Field Theories for Copolymer Blends:
Self-Consistent Approaches and
Monte Carlo Simulations

Dissertation
zur Erlangung des Doktorgrades
an der Fakultät für Physik
der Universität Bielefeld

vorgelegt von
Dominik Düchs

Februar 2003
## Contents

1 Introduction

2 Self-Consistent Field Theory (SCFT) for Gaussian Chains
   2.1 The Gaussian chain model
   2.1.1 The random walk
   2.1.2 The Gaussian chain model
   2.1.3 Discussion of the Gaussian model
   2.2 SCFT for Gaussian chains
      2.2.1 Canonical ensemble
      2.2.2 Grand canonical ensemble
   2.3 The phase diagram for ternary AB+A+B melts

3 Entropy-Induced Smectic Phases in Rod-Coil Polymers
   3.1 Introduction
   3.2 Theory
      3.2.1 General three-dimensional theory
      3.2.2 One-dimensional theory
   3.3 Results
   3.4 Summary and conclusions

4 Fluctuations in the Ternary AB+A+B System
   4.1 Introduction
   4.2 Related research
   4.3 The field-theoretic model
   4.4 The field-theoretic Monte Carlo method
      4.4.1 $W_-$ fluctuations
      4.4.2 $W_+$ fluctuations
   4.5 Symmetries of the complex partition function
   4.6 The order-disorder transition
   4.7 The demixing transition
   4.8 The fluctuation-corrected phase diagram at $C=50$
   4.9 Summary and conclusions

5 The Formation and Structure of the Microemulsion Phase
   5.1 Introduction
   5.2 The formation of the microemulsion phase
List of Figures

1.1 Schematic drawing of an AB diblock copolymer. .................... 2
1.2 Schematic drawing of the lamellar phase in a ternary AB+A+B system. Solid lines correspond to homopolymers A and B, and dashed lines to AB copolymers. ................................. 3

2.1 Mean-field phase diagram for a ternary AB+A+B blend with \( \alpha = 0.2 \), as obtained from a Fourier-space implementation of SCFT. \( 2\phi \) denotes a region of two-phase coexistence between an A-rich and a B-rich phase, \( 3\phi \) one of three-phase coexistence between an A-rich, a B-rich, and a lamellar phase. 19
2.2 Mean-field phase diagram for ternary AB+A+B blend with \( \alpha = 0.5 \). Notation as in Fig. 2.1. .......................... 19
2.3 Lamellar periodicities according to the mean-field equations for \( \chi N = 12.5 \). The inbox shows the end points of these graphs at the boundary of the three-phase (L+A+B) coexistence region. The gap in the \( \chi N = 11.5 \) curve corresponds to a region in the disordered phase. ............... 20
2.4 A and B densities in a lamellar solution of the mean-field equations at \( \phi_H = 0.5, \chi N = 12.5, L_{MF} = 3.92 \). ...................... 20

3.1 Liquid-crystalline phases of rod-like colloids (schematic): (a) isotropic, (b) nematic, (c) smectic-A, (d) smectic-C. ..................... 22
3.2 Liquid-crystalline phases of rod-coil block copolymers (schematic): (a) isotropic, (b) nematic, (c) smectic-A_1 (monolayer), (d) smectic-A_2 (bilayer). Rods are black, coils are red. ......................... 23
3.3 Density profiles of a smectic configuration with \( f = 2/3, \xi_{\text{rigid}} = 10, \xi_{\text{flex}} = 0.1, G = 20 \), and period \( 1.3L \). The solid, dotted, and dashed lines correspond to \( \sqrt{4\pi \phi_0}, \sqrt{4\pi \phi_{\text{rigid},0}}, \) and \( \sqrt{4\pi \phi_{\text{flex},0}} \), respectively. ................. 31
3.4 Orientational order parameters \( \bar{P}_{1,\text{rigid}} \) (dotted line) and \( \bar{P}_{2,\text{rigid}} \) (solid line) for the same case as in Fig. 1. ......................... 31
3.5 Order parameters \( \bar{P}_{1,\text{flex}} \) (dotted line) and \( \bar{P}_{2,\text{flex}} \) (solid line) for the same case as in Fig. 1. .......................... 32
3.6 Density profiles as in Fig. 1 but for \( f = 2/3, G = 30 \), and period \( 1.24L \). .......................... 32
3.7 Orientational order parameters of rigid segments for the same case as in Fig. 4. .................................................. 33
3.8 Orientational order parameters of flexible segments for the same case as in Fig. 4. .................................................. 33
3.9 Phase diagram for rod-coil copolymers with $\xi_{\text{rigid}} = 10$, $\xi_{\text{flex}} = 0.1$. The isotropic and nematic phases are separated by a narrow coexistence region bounded by the solid and dash-dotted curves. The boundary between smectic and nematic phases is second order, indicated by the dashed curve. ... 34

