V_BW/d'dx_jk ) / A   */

              /* d'V_BW/d'dx_ij = [dx_jk - dx_ij * (d_ij*d_jk)/d_ij^2 ] */
              /*                  * -cospr / (|d_ij|*|d_jk|)            */

              /* +++++++++++++++++ ACHTUNG ++++++++++++++++++++++++++ */
              /*                                                      */
              /* Vorzeichen wurde berichtigt !                        */
              /*                                                      */
              /* ++++++++++++++++++++++++++++++++++++++++++++++++++++ */

              boangpress[0]= boangpress[0]
                   + (2.*dx1*dx2 - dx1*dx1*d1d2/d2st1 -dx2*dx2*d1d2/d2st2)
                   / (dst1*dst2);
              boangpress[1]= boangpress[1]
                   + (2.*dy1*dy2 - dy1*dy1*d1d2/d2st1 -dy2*dy2*d1d2/d2st2)
                   / (dst1*dst2);
              boangpress[2]= boangpress[2]
                   + (dy1*dx2 + dx1*dy2 - dy1*dx1*d1d2/d2st1 -dy2*dx2*d1d2/d2st2
)
                   / (dst1*dst2);
            } /* next i2 */
       } /* next i1 */

    boangpress[0] *= cospr/area;
    boangpress[1] *= cospr/area;
    boangpress[2] *= cospr/area;

    press[0]= press[0] + bolenpress[0] + boangpress[0];
    press[1]= press[1] + bolenpress[1] + boangpress[1];
    press[2]= press[2] + bolenpress[2] + boangpress[2];

    /* zum vollst. Druck fehlt noch der Term  + N * k_B*T  / A * [1]  */
```



```
    press[0] = press[0] + ntot * temp / area;
    press[1] = press[1] + ntot * temp / area;

    /* Druck ist jetzt korrekt, muss nicht weiter korrigiert werden ! */
}

/* ------------------------------------------------------------------ */
/* Prueft Hexagonalitaet des Kopfgruppen-Gitters                      */
/* vgl. "msr_latt2"                                                   */
/* Aufruf mit (0, &mdef, &dist, re6, im6)                             */

void msr_latt(int ilay,int *def,double *mdist,double *re6ord, double *im6ord)
{
    int ii,i1,i2,ihist,flag;
    double dx,dy,maxdif,tphi,xp,yp;

    struct sortlst {int pn; double x; double y; double dst; double phi;}
                  nsort[8],tmp;

    maxdif = 0.3;
    *def = 0;
    *mdist =0.;
    for (ii=0;ii<7;ii++)
       { re6ord[ii] = 0.;
         im6ord[ii] = 0.;
       }

    /* ................Schleife ueber alle Ketten  ................. */
    /* bzw. alle Monomere einer Ebene --> Kopf-Monomere               */
    for (i1=0;i1<npm;i1++)
    {
      for (ii=0;ii<7;ii++)                /* Sortier-Array vorbesetzen */
         {
           nsort[ii].pn = 0;
           nsort[ii].x = 0.;
           nsort[ii].y = 0.;
           nsort[ii].dst = 1000.;         /* uebergrosser Abstand      */
           nsort[ii].phi = 0.;
         }

      /* sucht aus allen anderen Monomeren derselben Ebene             */
      /* die 6 naechsten Nachbarn aus --> ilay = 0 entspr. Kopf-Ebene  */
      for (i2=0;i2<npm;i2++)
         {
           nsort[7].pn = i2;
           speriod_dist_min(&dx,&dy,
                            xmm[i2*lpm+ilay] - xmm[i1*lpm+ilay],
                            ymm[i2*lpm+ilay] - ymm[i1*lpm+ilay],
                            inv_SSX,inv_SHY,ctg_alpha);
           nsort[7].x = dx;      /* periodischer Abstand in x-Richtung */
           nsort[7].y = dy;      /* periodischer Abstand in y-Richtung */

           nsort[7].phi = save_atan2(dy,dx);

           /* Einordnen in Liste der kleinsten Abstaende (Bubble Sort) */
           nsort[7].dst = sqrt(dx*dx + dy*dy);
           for (ii=6; (ii>=0)&&(nsort[ii].dst>nsort[ii+1].dst); ii--)
              { tmp         = nsort[ii];
                nsort[ii]   = nsort[ii+1];
                nsort[ii+1] = tmp;
              }
         } /* next i2, Ende der Sortierschleife fuer Monomer i1       */
```

```
      /* nsort[0] ist pathologisch,                                  */
      /* denn jedes Monomer ist sich selbst am naechsten !           */
      if (nsort[6].dst-nsort[1].dst > maxdif)
         (*def)++;

      else /* Wenn kleinste Abstaende um weniger als 'maxdif' versch.  */
        {
         /* order with respect to phi  (bubble sort)                 */
         flag = 1.;
         while (flag != 0)
              {
                flag = 0;
                for (ii=1; ii<6; ii++)
                   {
                     if (nsort[ii].phi > nsort[ii+1].phi)
                       {
                         flag        = 1.;
                       tmp         = nsort[ii];
                       nsort[ii]   = nsort[ii+1];
                       nsort[ii+1] = tmp;
                       }
                   }
              } /* Ende der Sortierung nach Winkel phi                */

      /* Berechnung des hexagonalen Ordnungsparameters Psi_6         */

      /* Ueberlegung : im hexagonalen Gitter liegen die 6 naechsten */
      /* Nachbarn bei phi =~= {0..5}*60 Grad,                        */
      /* also 6*phi =~= {0..5}*360 Grad, die Punkte (re6ord,im6ord) */
      /* sollten also etwa am selben Ort um (1,0) sein !             */

      for (ii=1; ii<7; ii++)
         {
           *mdist = *mdist + nsort[ii].dst/6.;
           tphi   = save_atan2(nsort[ii].y,nsort[ii].x);

           re6ord[ii]= re6ord[ii] + cos(6*tphi);
           im6ord[ii]= im6ord[ii] + sin(6*tphi);
         }

      /* Auswerten der Daten fuer die Histogramme                    */

      /* x-y-Komponenten des Kopf-End-Vektors der Ketten             */
      xp   = xax[i1];
      yp   = yax[i1];
      tphi = nphi[i1];                  /* Neigungsrichtung */

      /* Mehrfachnutzung des Histogramm-Arrays 'histophi[]' :        */
      /*                                                             */
      /* histophi[  0.. 99] : Richtungen der naechsten Nachbarn      */
      /* histophi[100..199] : Neigungsrichtung der Ketten            */
      /* histophi[200..299] : Nachbarrichtungen relat. Neigungsrich.*/
      /* histophi[300..399] : Zwischenwinkel der naechsten Nachbarn */
      /* weiter in 'msr_latt2'                                       */

      ihist = (int) (100.*fmod(tphi+20.*M_PI,2.*M_PI)/(2.*M_PI));
      histophi[ihist+100]++;
      /*    histogram for tilt phi's   */
      for (ii=1; ii<7; ii++)
         { ihist = (int) (100.*fmod(nsort[ii].phi+20.*M_PI,2.*M_PI)/(2.*M_PI)
);
           histophi[ihist]++;
      /*    histogram for NN phi's     */
```



```
             ihist = (int) (100.*fmod(nsort[ii].phi-tphi+20.*M_PI,2.*M_PI)/(2.*
M_PI));
             histophi[ihist+200]++;
         /*    histogram for NN phi's relative to tilt phi   */
            }
         for (ii=1; ii<6; ii++)
            { ihist = (int) (100.*
                      fmod(-nsort[ii].phi+nsort[ii+1].phi+20.*M_PI,2.*M_PI)/(2.*
M_PI));
             histophi[ihist+300]++;
            }
         ihist = (int) (100.*
                 fmod(-nsort[6].phi+nsort[1].phi+20.*M_PI,2.*M_PI)/(2.*M_PI));
         histophi[ihist+300]++;
         /*    histogram for phi's between NN's   */

       } /* "else" : kleinste Abstaende etwa gleich                   */

    } /* ........... Ende der Schleife ueber die Ketten  ........... */

    if ((npm-*def)<1) *def=npm-1;

    *mdist= *mdist/(npm-*def);     /*  mittlerer Abstand zu den 6 NNs  */

    /* mittlere Lage der 6 naechsten Nachbarn,                        */
    /* auf (1,0)-Richtung gedreht !                                   */
    for (ii=0; ii<7; ii++)
       {
         re6ord[ii] = re6ord[ii]/(npm-*def);
         im6ord[ii] = im6ord[ii]/(npm-*def);
       }
}