3.10 Smectic order parameter vs. $W$. ... 35

4.1 Experimental phase diagram of the PDMS-PEE/PDMS/PEE blend with $\alpha \approx 0.2$ [73] - Reproduced by permission of The Royal Society of Chemistry. ... 40

4.2 Re($H$) in a simulation of $W_-$ and $W_+$ against number of Monte Carlo steps in $W_-$ for $C = 50$, $\chi N = 12$, $z = 2$ (grand canonical ensemble), started from a configuration equilibrated in $W_-$. Each $W_-$ step alternates with ten steps in $\tilde{\omega}_+$. ... 48

4.3 $\Delta H^*$ (difference of $\Delta H$ and a reference value at $\tilde{\omega}_+$ fluctuations switched off) in the same simulation as in Fig. 4.2. ... 49

4.4 The argument, $A$, of the cosine in the reweighting factor in the same simulation as in Fig. 4.2. ... 49

4.5 Autocorrelation function of $W_-$ in the same simulation as in Fig. 4.2. ... 50

4.6 Correlation functions $C[I, I]$ (dashed), $C[\tilde{\omega}_+, \tilde{\omega}_+]$ (dashed-dotted), $C[I, W_-]$ (left inset), and $C[\tilde{\omega}_+, W_-]$ (right inset) in the same simulation as in Fig. 4.2. Note the different scale from Fig. 4.5. ... 50

4.7 Snapshots at $C=50$, $\phi_H = 0$, and $\chi N = 11.1, 11.2, 11.3, 11.4$, and 11.5. White points: $0 < \tilde{\phi}_A < 0.49$, red points: $0.49 < \tilde{\phi}_A < 0.51$, black points: $0.51 < \tilde{\phi}_A \leq 1$. All runs were started from disordered configurations. ... 54

4.8 Direction persistence as measured by the $\Lambda$ parameter vs. $\chi N$ for different homopolymer volume fractions $\phi_H$ at the order-disorder transition in simulations on $32\times32$ lattices. For $\phi_H = 0$ and 0.2, the discretization $dx = 0.625$ was used, otherwise $dx = 0.78125$. ... 54

4.9 $\bar{F}_2$ (solid) and $\bar{F}_4$ (dashed) vs. $\chi N$ for different homopolymer volume fractions $\phi_H$ at the order-disorder transition in the same simulations as in Fig. 4.8. ... 55

4.10 Snapshots ($48\times48$ lattice, L-started) at $C = 50$, $\phi_H = 0.7$ and $\chi N = 12.2, 12.3, 12.4$, and 12.5 (from left to right). ... 56

4.11 Snapshots ($48\times48$ lattice, L-started) at $C = 50$, $\phi_H=0.8$ and $\chi N = 12.5, 12.7$, and 13.0 (from left to right). ... 56

4.12 Snapshots ($48\times48$ lattice, L-started) at $C = 50$, $\phi_H=0.85$ and $\chi N = 12.5, 12.7, 13.0, 13.2$, and 13.4 (from left to right). ... 56

4.13 Direction persistence, $\Lambda$, at $\phi_H = 0.7$ in simulations on a $48\times48$ lattice that started from lamellar configurations. ... 57

4.14 $\bar{F}_2$ at homopolymer volume fraction $\phi_H = 0.7$ for different $\chi N$ in simulations on a $48\times48$ lattice that started from lamellar configurations. ... 57