/* ------------------------------------------------------------------ */
/* prueft Hexagonalitaet des Kopfgruppen-Gitters analog zu "msr_latt", */
/* jedoch ohne auf evtl. vorhandene Loecher zu achten                 */
/* mS2cos : Winkel-Korrelationsfunktion unter naechsten Nachbarn :    */
/* mS2cos :=  < 3/2 * cos^2(Theta(nn)) - 1/2 >                        */

void msr_latt2(int ilay,double *mS2cos,double *mdist,
               double *re6ord,  double *im6ord,
               double *re6_phi, double *im6_phi,
               double *re_phi,  double *im_phi)
{
    int ii,ihist,i1,i2,flag;
    double dx,dy,tphi;
    double disx1,disx2,disy1,disy2,disz1,disz2,end1,end2,n1n2;

    struct sortlst {int pn; double x; double y; double dst; double phi;}
                   nsort[8],tmp;

    *mS2cos = 0.;
    *mdist =0.;
    for (ii=0; ii<7; ii++)
       { re6ord[ii] = 0.;
         im6ord[ii] = 0.;
         re6_phi[ii] = 0.;
         im6_phi[ii] = 0.;
       }
    *re_phi = 0.0;    *im_phi = 0.0;

    /* ................... Schleife ueber alle Koepfe ............... */
    for (i1=0; i1<npm; i1++)
```

```
{
  for (ii=0; ii<7; ii++)
     { nsort[ii].pn = 0;
       nsort[ii].x = 0.;
       nsort[ii].y = 0.;
       nsort[ii].dst = 1000.;
       nsort[ii].phi = 0.;
     }

  /* select 6 nearest Neibours for particle i1 */
  for (i2=0; i2<npm; i2++)
     { nsort[7].pn = i2;

       speriod_dist_min(&dx,&dy,
                        xmm[i2*lpm+ilay] - xmm[i1*lpm+ilay],
                        ymm[i2*lpm+ilay] - ymm[i1*lpm+ilay],
                        inv_SSX,inv_SHY,ctg_alpha);
       nsort[7].x = dx;      /* periodischer Abstand in x-Richtung */
       nsort[7].y = dy;      /* periodischer Abstand in y-Richtung */

       /* Richtung zum NN bezogen auf x-Achse                       */
       nsort[7].phi = save_atan2(dy,dx);

       nsort[7].dst = sqrt(dx*dx + dy*dy);

       for (ii=6; (ii>=0)&&(nsort[ii].dst>nsort[ii+1].dst); ii--)
          { tmp         = nsort[ii];
            nsort[ii]   = nsort[ii+1];
            nsort[ii+1] = tmp;
          }
     }
  /* sort(Entfernung)                          */

  /* order with respect to phi (bubble sort)   */
     flag = 1.;
     while (flag != 0)
          { flag =0;
            for (ii=1; ii<6; ii++)
               { if (nsort[ii].phi >nsort[ii+1].phi)
                   { flag = 1.;
                     tmp = nsort[ii];
                     nsort[ii] = nsort[ii+1];
                     nsort[ii+1] = tmp;
                    }
               }
          }
   /* sort(Winkel)                             */


   /* . . . . . Schleife ueber die 6 NNs . . . . . . . . */

   *re_phi = *re_phi + cos(6.0*nphi[i1]);
   *im_phi = *im_phi + sin(6.0*nphi[i1]);

     for (ii=1; ii<7; ii++)
        { *mdist = *mdist + nsort[ii].dst/6.;
          tphi   = save_atan2(nsort[ii].y,nsort[ii].x);
          /* mittlere Lage der 6 NN auf (1|0) gedreht   */
          re6ord[ii]= re6ord[ii] + cos(6*tphi);
          im6ord[ii]= im6ord[ii] + sin(6*tphi);

          re6_phi[ii] += cos(6.0*(nphi[ii]-tphi));
          im6_phi[ii] += sin(6.0*(nphi[ii]-tphi));
```



```
       /*   measurement of orientation correlation function */
       /*   for nearest neighbours                          */
            /* Axial-Vektor der Kette i1         */
            disx1 = xax[i1];
            disy1 = yax[i1];
            disz1 = zax[i1];
            /* Axial-Vektor des ii.ten Nachbarn  */
            disx2 = xax[nsort[ii].pn];
            disy2 = yax[nsort[ii].pn];
            disz2 = zax[nsort[ii].pn];
            end1 = disx1*disx1 + disy1*disy1 + disz1*disz1;
            end2 = disx2*disx2 + disy2*disy2 + disz2*disz2;
            n1n2 = disx1*disx2 + disy1*disy2 + disz1*disz2;
            /* mS2cos = < 3/2 * cos^2(theta(nn)) - 1/2 */
            *mS2cos = *mS2cos + 3.*n1n2*n1n2/(end1*end2) -1.;
          }
       /* . . . . . Schleife ueber die 6 NNs . . . . . . . . */

         /* Mehrfachnutzung des Histogramm-Arrays 'histophi[]' :       */
         /*                                                            */
         /* histophi[  0.. 99] : Richtungen der naechsten Nachbarn     */
         /* histophi[100..199] : Neigungsrichtung der Ketten           */
         /* histophi[200..299] : Nachbarrichtungen relat. Neigungsrich.*/
         /* histophi[300..399] : Zwischenwinkel der naechsten Nachbarn */
         /*                                                            */
         /* histophi[400..499] : Richtungen der naechsten Nachbarn [#] */
         /* histophi[500..599] : Neigungsrichtung der Ketten       [#] */
         /* histophi[600..699] : Nachbarricht. relat. Neigungsrich.[#] */
         /* histophi[700..799] : Zwischenwinkel der NNs            [#] */
         /* [#] : in 'msr_latt2' werden evtl. Loecher nicht separiert  */

       /* add to histograms                          */
         tphi = nphi[i1];

         ihist = (int) (100.*fmod(tphi+20.*M_PI,2.*M_PI)/(2.*M_PI));
         histophi[ihist+500]++;
         /*    histogram for tilt phi's   */
         for (ii=1; ii<7; ii++)
            { ihist = (int) (100.*
                      fmod(nsort[ii].phi+20.*M_PI,2.*M_PI)/(2.*M_PI));
              histophi[ihist+400]++;
         /*    histogram for NN phi's     */
              ihist = (int) (100.*
                      fmod(nsort[ii].phi-tphi+20.*M_PI,2.*M_PI)/(2.*M_PI));
              histophi[ihist+600]++;
         /*    histogram for NN phi's relative to tilt phi   */
            }
         for (ii=1; ii<6; ii++)
            { ihist = (int) (100.*
                      fmod(-nsort[ii].phi+nsort[ii+1].phi+20.*M_PI,2.*M_PI)/(2.*
M_PI));
              histophi[ihist+700]++;
            }
         ihist = (int) (100.*
                 fmod(-nsort[6].phi+nsort[1].phi+20.*M_PI,2.*M_PI)/(2.*M_PI));
         histophi[ihist+700]++;
         /*    histogram for phi's between NN's   */

         /* Histogramm fuer die Entfernungen zu den 6 NNs              */
         /* erwarteter Wert fuer Gitterkonstante der Koepfe : 1.0      */
         for (ii=1; ii<7; ii++)
            { if (nsort[ii].dst>=10.0)
```

MC_m.c 23/27

```
                    histodst[1000]++;   /* Obergrenze  */
              else
                { ihist= (int) (100.0*(nsort[ii].dst));
                  histodst[ihist]++;
                }
         /*    histogram for distances to NN's     */
            }

    } /* ...... i1 Schleife ueber alle Kopfgruppen ..................  */


    *mdist= *mdist/npm;            /* Mittlere Entfernung              */
    *mS2cos = *mS2cos/(12.*npm);   /* Neigungswinkel-Korrelation       */
    /* mittlere Lage der 6 NN auf (1|0) gedreht                        */
    for (ii=0; ii<7; ii++)
       { re6ord[ii] = re6ord[ii]/(npm);
         im6ord[ii] = im6ord[ii]/(npm);
         re6_phi[ii] /= (double) npm;
         im6_phi[ii] /= (double) npm;
       }
    *re_phi = *re_phi/(double) npm;
    *im_phi = *im_phi/(double) npm;
}

/* ------------------------------------------------------------------ */
/* Bildet den Mittelwert des Laengenquadrats der Ketten               */
/* Laenge = Abstand von Kopf-Monomer zu End-Monomer                   */

double msr_end()
{
    int ii;
    double dx,dy,dz,end;

    end= 0.;
    for (ii=0;ii<npm;ii++) {
        dx = xendt[2*ii]-xendt[2*ii+1];
        dy = yendt[2*ii]-yendt[2*ii+1];
        dz = zendt[2*ii]-zendt[2*ii+1];
        end = end  +  dx*dx + dy*dy + dz*dz;
    }
    end = end /(double ) npm;

    return(end);
}

/* ------------------------------------------------------------------ */
/* nur Bond-Laengen- und LJ-WW-Energie pro Monomer                    */
/* d.h. ohne Bond-Winkel-Potential                                    */

double msr_e()
{
        int ii;
        double e;

        e = 0.;
        for (ii=0;ii<npm*lpm;ii++){
                e = e + energy[ii];
        }
        e = e /(npm*lpm);
        e = e / 2.;
        return(e);
}
```

MC_m.c 24/27

```
/* ------------------------------------------------------------------ */
/* Bond-Winkel-Energie pro Monomer                                    */

double msr_angene()
{
        int ii;
        double ae;

        ae = 0.;
        for (ii=0;ii<npm*lpm;ii++){
                ae = ae + cospot[ii];
        }
        ae = ae /(npm*lpm);
        return(ae);
}

/* ------------------------------------------------------------------ */
/* berechnet gesamte Innere Energie pro Monomer;                      */
/* Zahl der Monomere := npm*lpm = #Ketten   * Kettenlaenge            */
/* Gesamtenergie     := energy + cospot = Bond-, LJ- und Bond-Winkel  */

double msr_energy()
{
        int ii;
        double e,cosene;

        e = 0.;
        cosene = 0.;
        for (ii=0;ii<npm*lpm;ii++){
                e = e + energy[ii];
                cosene = cosene + cospot[ii];
        }
        e = e /(npm*lpm);
        cosene = cosene/(npm*lpm);

     /* Da alle Monomer-Monomer-WW bisher beiden Monomeren zugeordnet  */
     /* wurden, sind sie in der Summe doppelt gezaehlt worden !        */
        e = e / 2.;

        e = e + cosene;
        return(e);
}

/* ------------------------------------------------------------------ */
/* Mittleres Bond-Laengen-Potential pro Monomer                       */

double msr_bondene()
{
        int i1,i2;
        double be,xdist,ydist,zdist,dist;

        be = 0.;
        for (i1=0;i1<npm;i1++){
            for (i2=0;i2<(lpm-1);i2++){
                xdist = xmm[i1*lpm +i2] -xmm[i1*lpm +i2 +1];
                ydist = ymm[i1*lpm +i2] -ymm[i1*lpm +i2 +1];
                zdist = zmm[i1*lpm +i2] -zmm[i1*lpm +i2 +1];
                dist  = sqrt(xdist*xdist +ydist*ydist +zdist*zdist);
                if ((dist>bond_max)||(dist<bond_min))
                  { be = be + LJ_clip; }
                else
                  { be = be + bond_laengen_potential(dist); }
            }
```

```c
        }
        be = be/(npm*lpm);
        return(be);
}

/* ---------------------------------------------------------------- */
/* zentraler Aufruf fuer Messungen am System                        */

void  measurement(accpt,vaccptx,vaccpty,skin_verlet,shracpt,ptaccpt)
double            accpt,vaccptx,vaccpty,skin_verlet,shracpt,ptaccpt;
{
        char fname[20];
        int ii;
        int mdef;
        double S2cos,dist,re6[7],im6[7],ppr[3],bapr[3],blpr[3],pbco[20],
               re6_phi[7],im6_phi[7],re_phi,im_phi;
        double enth,dens,innere_energie,O,R_xy,nnr;
        FILE *f1,*f2,*f3,*f4,*f5,*f6,*f8,*f9,*f10,*f11;

        sprintf(fname,"%s_b.dat",swtch[1]);
    f1 = fopen(fname,"a");                  /* "a" --> append-Zugriff ! */
        sprintf(fname,"%s_u.dat",swtch[1]);
    f2 = fopen(fname,"a");
        sprintf(fname,"%s_a.dat",swtch[1]);
    f3 = fopen(fname,"a");
        sprintf(fname,"%s_p.dat",swtch[1]);
    f4 = fopen(fname,"a");
        sprintf(fname,"%s_o.dat",swtch[1]);
    f10= fopen(fname,"a");
        sprintf(fname,"%s_n.dat",swtch[1]);
    f11= fopen(fname,"a");
        sprintf(fname,"%s_t.dat",swtch[1]);
    f5 = fopen(fname,"a");
        sprintf(fname,"%s_v.dat",swtch[1]);
    f6 = fopen(fname,"a");

    if (dim == 3)
      {  sprintf(fname,"%s_l.dat",swtch[1]);
         f8 = fopen(fname,"a");
         sprintf(fname,"%s_sprpr.dat",swtch[1]);
         f9 = fopen(fname,"a");
      }

    /* Sammelt ausgew. Monomer-Koordinaten                          */

    measure_all();

    /* relat. Bondorientierung gegen  Ketten-Neigungsrichtung       */
    msr_bondcos(pbco);

    /* msr_end  = Mittelwert des Laengenquadrats der Ketten         */
    /* msr_bend = Mittelwert der quadr. Mittenabweichung der Ketten */

    fprintf(f1,"%f %f %f",mcs,msr_end(),msr_bend());
    for(ii=0;ii<lpm-1;ii++) fprintf(f1," %f",pbco[ii]);
    fprintf(f1,"\n");
    fclose(f1);                     /*   <-----  schliesst '*_b.dat'  */
                                    /*    -----> weiter mit MEANK     */

    /* Bestimmung der Enthalpie pro Monomer !                       */
    /* H = U + p*V                                                  */
    /* msr_energy() gibt Innere Energie U pro Monomer zurueck       */
    innere_energie = msr_energy();
```



```c
    enth           = innere_energie +
                     pressure*SSX*SHY/ntot;
    /* U; U-BondWinkel; BondWinkel; BondLaenge; Enthaltpie H   / Mono  */
    fprintf(f2,"%f %f %f %f %f %f\n"
           ,mcs,innere_energie,msr_e(),msr_angene(),msr_bondene(),enth);
    fclose(f2);                     /*   <-----  schliesst '*_u.dat'  */
                                    /*    -----> weiter mit MEANu     */

    /* speichert Akzeptanz-Raten fuer Monomer- und Volumen-Moves    */
    fprintf(f3,"%f %f %f %f %f %f %f\n",
            mcs,accpt,vaccptx,vaccpty,skin_verlet,shracpt,ptaccpt);
    fclose(f3);                     /*   <-----  schliesst '*_a.dat'  */

    /* Mittelwert der  Ketten-Achsen  msr_x,_y,_z                   */
    /* Varianz sigma der x-Komponente der  Ketten-Achsen            */
    fprintf(f4,"%f %f %f %f %f\n"
           ,mcs,msr_x(),msr_y(),msr_z(),msr_sigx());
    fclose(f4);                     /*   <-----  schliesst '*_p.dat'  */
                                    /*    -----> weiter mit MEANp     */

    /* normierter Parameter O, R_xy und zug. Nenner                 */
    msr_O(&O,&R_xy,&nnr);
    fprintf(f10,"%f %f %f %f\n"
           ,mcs,O,R_xy,nnr);
    fclose(f10);                    /*   <-----  schliesst '*_o.dat'  */


    /* <cos(theta)>, <theta>, sigma(theta), <phi>                   */
    fprintf(f5,"%f %f %f %f %f\n",
            mcs,msr_cos(),msr_theta(),msr_sigthe(),msr_phi());
    fclose(f5);                     /*   <-----  schliesst '*_t.dat'  */

    /* Dichte dens := Zahl der Ketten / Box-Flaeche !               */
    /* intuitiv -->   Zahl ... / Box-Volumen                        */
    dens     = npm/(SSX*SHY);
    fprintf(f6,"%f %f %f %f %f %f\n",mcs,
                SSX,SSY,system_size_z,dens,alpha_io);
    fclose(f6);                     /*   <-----  schliesst '*_v.dat'  */


    if (dim == 3) {
       mdef =0;
       dist =0.;
       /* prueft Hexagonalitaet des Kopfgruppen-Gitters             */
       msr_latt(0,&mdef,&dist,re6,im6);