4.15 $\Lambda$ at $\phi_H = 0.8$ in simulations on a $48\times48$ lattice that started from lamellar configurations. ... 58

4.16 $\bar{F}_2$ at homopolymer volume fraction $\phi_H = 0.8$ for different $\chi N$ in simulations on a $48\times48$ lattice that started from lamellar configurations. ... 58

4.17 $\Lambda$ at $\phi_H = 0.85$ in simulations on a $48\times48$ lattice that started from lamellar configurations. ... 59
4.18 $F_2$ at homopolymer volume fraction $\phi_H = 0.85$ for different $\chi N$ in simulations on a 48×48 lattice that started from lamellar configurations.

4.19 Difference of A and B monomer densities, $\Delta \phi$, vs. $\chi N$ in a binary system of A and B homopolymers in the grand canonical ensemble.

4.20 Difference in A and B monomer densities, $\Delta \phi$, vs. relative homopolymer fugacity, $z$, in the grand canonical ensemble.

4.21 Homopolymer concentration, $\phi_H$, vs. $\chi N$ at $z = 1$ across the order-disorder transition. The dotted line indicates the $\chi N$ coordinate of the ODT $\pm 0.1$.

4.22 Homopolymer concentration, $\phi_H$, vs. $\chi N$ at $z = 2$ across the order-disorder transition. The dotted line indicates the $\chi N$ coordinate of the ODT $\pm 0.1$.

4.23 The fluctuation-corrected phase diagram at $C=50$. The dotted lines indicate the mean-field result of Fig. 2.1. The order-disorder transition is weakly first-order, but the coexistence region is too small to be displayed.

5.1 Substructure of the microemulsion. $D\mu E$: defect-driven, $G\mu E$: genuine microemulsion morphology, $2\phi$: phase-separated region.

5.2 Snapshots of D-started runs at $C = 50$, $\chi N = 12.5$, $\phi_H = 0.74$, 0.78, 0.8, and 0.82 (first row from left), and $\phi_H = 0.84$, 0.86, 0.88, and 0.9 (second row from left).

5.3 Preferential length scale, $L_0$, at $\chi N = 12.5$, averaged over the equilibrated parts of simulations on a 48×48 lattice. Filled and empty dots correspond to disordered and lamellar initial conditions, respectively. For error bars, see Fig. 5.4. The solid line shows the lamellar periodicities according to mean-field theory (cf. Fig. 2.1).

5.4 Difference of the preferential length scales, $L_0$, from the corresponding mean-field periodicities at $\chi N = 12.5$ (equilibrated averages over the same simulations as in Fig. 5.3).

5.5 Curvature diameter, $D_C$, at $\chi N = 12.5$ vs. homopolymer volume fraction, $\phi_H$ (equilibrated averages over the same simulations as in Fig. 5.3). Filled and empty dots correspond to disordered and lamellar initial conditions, respectively. The solid line shows the lamellar periodicities according to mean-field theory.

5.6 Snapshots of D-started runs at $C = 50$, $\chi N = 12$, $\phi_H = 0.74$, 0.78, 0.8, and 0.82 (first row from left), and $\phi_H = 0.84$, 0.86, 0.88, and 0.9 (second row from left).

5.7 Preferential length scale, $L_0$, at $\chi N = 12$, averaged over the equilibrated parts of D-started simulations on a 48×48 lattice. The solid line shows the lamellar periodicities according to mean-field theory (cf. Fig. 2.1).

5.8 Difference of the preferential length scales, $L_0$, from the corresponding mean-field periodicities at $\chi N = 12$ (equilibrated averages over the same simulations as in Fig. 5.7).

5.9 Curvature diameter, $D_C$, at $\chi N = 12$ vs. homopolymer volume fraction, $\phi_H$ (equilibrated averages over the same simulations as in Fig. 5.7). The solid line shows the lamellar periodicities according to mean-field theory.
5.10 Snapshots of D-started runs at $C = 50$, $\chi N = 11.5$, $\phi_H = 0.82$, 0.83, 0.84, and 0.85 (first row from left), and $\phi_H = 0.86, 0.87, 0.88$, and 0.89 (second row from left). ................................................................. 72

5.11 Preferential length scale, $L_0$, at $\chi N = 11.5$, averaged over the equilibrated parts of D-started simulations on a $48 \times 48$ lattice. The solid line shows the lamellar periodicities according to mean-field theory (cf. Fig. 2.1). .......... 72

5.12 Difference of the preferential length scales, $L_0$, from the corresponding mean-field periodicities at $\chi N = 11.5$ (equilibrated averages over the same simulations as in Fig. 5.11). ................................................................. 73

5.13 Curvature diameter, $D_C$, at $\chi N = 11.5$ vs. homopolymer volume fraction, $\phi_H$ (equilibrated averages over the same simulations as in Fig. 5.11). The solid line shows the lamellar periodicities according to mean-field theory. 73

5.14 $A_{3/4}$ at $C = 50$ and $\chi N = 12.5$. .............................................. 75

5.15 $A_{3/4}$ at $C = 50$ and $\chi N = 12$. ................................................. 76

5.16 $A_{3/4}$ at $C = 50$ and $\chi N = 11.5$. ............................................... 76

5.17 $U$ at $C = 50$ and $\chi N = 12.5$. ............................................... 77

5.18 $U$ at $C = 50$ and $\chi N = 12$. .................................................. 78

5.19 $U$ at $C = 50$ and $\chi N = 11.5$. .................................................. 78

5.20 Euler characteristic, $\chi_E$, vs. inflation radius, $r$, for a sample configuration at $\chi N = 12.5$ and $\phi_H = 0.82$. ................................................................. 79

5.21 $L_{\bar{\chi}}$ vs. homopolymer volume fraction, $\phi_H$. Dotted lines correspond to mean-field periodicities. ................................................................. 80

5.22 $A_{\bar{\chi}}$ vs. homopolymer volume fraction, $\phi_H$. .............................. 80

5.23 Snapshots of D-started runs at $C = 50$, $\phi_H = 0.82$, and $\chi N = 12.5, 12, 11.5$, and 11 (first row from left), and $\chi_N = 10.5, 10, 9, and 8$ (second row from left). ................................................................. 81

5.24 Equilibrated averages of $F_0(q)$ over the same simulations as in Fig. 5.23. 82

5.25 Equilibrated averages of $\bar{F}_0$. ................................................... 82

5.26 1d cross section of $\bar{A}_\chi$ at $\phi_H = 0.7$ and $\chi N = 12.5$ (black line), mean-field profile with same parameters (red line). ................................................... 84

5.27 1d cross sections of $\bar{A}_\phi$ at $\phi_H = 0.82$. ................................ 84

5.28 Structure factor of the lamellar phase at $C=50$, $\phi_H = 0.7$, and $\chi N = 12.5$. 85

5.29 Structure factor of the microemulsion phase at $C=50$, $\phi_H = 0.82$, and $\chi N = 12.5$. ................................................................. 86

5.30 Structure factor of the disordered phase at $C=50$, $\phi_H = 0.82$, and $\chi N = 11$. 87

5.31 Structure factor of the disordered phase at $C=50$, $\phi_H = 0.82$, and $\chi N = 10$. 88

A.1 Slab decomposition of a lattice, left, and geometric decomposition, right. 96

A.2 Relative CPU time per pixel (in an arbitrary time unit) vs. number of pixels in a sublattice for parallel DuFort-Frankel on a Cray T3E-1200. 96

A.3 Local contributions to the circumference. The current (black) pixel from step (3) here is painted blue. All diagrams are modulo $\pi/2$ rotations. 99
Chapter 1

Introduction

Polymer science has by now evolved into a major field of research at the nexus of condensed matter physics and chemical engineering [1]. Polymers are long chains of linearly aligned subunits called monomers that are joined to one another by chemical bonds. If a polymer contains only one kind of monomer, it is called a homopolymer. Conversely, if it contains more than one monomer species, it is called a copolymer.