       /* Zahl der Loecher, mittlerer Abstand der 6 NN,             */
       /* mittlere Lage der 6 NN (auf (1|0) gedreht                 */
       fprintf(f8,"%f %d %f\n",mcs,mdef,dist);
       for(ii=1;ii<7;ii++) fprintf(f8," %f",re6[ii]);
       fprintf(f8,"\n");
       for(ii=1;ii<7;ii++) fprintf(f8," %f",im6[ii]);
       fprintf(f8,"\n");

       dist =0.;
       S2cos=0.;
       /* prueft Hexagonalitaet des Kopfgruppen-Gitters ohne auf evtl. */
       /* Loecher zu achten  (analog "msr_latt")                    */
       /* bestimmt S2cos die Neigungswinkel-Korrelation der NNs     */
       msr_latt2(0,&S2cos,&dist,re6,im6,
                 re6_phi,im6_phi,&re_phi,&im_phi);

       fprintf(f8,"%f %f %f\n",mcs,S2cos,dist);
```

MC_m.c **27/27**

```
        for(ii=1;ii<7;ii++) fprintf(f8," %f",re6[ii]);
        fprintf(f8,"\n");
        for(ii=1;ii<7;ii++) fprintf(f8," %f",im6[ii]);
        fprintf(f8,"\n");
        fclose(f8);                  /*   <-----  schliesst '*_l.dat'  */
                                     /*    -----> weiter mit MEANlatt  */

        fprintf(f11," %f",mcs);
        for(ii=1;ii<7;ii++) fprintf(f11," %f",re6_phi[ii]);
        fprintf(f11," %f",re_phi);
        fprintf(f11,"\n           ");
        for(ii=1;ii<7;ii++) fprintf(f11," %f",im6_phi[ii]);
        fprintf(f11," %f",im_phi);
        fprintf(f11,"\n");
        fclose(f11);                 /*   <-----  schliesst '*_n.dat'  */

        /* bestimmt die xx-,yy,xy-Komponenten des Spreizdruck-Tensors   */
        /* blpr,bapr sind dessen Anteile vom Bond-Laengen- und vom      */
        /*           Bond-Winkel-Potential                              */
        msr_spread_press(ppr,blpr,bapr,mcs);

        fprintf(f9,"%f %f %f %f %f %f %f %f %f %f\n",mcs
        ,ppr[0],ppr[1],ppr[2],blpr[0],blpr[1],blpr[2],bapr[0],bapr[1],bapr[2]);
        fclose(f9);                /*   <-----  schliesst '*_sprpr.dat'  */
                                   /*    -----> weiter mit MEANsprB      */

      }
}
```

matrix.h & matrix.c 1/8

```c
/* matrix.h */
/* %W%   %E% */
#define STANDARD_C

#ifdef  STANDARD_C
void *matrix_alloc( size_t dim, size_t el_size, ...);
void *harve_alloc(size_t dim, size_t el_size, ... );
void mallprot(void *newblck, int size, void *oldblck);
#endif

#ifdef  NO_STANDARD_C
void *matrix_alloc ();
void *harve_alloc ();
void mallprot();
#endif

#ifndef My_MATRIX_ALLOC
#define My_MATRIX_ALLOC
#include <malloc.h>
#define mfree(a) mallprot(a,-1,a); free((char *) a);
#endif
```

```c
/* matrix.c */
/* %W%   %E% */
#define PROTOCOL
#define STANDARD_C

#define MAXBLOCKS 1000
#define MAXDIM 10
#define NULL    0

#ifdef NO_STANDARD_C
#include <varargs.h>
#endif

/* Header-Datei mit den Definitionen va_...
   zur Behandlung variabler Parameterlisten  */
#ifdef STANDARD_C
#include <stdarg.h>
#endif

#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "matrix.h"

/* ------------------------------------------------------------ */
/* belegt den Speicherplatz fuer "cell"                         */

/* matrix_alloc(3, sizeof(struct part), 2*number_of_cells_x, 2*.._y, .._z) */

/* hat zwei verschiedene Oberteile aber gemeinsames Unterteil !  */
#ifdef STANDARD_C

/*      dim = Dimensionalitaet des zu reservierenden Arrays      */
/*  el_size = Bytegroesse eines einzelnen Elementes              */

void *matrix_alloc( size_t dim, size_t el_size, ...)
{
```

matrix.c 2/8

```c
        size_t cant_length[MAXDIM];
        va_list pvar;
        size_t  i,j;
        size_t  data_mem,aux_mem,mem_help,offset;
        void    **pointer,*ptr;
        char    *block, *address;

        if(dim<0 || dim>MAXDIM) return(NULL);

        if(dim==0)
        {
                ptr=(void *)calloc(1,el_size);      /* pathologischer Fall */
        mallprot(ptr,(int)el_size,ptr);             /* nur ein Element     */
                assert(ptr!=NULL);                  /* Abbruch mit Meldung */
                return(ptr);
        }

        va_start(pvar,el_size);
        for(i=0;i<dim;i++) cant_length[i]=va_arg(pvar,size_t);

        if(dim==1)
        {
                ptr=(void *)calloc(cant_length[0],el_size);
        mallprot(ptr,(int)(cant_length[0]*el_size),ptr);
                assert(ptr!=NULL);
                return(ptr);
        }
        va_end(pvar);
#endif

/* - - -
```

matrix.c 3/8

```
                ptr=(void *)calloc(cant_length[0],el_size);
        mallprot(ptr,(int)(cant_length[0]*el_size),ptr);
                assert(ptr!=NULL);
                return(ptr);
        }
        va_end(pvar);
#endif

/* - - -
```

```c
/*  - - -
```

matrix.c 7/8

```
            " > free\t\t\t: %i bytes ==> %i total bytes\n",
                        sizes[indx],totsize-sizes[indx]);
                fflush(stdout);
#endif          /* - - -
```

```
/*******************************************************************/
/*                                                                 */
/*   File: r250.h                                                  */
/*                                                                 */
/*   Prototypes for the R250 random number generator               */
/*                                                                 */
/*******************************************************************/

/*
        $Id: r250.h,v 1.4 90/10/22 15:24:00 reger Exp $
*/

double double_r250(void);
double gauss_r250(double mean, double variance);
float float_r250(void);
float float_rand(void);
int r250_random(void);
int rand(void);
int rand_save(void);
void float_r250_vector(register float *x, int n);
void r250_restart(int table[]);
void r250_save(int table[]);
void r250_srandom(unsigned int seed);
void r250_vector(register int *x, int n);
void Srand(int seed);


/* ___________________________________________________________ */
/*                                                             */
/*  Der Zufallsgenerator r250 erzeugt gleichverteilte Zahlen aus  */
/*  dem Intervall [0;1[                                        */
/*                                                             */
/* vgl. Vattulainen et al.; Comp. Phys. Comm. 86 (1995) S. 209 */
/*                                                             */
/* und  Kirkpatrick, Stoll; J. Comp. Phys. 40 (1981) S. 517    */
/*                                                             */
/* ___________________________________________________________ */
```

r250.c 1/7

```
/*****************************************************************/
/*                                                               */
/*   File: r250.c                                                */
/*                                                               */
/*   The R250 random number generator routines.                  */
/*                                                               */
/*****************************************************************/

static char _ID_[] = "$Id: r250.c,v 1.8 90/10/31 14:57:50 reger Exp $";

/* #define int long */
#define INT16

/* Comment out nextline if sizeof(int) == 2 ( 16 Bit integer )  */
#undef  INT16

#include <math.h>
#include <stdio.h>

#ifndef INT_MAX
#ifdef INT16
#define INT_MAX         0x7FFF
#else
#define INT_MAX                 0x7FFFFFFF
#endif
#endif

#define INVERSE_INT_MAX         (1.0/INT_MAX)

#define         TYPE_3          3               /* x**31 + x**3 + 1 */
#define         BREAK_3         128
#define         DEG_3           31
#define         SEP_3           3