Melts of one or more kinds of polymers have been demonstrated over the past few decades to exhibit a wealth of diverse phases whose geometric properties make them interesting systems not only for condensed matter research, but for industrial applications, as well. It is understood that mixtures of pure homopolymers with different constituent monomers (say, A and B) will not reach equilibrium but instead phase-separate on a macroscopic scale due to repulsive interactions generally found between unlike monomers, which build up a high interfacial tension at the domain boundaries [2, 3]. The situation here is analogous to the incongruous behaviors of water and oil. In both cases, however, there exist substances whose addition to the system allows for this predicament to be overcome. For water and oil, the task is accomplished through the use of amphiphiles like soap. In the polymer scenario, the magic bullet is called ‘block copolymer.’

Block copolymers consist of a string of A monomers at one end, and one of B monomers at the other (Fig. 1.1). The two parts are joined by chemical bonds similar in strength to the inter-A and inter-B bonds, such that separation is prevented. Molten block copolymers, whether alone or in mixtures with their corresponding homopolymers, do exhibit microphase separation, however, at sufficiently strong repulsive interactions, thus forming regular patterns of varying complexity [4].

It should be noted that apart from block copolymers, other compatibilizers (like graft or random copolymers) may also be efficiently put to use. However, these alternative substances are not examined in the present thesis. Compatibilizers in general are key to creating polymer alloys with morphological, mechanical, conductive, or otherwise desirable properties. In particular, (bi-)continuous morphologies like the microemulsion discussed below and in subsequent chapters of this thesis, lend themselves to improving percolation-like properties like the conductivity or gas-permeability of the alloy, if one of the original components possesses these features. Moreover, the mechanical stiffness, as measured by the strain at break and toughness index, has been shown to greatly improve with bicontinuous morphologies [5].

Self-consistent field theory (SCFT) is a powerful tool to examine the formation of
Chapter 1. Introduction

these phases [6, 7, 8, 9, 10]. It is built around the concept of a test polymer which moves in the mean field generated by the other polymers in the system. The polymers are represented as smooth space curves. In pure, symmetric AB copolymer melts, an order-disorder transition to a lamellar phase is observed. Upon adding equal amounts of A and B homopolymers, the lamellar periodicity is increased: the homopolymers place themselves within the microdomains defined by the midpoints of the copolymers and swell the latter (Fig. 1.2). For the incompressible ternary (copolymer AB + homopolymers A+B) system, mean-field SCFT predicts a first-order transition to a two-phase region at a threshold homopolymer chemical potential. However, it is known from experimental observations that this transition is an artifact of the mean-field approximation. Instead, one observes a bicontinuous microemulsion in the vicinity of the presumed first-order transition (termed the “microemulsion channel”) close to where mean-field theory predicts a Lifshitz critical point [11].

In the research of many-body systems, computer simulations have been established in recent decades as a powerful investigative tool [12, 13]. Rooted in the concepts of thermodynamics and statistical physics, simulation techniques on computers have made possible the solution of many hitherto untractable problems. The techniques used in this thesis – field theory, Monte Carlo simulation, solution schemes for differential equations – are also utilized in other branches of theoretical physics, notably elementary particle theory [16].

Before embarking on any theoretical research of a many-body system, whether simulational or analytical, one must first define the underlying model. As far as polymers are concerned, the degree of coarse-graining of the model must be decided. Atomistic models are generally used only if the chemical properties of the polymer are to be examined. If one is interested in mesoscopic properties, the chain structure dominates such that in particle-based coarse-grained models, 2-3 monomers may be grouped together to form effective monomers. These effective monomers and the bonds between them are then simulated in boxes with continuous or discrete coordinates. Simulations based on Monte Carlo algorithms most commonly make this type of approach; a popular model on the lattice is the bond fluctuation model [14, 15].

By contrast, in this thesis, the internal structure of the chains is taken to be continuous. The position of a test polymer is determined by calculating the propagator of a point particle in an external potential field, following the same formalism as in non-relativistic quantum mechanics. The propagators are governed by a diffusion equation. In the case of perfectly flexible polymers, the analogy extends even further because the diffusion equation has the same form as the Schrödinger equation, which governs the time evolution of scalar wave functions. The corresponding polymer model is called the Gaussian model.
The appropriate model for semiflexible polymers is the *wormlike chain* model [17, 18].