#define P 250
#define Q 103

static int m[P]   =
          {  0x7be9c1bd, 0x088aa102, 0x3d38509b, 0x746b9fbe, 0x2d04417f,
             0x775d4351, 0x53c48d96, 0x02b26e0b, 0x418fedcf, 0x19dbc19e,
             0x78512adb, 0x1a1f5e2b, 0x307d7761, 0x6584c1f0, 0x24e3c36f,
             0x2232310f, 0x2dac5ceb, 0x106e8b5a, 0x5a05a938, 0x5e6392b6,
             0x66b90348, 0x75264901, 0x4174f402, 0x618b18a4, 0x3a6ab4bf,
             0x3c4bc289, 0x2657a9a9, 0x4e68589b, 0x09648aa6, 0x3fc489bb,
             0x1c1b715c, 0x054e4c63, 0x484f2abd, 0x5953c1f8, 0x79b9ec22,
             0x75536c3d, 0x50b10549, 0x4d7e79b8, 0x7805da48, 0x1240f318,
             0x675a3b56, 0x70570523, 0x2c605143, 0x17d7b2b7, 0x55dbc713,
             0x514414b2, 0x3a09e3c7, 0x038823ff, 0x61b2a00c, 0x140f8cff,
             0x61ebb6b5, 0x486ba354, 0x0935d600, 0x2360aab8, 0x29f6bbf8,
             0x43a08abf, 0x5fac6d41, 0x504e65a2, 0x1208e35b, 0x6910f7e7,
             0x1012ef5d, 0x2e2454b7, 0x6e5f444b, 0x58621a1b, 0x077816af,
             0x6819306d, 0x4db58658, 0x58291bf8, 0x3597aa25, 0x45bb60a0,
             0x6a6a0f11, 0x1cf1e57c, 0x361265c4, 0x16ca6054, 0x34c99833,
             0x0bee2cd7, 0x680e7507, 0x6ed37bfa, 0x0f7650d6, 0x49c11513,
             0x02e308f9, 0x7162078c, 0x122cb868, 0x0c18defa, 0x14c2b244,
             0x3c237460, 0x4fb969b9, 0x746f1f85, 0x0c71da02, 0x61c24d14,
             0x5d80176d, 0x1c84c960, 0x0fe6a1cc, 0x4bdf5bb8, 0x74e6e37b,
             0x175eb87b, 0x33f88c25, 0x429c69d3, 0x6f87d474, 0x6990364a,
             0x0857ca73, 0x59f1e385, 0x06821bc6, 0x3e6a3037, 0x70bc43d9,
             0x3b4bb3fa, 0x4a585d0f, 0x58cab8e0, 0x2a1f2ff4, 0x59ceade5,
             0x228bcdf4, 0x2d0238ee, 0x4b30b571, 0x34b8865c, 0x391b17e8,
             0x5ff367b5, 0x70dbfabc, 0x08d481a1, 0x5462873b, 0x7d4dd4bf,
             0x6a96ceb6, 0x31e29ea8, 0x19d29e1f, 0x7a7d7082, 0x7dc1fa60,
```

r250.c 2/7

```
             0x0eb9819a, 0x11dc28fd, 0x31ba8685, 0x5155eb6d, 0x0163fd71,
             0x1b4abccf, 0x59adb5e0, 0x5b55e0f6, 0x21ccd896, 0x1817e618,
             0x4c1224d0, 0x5d188c90, 0x62704327, 0x24dcddb0, 0x0737bc84,
             0x3c3ef10c, 0x4768aba4, 0x3439f572, 0x076fa67e, 0x7c213200,
             0x6d550d5a, 0x67630e33, 0x6cfd2cbd, 0x76298efc, 0x3bc5956e,
             0x6a4b017c, 0x60c05db2, 0x6da83416, 0x041d9f9b, 0x5b3dce34,
             0x6b6a2e76, 0x12d72135, 0x6d19f731, 0x1d24b4fb, 0x642d0ca2,
             0x6e7df4a3, 0x386f71cb, 0x3ddac282, 0x49d3d599, 0x5a3c4a61,
             0x55f2a89a, 0x15e5fa69, 0x3754d6f1, 0x3862ebc1, 0x3ac2d81a,
             0x3e8c9375, 0x74a1dcce, 0x022b83be, 0x72c688e8, 0x7c11834c,
             0x7e4cb5bf, 0x601b9642, 0x6374917f, 0x6b49e27c, 0x5645253e,
             0x1f3a26ee, 0x5594e3f8, 0x370582f0, 0x0ce25b04, 0x59b28393,
             0x12435124, 0x784c897b, 0x6c89a4c8, 0x7f5d4856, 0x15713e76,
             0x50b6b16a, 0x6ddb3cf9, 0x4de0b041, 0x0e9173ec, 0x37af1292,
             0x281cfaa2, 0x64841c87, 0x4d950cfc, 0x5f71d193, 0x1ce70848,
             0x0857e516, 0x1dfe6509, 0x1188e516, 0x0a8368d4, 0x10c4edf1,
             0x0d9a6862, 0x08d01e93, 0x70e08433, 0x710ef9e2, 0x741a010f,
             0x4725a972, 0x104920d0, 0x49aee507, 0x7e2b2c62, 0x1d2b7bd4,
             0x2361689a, 0x106e7d87, 0x1578054f, 0x0feb0d62, 0x0fcbc5dd,
             0x2ae943c6, 0x60a1becc, 0x7da702d6, 0x78c9f407, 0x6f3332b9,
             0x35561568, 0x20e6eeaa, 0x53b74f40, 0x02eb2264, 0x0058c03d,
             0x709e5788, 0x0b43077a, 0x1e572546, 0x02273c9f, 0x15c6704f,
             0x2f1c1337, 0x0fc1a501, 0x1e968ee2, 0x1ffc976b, 0x00d09ee3,
             0x12b08ff2, 0x672240dd, 0x1119bfb3, 0x5c5f74f9, 0x654d6d3f,
             0x2e453b88, 0x7fc0dd94, 0x75bbeac6, 0x43bd40d7, 0x0fabeaf6 };

static int *k_static = m + P;
static int *j_static = m + P-Q;
static int *end      = m + P-1;

static  int     randtbl[ DEG_3 ]        =
                      { 0x9a319039, 0x32d9c024, 0x9b663182, 0x5da1f342,
                        0xde3b81e0, 0xdf0a6fb5, 0xf103bc02, 0x48f340fb,
                        0x7449e56b, 0xbeb1dbb0, 0xab5c5918, 0x946554fd,
                        0x8c2e680f, 0xeb3d799f, 0xb11ee0b7, 0x2d436b86,
                        0xda672e2a, 0x1588ca88, 0xe369735d, 0x904f35f7,
                        0xd7158fd6, 0x6fa6f051, 0x616e6b96, 0xac94efdc,
                        0x36413f93, 0xc622c298, 0xf5a42ab8, 0x8a88d77b,
                                    0xf5ad9d0e, 0x8999220b, 0x27fb47b9 };

static  int            *fptr                   = randtbl + SEP_3;
static  int            *rptr                   = randtbl;
static  int            *end_ptr                = randtbl + DEG_3;

static  int            *state                  = randtbl;

static  int             rand_deg                = DEG_3;
static  int             rand_sep                = SEP_3;

/* ---------------------------------------------------------------- */

static int rrandom()
{
        int            i;

        *fptr += *rptr;
        i = (*fptr >> 1) & INT_MAX;    /* chucking least random bit */
        if(  ++fptr  >=  end_ptr  )  {
            fptr = state;
            ++rptr;
        }
        else  {
            if(  ++rptr  >=  end_ptr  )  rptr = state;
```

```
        }
        return( i );
}

/* ------------------------------------------------------------------- */

static void ssrandom( x )

        unsigned int           x;
{
        register  int           i;

        int             rrandom();

        state[ 0 ] = x;
        for( i = 1; i < rand_deg; i++ )  {
            state[i] = (1103515245*state[i - 1] + 12345) & INT_MAX;
        }
        fptr = &state[ rand_sep ];
        rptr = &state[ 0 ];
        for( i = 0; i < 10*rand_deg; i++ )  rrandom();
}

/* ------------------------------------------------------------------- */
/* initialisiert den Zufallszahlengenerator mit 'seed'                 */

void r250_srandom ( seed )

    unsigned int seed;
{
        register int  i;

        ssrandom( seed );

        for ( i=0; i<P; ++i) m[i] = rrandom();

        k_static = m + P;
        j_static = m + P-Q;
}

/* ------------------------------------------------------------------- */

int r250_random ()
{

    if (--k_static < m) k_static = end;
    if (--j_static < m) j_static = end;

    return (*k_static ^= *j_static);
}

/* ------------------------------------------------------------------- */

float float_r250 ()
{
    if (--k_static < m) k_static = end;
    if (--j_static < m) j_static = end;

    return ( (*k_static ^= *j_static) * INVERSE_INT_MAX );
}

/* ------------------------------------------------------------------- */
```



```
/* liefert double-Randomwert in [0;1)                                  */

double  double_r250 ()
{
    if (--k_static < m) k_static = end;
    if (--j_static < m) j_static = end;

    return ( (*k_static ^= *j_static) * INVERSE_INT_MAX );
}

/* ------------------------------------------------------------------- */

void r250_save ( table )

    int table[];
{
    int i;

    for ( i = 0; i < P; i++ )
        table[i] = m[i];

    table[P  ] = k_static - m;
    table[P+1] = j_static - m;
}

/* ------------------------------------------------------------------- */

void r250_restart ( table )

    int table[];
{
    int i;

    for ( i = 0; i < P; i++ )
        m[i] = table[i];

    k_static = m + table[P  ];
    j_static = m + table[P+1];
}

/* ------------------------------------------------------------------- */

double gauss_r250 ( mean, variance )

    double              mean, variance;
{
    static int          empty = 1;
    static double       store;
    double              v1, v2, r, root;

    if ( empty ) {

        do {

            /*  Inline double_random()  */

            if (--k_static < m) k_static = end;
            if (--j_static < m) j_static = end;
            v1 = (*k_static ^= *j_static) * INVERSE_INT_MAX;

            v1 = 2.0 * v1 - 1.0;