The wormlike chain model can be used to extend the applicability of SCFT to rod-coil polymers, in which the rods are rigid and the coils flexible. One aim of the present thesis is to introduce such an extension of SCFT and demonstrate that in rod-coil polymers, entropic effects suffice to give rise to a partial bilayer smectic-A, i.e., a liquid-crystalline lamellar phase.

Back with the Gaussian model, it is another aim of this thesis to improve the mean-field predictions of SCFT by systematically including thermal field fluctuations. This is essential to examining the aforementioned bicontinuous microemulsion in ternary blends. To this end, I transform the partition function of the ternary system to a Monte Carlo-integrable form for the composition fluctuations on a two-dimensional lattice. Each step in a Monte Carlo program then requires an iteration to solve the self-consistent field equation for the field governing the total density, and in turn, each step in an iteration requires the solution of a diffusion equation. Parallelized solution schemes for the diffusion equation have been devised and implemented. To analyze the resulting configurations, novel methods for image processing have been developed.

This thesis is structured as follows: after this Introduction, the framework of self-consistent field theory will be laid out for the Gaussian chain model, and the mean-field equations will be derived in Chapter 2. The occurrence and properties of entropy-induced smectic phases in rod-coil polymers will form the topic of Chapter 3. Chapter 4 lays the theoretical foundations for including fluctuations in SCFT and applies them to the phase transitions in the ternary model system. Further, I present evidence suggesting the insignificance of total density fluctuations in this incompressible system. In Chapter 5, the formation and structure of the microemulsion region within the disordered phase will be analyzed. I will summarize the main results of this thesis in Chapter 6. The numerical methods that have been used in this work are presented in Appendix A, and the program listings given in Appendix B.
Chapter 2

Self-Consistent Field Theory (SCFT) for Gaussian Chains

In this chapter, we shall first introduce the Gaussian model for flexible chains and then use this model to derive, in a real-space version, the self-consistent field theory of incompressible polymer melts for the mean-field case. First introduced by Edwards [19], then further developed by Helfand [20, 21, 22, 6], it was later used by Scheutjens and Fleer [23, 24], and more recently by Matsen [25, 26, 4, 27, 28, 29, 30], who applied a Fourier-space version of SCFT to various problems. Approximations for the weak- and strong-segregation limits were derived by Leibler [31] and Semenov [32], respectively.

SCFT is able to predict phase diagrams for melts consisting of any number of different polymer species. In general, its predictions are in good agreement with experiments in regions of the phase diagram that are somewhat removed from phase boundaries, whereas in close vicinity to phase transitions, thermal fluctuations not captured by the theory, in particular those of the monomeric composition, gain crucial importance.

In a mean-field context, the examined phases exhibit a high degree of regularity. Therefore, a Fourier space representation may appear advantageous. However, if SCFT is to be extended to include fluctuations, the focus is on deviations from regular phase patterns, or even a complete breakdown of a regular pattern, as in the case of the bicontinuous microemulsion. The real-space representation of the theory given here should therefore be better suited for these problems. Our method of extending this treatment to fluctuations will be presented in Chapter 4.

2.1 The Gaussian chain model

2.1.1 The random walk

Consider a lattice with lattice constant $b$ and number of nearest neighbors (coordination number) $z$. A random walk on this lattice consists of $N$ consecutive steps in any of the $z$ possible directions. On a $d$-dimensional rectangular lattice, $z = 2d$. (A good account of these basic facts is found in [94].) The end-to-end vector $\mathbf{R}$ is

$$\mathbf{R} = \sum_{n=1}^{N} \mathbf{r}_n,$$  \hspace{1cm} (2.1)
where the $\mathbf{r}_n$ are the bond vectors. For symmetry reasons, $\langle \mathbf{R} \rangle = 0$, but

$$
\langle \mathbf{R}^2 \rangle = \sum_{n=1}^{N} \sum_{m=1}^{N} \langle \mathbf{r}_n \cdot \mathbf{r}_m \rangle = \sum_{n=1}^{N} \langle \mathbf{R}_n^2 \rangle = Nb^2,
$$