            /*  Inline double_random()  */
```

r250.c 5/7

```
            if (--k_static < m) k_static = end;
            if (--j_static < m) j_static = end;
            v2 = (*k_static ^= *j_static) * INVERSE_INT_MAX;

            v2 = 2.0 * v2 - 1.0;

            r  = v1 * v1 + v2 * v2;

        } while ( r > 1.0 );

        root  = variance * sqrt( -2.0 * log(r) / r );
        store = v2 * root + mean;
        empty = 0;

        return ( v1 * root + mean );

    } else {

        empty = 1;
        return ( store );

    }

}

/*********************************************************************/
/*                                                                   */
/*   File: r250.c                                                    */
/*                                                                   */
/*   A simple multiplicative generator from Kalos & Whitlock.        */
/*                                                                   */
/*********************************************************************/

static int      rand_seed = 1;

int rand ()
{
    rand_seed = ( 1812433253 * rand_seed + 314159265 ) & INT_MAX;
    return  ( rand_seed );
}

/* ------------------------------------------------------------------- */

float float_rand ()

{
    rand_seed = ( 1812433253 * rand_seed + 314159265 ) & INT_MAX;
    return  ( rand_seed * INVERSE_INT_MAX );
}

/* ------------------------------------------------------------------- */

void Srand ( seed )

    int  seed;
{
    rand_seed = seed;
}

/* ------------------------------------------------------------------- */

int rand_save ()
```

r250.c 6/7

```
{
    return ( rand_seed );
}

#define MAX(a,b) ((a)>(b)?(a):(b))

/* ------------------------------------------------------------------- */

void r250_vector (x, n)

register int    *x;
int              n;
{
    register int        *j, *k, *l;

    j = j_static;
    k = k_static;

    while ( n > 0 ) {

        if ( k == m ) k = m + P;
        l = MAX( k-n, m+Q );
/*  printf("\n I: N = %d, k,l : %d %d %d %d", n, k, l, m, m+P); fflush(stdout);
 */
        if ( k > l ) {
            n -= k-l;
#ifdef __STDC__
#pragma ivdep
#endif
            while ( k > l )
                *x++ = (*--k ^= *--j);
        }

        if ( j == m ) j = m + P;
        l = MAX( j-n, m+P-Q );
/*  printf(" ==> N = %d, j,l : %d %d", n, j, l); fflush(stdout); */
        if ( j > l ) {
            n -= j-l;
#ifdef __STDC__
#pragma ivdep
#endif
            while ( j > l )
                *x++ = (*--k ^= *--j);
        }

    }

    j_static = j;
    k_static = k;

}

/* ------------------------------------------------------------------- */

void float_r250_vector (x, n)

register float  *x;
int              n;
{
    register int        *j, *k, *l;

    j = j_static;
    k = k_static;
```

```
    while ( n > 0 ) {

        if ( k == m ) k = m + P;
        l = MAX( k-n, m+Q );
        if ( k > l ) {
            n -= k-l;
#ifdef __STDC__
#pragma ivdep
#endif
            while ( k > l )
                *x++ = (*--k ^= *--j) * INVERSE_INT_MAX;
        }

        if ( j == m ) j = m + P;
        l = MAX( j-n, m+P-Q );
        if ( j > l ) {
            n -= j-l;
#ifdef __STDC__
#pragma ivdep
#endif
            while ( j > l )
                *x++ = (*--k ^= *--j) * INVERSE_INT_MAX;
        }

    }

    j_static = j;
    k_static = k;

}
```

DIAG.f 1/5
fuer nicht−periodische Konfigurationen

```
C  Programm DIAG.f liest Kopplungskonstanten ein,
C
C  diagonalisiert Kopplungsmatrix mittels NAG−Routine
C
C  und schreibt Eigenwerte/Eigenvektoren nach "DIAG"
C
C
C  CPU−Zeit : ca. 2 Stunden,  Plattenplatz : zusl. 170 MByte

      implicit none

C  2883 = 144 * (1*2 + 6*3) + 3
C
C  144 Ketten, 1 Kopfgruppe, 6 Schwanzmonomere  +  3 Boxparameter

      integer size
      parameter (size=2883)

      double precision cc(size,size),ew(size)

      integer i,j,k,l,m,n,ip,iq,monoip,monoiq,indxip,indxiq
      integer npm,lpm,npairs,mi1,mi2,ii,jj
      integer id1,id2,id3,id4,id5
      double precision ddum1,ddum2,ddum3,ddum4

C     fuer Eigenwertroutine :

      double precision work(16*size)

      character*1 job,uplo
      integer lwork,ifail

C  NAG−Routine zur Diagonalisierung einer reellen, symmetrischen Matrix
      external F02FAF

C ..................................................................

      lpm = 7
      npm = (size−3)/20

      ip = 546

C ------------------------------------------------------------------------

C     Kopplungsmatrix mit Nullen vorbesetzen

      do i=1,size
         do j=1,size
            cc(j,i) = 0d0
         enddo
      enddo

C --- Einlesen der von Null verschiedenen Kopplungkonstanten ---
C --- direkt in die Matrix CC                                 ---

C hier : Kopplungen zwischen 2 Monomeren

      open(unit=700,file='RESULT',status='OLD',
     &             access='SEQUENTIAL',form='FORMATTED')

      npairs = 0
      write(*,*) 'RESULT erfolgreich geoeffnet'
```

DIAG.f 2/5

```
C − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − −
10000 continue

      read(700,*,END=10001) id1,id2,id3,id4,id5,ddum1
      backspace 700

      ip = id2
      iq = id4

C Datenstruktur enthaelt 20 Freiheitsgrade pro Kette
C
C              1,2 : Kopfgruppe x,y
C            3,4,5 : 1.Monomer x,y,z
C            ...
C         18,19,20 : 6.Monomer x,y,z
C
C
C             2881 : x−Laenge    der Simulationsbox
C             2882 : y−Laenge    der Simulationsbox
C             2883 : Scherwinkel der Simulationsbox
C
C Im Eingabe−File sind z−Komponenten der Koepfe noch enthalten,
C dafuer nur Kopplungen zwischen Ketten i<=j.
C Diese Symmetrie ist exakt und keine Naeherung.
C
C Die andere "Haelfte" der Kopplungskonstanten muss rekonstruiert werden

      indxip = 20*(ip/lpm)
      indxiq = 20*(iq/lpm)
      monoip = mod(ip,lpm)
      monoiq = mod(iq,lpm)

      if (monoip.gt.0) indxip = indxip + 3*monoip − 1
      if (monoiq.gt.0) indxiq = indxiq + 3*monoiq − 1

      do m=1,3
         do n=1,3

            read(700,*,END=10002) id1,id2,id3,id4,id5,ddum1

            if ((id2.ne.ip).or.(id3.ne.m).or.
     &          (id4.ne.iq).or.(id5.ne.n)) then
               write(7,*) 'Fehler beim Einlesen bei Satz',
     &                    id1,id2,id3,id4,id5,ddum1
               write(7,*) 'erwartet : ',ip,m,iq,n
               write(7,*)
               stop 'ABBRUCH'
            endif

            if (abs(ddum1).lt.−1d−5) ddum1 = 0d0

            npairs = npairs + 1

            if (((monoip.ne.0).or.(m.ne.3)).and.
     &          ((monoiq.ne.0).or.(n.ne.3))) then

               cc(indxip+m,indxiq+n) = ddum1

               if (cc(indxiq+n,indxip+m).eq.0d0)
     &          cc(indxiq+n,indxip+m) = ddum1

            endif
```

```
         enddo
      enddo

      goto 10000

C ---------------------------------------------------------------

10001 continue

      close(700)

C hier : Kopplungen zwischen Monomer und Boxparameter

      open(unit=702,file='RESULT.box',status='OLD',
     &             access='SEQUENTIAL',form='FORMATTED')

      indxiq = 20*npm

C ...............................................................

      do ip=0,(npm*lpm)-1

         read(702,*,END=20003) id1,id2,id3,id4,ddum1,ddum2,ddum3
         backspace 702

         if (ip.ne.id2) goto 20003

         indxip = 20*(ip/lpm)
         monoip = mod(ip,lpm)
         if (monoip.gt.0) indxip = indxip + 3*monoip - 1

         do m=1,3
            do n=1,3

               read(702,*,END=20002) id1,id2,id3,id4,ddum1,ddum2,ddum3

               if ((id2.ne.ip).or.(id3.ne.m).or.(id4.ne.n)) then
                  write(7,*) 'Box : Fehler beim Einlesen bei Satz',
     &                       id1,id2,id3,id4,ddum1,ddum2
                  write(7,*)
                  stop 'Box : ABBRUCH'
               endif

               if (abs(ddum1).lt.-1d-5) ddum1 = 0d0
               if (abs(ddum2).lt.-1d-5) ddum2 = 0d0

               npairs = npairs + 1

               if ((monoip.ne.0).or.(m.ne.3)) then
                   cc(indxip+m,indxiq+n) = ddum1
                   cc(indxiq+n,indxip+m) = ddum2
               endif

            enddo
         enddo

      enddo

C ...............................................................

      indxip = 20*npm
      indxiq = 20*npm
```



```
      do m=1,3
         do n=1,3

            read(702,*,END=21002) id1,id2,id3,ddum1

            if ((id2.ne.m).or.(id3.ne.n)) then
               write(7,*) 'Box : Fehler beim Einlesen bei Satz',
     &                    id1,id2,id3,ddum1
               write(7,*)
               stop 'Box : ABBRUCH'
            endif

            npairs = npairs + 1

             if (abs(ddum1).lt.-1d-5) ddum1 = 0d0

            cc(indxip+m,indxiq+n) = ddum1

         enddo
      enddo

      close(702)

        write(*,*) 'Alle Daten erfolgreich gelesen'


C --- Symmetrisieren der Matrix (schon recht gut erfuellt) ---

      do i=1,size
         do j=i,size

            ddum1 = 0.5d0*(cc(j,i)+cc(i,j))
            cc(i,j) = ddum1
            cc(j,i) = ddum1

         enddo
      enddo

C ===================================================================
C Hier beginnt die Diagonalisierung der symmetrischen reellen Matrix cc

      job   = 'V'
      uplo  = 'U'
      ifail = 0

      call F02FAF(job,uplo,size,cc,size,ew,work,16*size,ifail)

C --- Ausgabe in grosses Datenfile (ca. 170 MByte) ---

      open(unit=701,file='DIAG',status='NEW',
     &             access='SEQUENTIAL',form='FORMATTED')

C     Spektrum der Eigenwerte
      write(701,'(F20.13)') (ew(k),k=1,size)

C fuer jede der 2883 Moden :
      do l=1,size

         i=l
         j=1
C      - Eigenwert
         write(701,*)
```

DIAG.f **5/5**

```
         write(701,'(F20.13)') ew(i)

         do m=0,npm-1
C      - Monomer-Komponenten des Eigenvektors
            write(701,'(3F20.13)') cc(j,i),cc(j+1,i)
            write(701,'(3F20.13)') (cc(k+j,i),k=2,19)
            j = j+20
         enddo

C      - Box-Komponenten des Eigenvektors
         write(701,'(3X,3F20.13)') (cc(k+j,i),k=0,2)

      enddo

      close(701)

C ..............................................................

      stop 'Programm erfolgreich beendet'

C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C Fehlermeldungen beim Einlesen der Daten :

10002 write(7,*) 'Unvollstaendiger 9er-Block'
      stop 'ABBRUCH'

20002 write(7,*) 'Box : Unvollstaendiger 9er-Block'
      stop 'Box : ABBRUCH'

20003 write(7,*) 'Box : auf falsches Monomer gestossen'
      stop 'Box : ABBRUCH'

21002 write(7,*) 'Box : Unvollstaendiger letzter 9er-Block'
      stop 'Box : ABBRUCH'

C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

 9000 format(I4,3X,F20.13,5X,2F20.13)

      end
```

Literaturverzeichnis

[1] F.F. Abraham
in *Melting, Localization, and Chaos*
Elsevier Sci. Publ. Comp. Inc. (1982)

[2] O. Albrecht, H. Gruler, E. Sackmann
J. Physique **39** (1978) S. 301

[3] M.P. Allen, D.J. Tildesley : *Computer Simulation in Chemical Physics*
NATO ASI Series Vol. 397, Kluwer Academic Publishers (1993)

[4] M.P. Allen, D.J. Tildesley : *Computer Simulation of Liquids*
Oxford Science Publications, Oxford University Press (1987)

[5] J. Als-Nielsen et al.
Phys. Rep. **246** (1994) S. 251

[6] N.W. Ashcroft, N.D. Mermin : *Solid State Physics*
Saunders College Publishing (1976)

[7] J.P. Bareman, G. Cardini, M.L. Klein
Phys. Rev. Lett. **60** (1988) S. 2152

[8] W.A. Barlow : *Langmuir-Blodgett-Films*
Elsevier Scientific Publishing Company (1980)

[9] J. Baschnagel et al.
J. Chem. Phys. **95**,8 (1991) S. 6014

[10] A.M. Bibo, I.R. Peterson
Adv. Mater. **2** (1990) S. 309

[11] K. Binder, D.W. Heermann : *Monte Carlo Simulation in Statistical Physics*
Solid-State Sciences 80, Springer-Verlag (1992)

[12] K.B. Blodgett
J. Am. Chem. Soc. **57** (1935) S. 1007

[13] M.J. Callaway, D.J. Tildesley, N. Quirke
Molecular Simulation **18** (1996) S. 277

[14] G. Cardini. J.P. Bareman, M.L. Klein
Chem. Phys. Lett. **145** (1988) S. 493

[15] K. Chen, T. Kaplan, M. Mostoller
Phys. Rev. Lett. **74** (1995) S. 4019

[16] N. Collazo, S. Shin, S.A. Rice
J. Chem. Phys. **96**,6 (1992) S. 4735

[17] B. Dünweg, K. Kremer
J. Chem. Phys. **99** (1993) S. 6983

[18] H.T. Epstein
J. Phys. Chem. **54** (1950) S. 1053

[19] V.B. Fainerman, D. Vollhardt, V. Melzer
J. Phys. Chem. **100** (1996) S. 15478

[20] B. Fischer, E. Teer, C.M. Knobler
J. Chem. Phys. **103** (1995) S. 2365

[21] D. Frenkel, G.C.A.M. Mooij, B. Smit
J. Phys. Condens. Matter **3** (1991) S. 3053

[22] D. Frenkel, B. Smit : *Understanding Molecular Simulation*
Academic Press (1996)

[23] S. Garoff et al.
J. Physique **47** (1986) S. 701

[24] R. Goetz, R. Lipowsky : *Self Assembly of Model Bilayers*
Preprint (1997)

[25] F.M. Haas
Dissertation, Universität Mainz, (1995)

[26] F.M. Haas, R. Hilfer, K. Binder
J. Phys. Chem. **100**,37 (1996) S. 15290

[27] F.M. Haas, R. Hilfer
J. Chem. Phys. **105**,9 (1996) S. 3859

[28] J. Harris, S.A. Rice
J. Chem. Phys. **39** (1988) S. 5898

[29] C.A. Helm, H. Möhwald, K. Kjaer, J. Als-Nielsen
Biophys. J. **52** (1987) S. 381

[30] S. Hénon, J. Meunier
Rev. Sci. Instrum. **62**,4 (1991) S. 936

[31] D. Hönig, D. Möbius
J. Phys. Chem. **95** (1991) S. 4590

[32] J. Israelachvili
Langmuir **10** (1994) S. 3774

[33] V.M. Kaganer, E.B. Loginov
Phys. Rev. E **51**,3 (1995) S. 2237

[34] V.M. Kaganer et.al.
J. Chem. Phys. **102**,23 (1995) S. 9412

[35] V.M. Kaganer, H. Möhwald, P. Dutta
Rev. Mod. Phys. (1998) Preprint

[36] S. Karaborni, S. Toxvaerd
J. Chem. Phys. **97**,8 (1992) S. 5876

[37] B.W. Kernighan, D.M. Ritchie : *The C Programming Language*
Prentice Hall (1988)

[38] S. Kirkpatrick, E.P. Stoll
J. Comp. Phys. **40** (1981) S. 517

[39] C.M. Knobler, R.C. Desai
Annu. Rev. Phys. Chem. **43** (1992) S. 207

[40] J.M. Kosterlitz, D.J. Thouless
J. Phys. C **6** (1973) S. 1181

[41] L.D. Landau, E.M. Lifschitz : *Lehrbuch d. theor. Physik V : Statistische Physik 1*
Akademie-Verlag Berlin (1987)

[42] H. Lange
Diplomarbeit, Universität Mainz, (1997)

[43] I. Langmuir
J. Am. Chem. Soc. **39** (1917) S. 1848

[44] I. Langmuir
J. Chem. Phys. **1** (1933) S. 756

[45] Jooyoung Lee, J.M. Kosterlitz
Phys. Rev. B **43**,4 (1990) S. 3265

[46] M. Lösche, E. Sackmann, H. Möhwald
Ber. Bunsenges. Phys. Chem **87** (1983) S. 848

[47] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller
J. Chem. Phys. **21** (1953) S. 1087

[48] H. Möhwald
Annu. Rev. Phys. Chem. **41** (1990) S. 441

[49] J.R. Morris, K.M. Ho
Phys. Rev. Let. **74**,6 (1995) S. 940

[50] V.T. Moy et al.
J. Phys. Chem. **90** (1986) S. 3198

[51] NAG Fortran Library Mark 17, Online-Dokumentation

[52] D.R. Nelson, B.I. Halperin
Phys. Rev. B **19** (1979) S. 2457

[53] W. Nolting : *Grundkurs : Theoretische Physik 4*
Zimmermann-Neufang, Ulmen (1991)

[54] M. Parrinello, A. Rahman
Phys. Rev. Lett. **45** (1980) S. 1196

[55] M. Parrinello, A. Rahman
J. Appl. Phys. **52** (1981) S. 7182

[56] M. Plischke, B. Bergersen : *Equilibrium Statistical Physics*
Prentice-Hall Int. (1989)

[57] X. Qiu et al.
Phys. Rev. Lett. **67** (1991) S. 703

[58] D.J. Rigby, R.J. Roe
J. Chem. Phys. **87** (1987) S. 7285

[59] M.N. Rosenbluth, A.W. Rosenbluth
J. Chem. Phys. **23** (1955) S. 356

[60] F. Schmid, H. Lange
J. Chem. Phys. **106**,9 (1996) S. 3757

[61] F. Schmid, C. Stadler, H. Lange : *MC-Simulation of Langmuir Monolayer Models*
in *Colloids and Surfaces A*;
Comp. Sim. Cond. Mat. Phys. X; Springer-Verlag, Heidelberg (1997)

[62] M.E. Schmidt, Seokmin Shin, S.A. Rice
J. Chem. Phys. **104**,5 (1996) S. 2114

[63] M.C. Shih et al.
J. Chem. Phys. **96**,2 (1992) S. 1556

[64] M.C. Shih et al.
J. Chem. Phys. **101**,10 (1994) S. 9132

[65] J.I. Siepmann, D. Frenkel
Mol. Phys. **75**,1 (1992) S. 59

[66] R.D. Smith, J.C. Berg
J. Coll. Int. Sci. **74** (1980) S. 273

[67] E. Teer et al.
J. Chem. Phys. **106**,5 (1997) S. 1913

[68] A.V. Tkachenko, Y. Rabin
Phys. Rev. Lett. **76**,14 (1996) S. 2527

[69] I. Vattulainen et al.
Comp. Phys. Comm. **86** (1995) S. 209

[70] N. Wilding, A.D. Bruce
J. Phys. Cond. Matt. **4** (1992) S. 3087

[71] A.P. Young
Phys. Rev. B **19** (1979) S. 1855

Index

A

Alkankette, 3, 7, 8, 10
    Länge der, 91
Amphiphile, 3
anharmonische Effekte, 112
Anisotropie, 8, 92
Ausdehnungskoeffizient, 111

B

Beschleunigung des Programms, 31–33
Bondlängen-Peaks, 18
Bondlängenpotential,
    siehe Potential, Bondlängen-
Bondordnung, 4, 78
Bondwinkelpotential,
    siehe Potential, Bondwinkel-

C

Clausius-Clapeyron-Gleichung, 47, 77
Cluster, 85, 98
Configurational Biased MC, 26–31

D

detailed balance, 27
Determinante, 43
Domänen, 19
Drucktensor,
    siehe Innerer Druck, Tensor des
dynamische Matrix, 39

E

effektive Monomere, 7, 10
Einheiten, 10, 52
Einrast-Effekt, 62
Entartung, 24, 62, 72, 82
Entkopplung, 61, 76
Entkopplungs-Mode, 48

F

Finite-Size-Effekte, 58, 84, 94–97
Flächenmoves,
    siehe Moves, Volumen-

G

Gasphase, 6, 81–86
Gibbs'sches Potential, 35, 41–44
Gitter
    rechteckig-zentriertes, 25
    reziprokes, 17, 38
Grundzustand, 50, 51, 74
Grundzustands-Analyse, 49

H

Hamiltonian
    effektiver, 12, 26, 106
    für $N\Pi T$-Ensemble, 105–107
    harmonischer, 36
harmonische Näherung, 36–37
herringbone-Ordnung, 4, 8
Hessematrix,
    siehe Kopplungsmatrix
hexatische Phasen, 5, 78
Hysterese, 59, 72, 77–80, 84

I

Innerer Druck
    Scherkomponenten des, 24
    Tensor des, 13
Integration
    thermodynamische, 44, 77, 90
Iteration des Grundzustands, 50

K

Kettenlängen-Abhängigkeit, 91
Kommensurabilität, 69
Koordinaten
    schiefwinklige, skalierte, 23
Kopplungsmatrix, 37, 43, 47
    Berechnung der, 109
    Determinante der, 43
Kumulante 4. Ordnung, 14, 96

L

Landau-Theorie, 5
Langmuir-Blodgett, 3
Langmuir-Monolagen, 3
Laplace-Transformation, 106

Lennard-Jones-Potential,
siehe Potential, Lennard-Jones-
Linked-Cell-Algorithmus, 32
liquid condensed, 5
liquid expanded, 5

M

Metropoliskriterium, 12, 27
Micelles, 98
Mittelwerte, 14
Modell, 7
Modellparameter,
siehe Potentialparameter
Modulation, 68–73
Moleküldirektoren, 13
Moves
kollektive, 25–31
Monomer-, 12
Translations-, 25–26
Volumen-, 12, 25, 82

N

Nachbarschafts-Tabelle,
siehe Verlet-Tabelle
Neigungshistogramm, 57
Neigungskorrelation, 13
Neigungsordnung, 4
Neigungswinkel, 13, 67
$NIIT$-Ensemble, 12

O

odd-even-Effekte, 92
Ordnungsparameter
der Neigungsordnung, 60
hexagonaler, 13
smektische, 4

P

Paarverteilungsfunktion, 15, 55
Parrinello-Rahman-Geometrie, 23
Periodizität, 17,
siehe Randbedingungen, periodische
Phasendiagramm
experimentelles, 4
großer Köpfe, 88–91
kleiner Köpfe, 64–67
synoptisches, 91
Positionsordnung, 4, 78
Potential
Bondlängen-, 8, 10, 11
Bondwinkel-, 8, 11
Lennard-Jones-, 8, 10
Soft Core-, 8
Potentialparameter, 10–11
Programm-Listing, 113–170

R

Randbedingungen
periodische, 12, 16, 17, 23, 25, 82, 94
re-entrance-Verhalten, 91
reziprokes Gitter,
siehe Gitter, reziprokes
Rosenbluth-Faktor, 28
Rosenbluth-Methode, 28
Rotator-Phasen, 8

S

Satelliten-Peaks, 71
smektische Phasen, 4
Snapshot
LC-NN-Phase, 51
LC-NNN-Phase, 87
LC-Phase moduliert, 87
LC-U-Phase, 88
LE-Phase, 88
sehr große Köpfe, 99, 100
Soft Core-Potential,
siehe Potential, Soft Core-
Spreizdruck, 4,
siehe Innerer Druck
Streifenbreite, 69, 71, 72
Strukturfaktor, 17–22
geneigter Phasen, 18
modulierter Phasen, 70
Pulver-Mittel, 19

T

Tabellarisierung
des LJ-Potentials, 32, 110
Thermodynamische Integration,
siehe Integration, thermodynamische
tilt order, 4

U

Überstruktur, 69

V

Verlet-Tabelle, 30
Virialtheorem, 13
Voronoi-Konstruktion, 13, 54, 73

W

Wasser-Luft-Grenzfläche, 7

Danksagung

Ich danke Herrn Professor Dr. Kurt Binder für die interessante Themenstellung und die Aufnahme in seine Arbeitsgruppe, sowie Frau Dr. Friederike Schmid für die gute, freundliche Betreuung und die zahlreichen, wertvollen Anregungen.

Im weiteren gilt mein Dank dem DFG-Graduiertenkolleg „Physik und Chemie supramolekularer Systeme“ für das gewährte Stipendium und dem „Materialwissenschaftlichen Forschungszentrum“ an der Universität Mainz für die finanzielle Unterstützung.

Ohne die Rechenleistungen, die das „Zentrum für Datenverarbeitung“ der Universität Mainz und das „Regionale Hochschulrechenzentrum Kaiserslautern“ zur Verfügung gestellt haben, wären die vorliegenden Simulationen nicht möglich gewesen – daher gilt auch ihnen mein Dank.

Schließlich möchte ich mich bei allen Mitgliedern der Arbeitsgruppe „Theorie der kondensierten Materie“ in Mainz für das ausgezeichnete Arbeitsklima und die anregenden Diskussionen bedanken.