(2.2)

since there is no correlation between the directions for different bond vectors (i.e., if $n \neq m$). Writing $b_i$ for the possible bond vectors, we obtain for the probability of the polymer end being at $\mathbf{R}$ after $N$ steps:

$$
P(\mathbf{R}, N) = \frac{1}{z} \sum_{i=1}^{z} P(\mathbf{R} - b_i, N - 1).
$$

(2.4)

For large $N$ and $|\mathbf{R}| \gg |b_i|$, we may expand $P(\mathbf{R}, N)$ in terms of a continuous $N$ and $\mathbf{R}$. Let $\mathbf{R}$ be at the origin for $N=0$. After some algebra, we find (in three dimensions)

$$
P(\mathbf{R}, N) = \left( \frac{3}{2\pi Nb^2} \right)^{3/2} \exp \left( -\frac{3\mathbf{R}^2}{2Nb^2} \right).
$$

(2.5)

i.e., a Gaussian distribution. This result is independent of the coordination number, $z$, and thus holds for the off-lattice ‘freely jointed’ chain model, as well, in which the number of bond directions is not limited.

### 2.1.2 The Gaussian chain model

We have seen that the continuous limit of the random walk yields a Gaussian distribution for the end-to-end vector. From the properties of the Gaussian integral, it can be derived that the distribution for the distance $|\mathbf{R}_n - \mathbf{R}_m|$ between any two chain segments is Gaussian, as well. The Gaussian chain model assumes that every bond vector $\mathbf{r}_n \equiv \mathbf{R}_n - \mathbf{R}_{n-1}$ can be stretched to a certain extent, following a Gaussian distribution:

$$
p(\mathbf{r}_n) = \left( \frac{3}{2\pi b^2} \right)^{3/2} \exp \left( -\frac{3\mathbf{r}_n^2}{2b^2} \right).
$$

(2.6)

Hence, the probability distribution for the set of segment position vectors $\{\mathbf{R}_n\} \equiv (\mathbf{R}_0, \mathbf{R}_1, \ldots, \mathbf{R}_N)$ is given by

$$
P(\{\mathbf{R}_n\}) = \left( \frac{3}{2\pi b^2} \right)^{3N/2} \exp \left( -\frac{3}{2b^2} \sum_{n=1}^{N} (\mathbf{R}_n - \mathbf{R}_{n-1})^2 \right).
$$

(2.7)

If the monomer index, $n$, is converted to a continuous contour variable, $s$, we obtain the Wiener distribution:

$$
P(\{\mathbf{R}(s)\}) = \left( \frac{3}{2\pi b^2} \right)^{3N/2} \exp \left( -\frac{3}{2b^2} \int_0^N ds \left( \frac{\partial \mathbf{R}(s)}{\partial s} \right)^2 \right).
$$

(2.8)
2.1. The Gaussian chain model

In this sense, the Gaussian chain model provides a continuous generalization of the discrete random walk model. The statistical properties of the Gaussian chain do not depend on the local structure but only on its effective bond length (or ‘statistical Kuhn length’), \( b \), and its length, \( N \). They are therefore invariant under rescaling of these variables according to:

\[
N \rightarrow N/\lambda, \quad b \rightarrow b\sqrt{\lambda},
\]

which can be used to confirm the exponent \( \nu = 1/2 \) for the dependence of the end-to-end vector on the chain length

\[
\langle R_N^2 \rangle^{1/2} \propto N^{1/2}.
\]

Often, the ‘radius of gyration’ is a more convenient quantity to measure the size of a polymer. It is defined as the root mean square distance between any two segments:

\[
R_g^2 = \frac{1}{2N^2} \sum_{n=1}^{N} \sum_{m=1}^{N} \langle (R_n - R_m)^2 \rangle,
\]

which is equivalent to the root of the mean squared distance between the segments and the center of mass of the polymer. In \( d \) dimensions, it can be shown that

\[
R_g^2 = \frac{1}{2d} N b^2 = \frac{1}{2d} \langle R_N^2 \rangle.
\]

2.1.3 Discussion of the Gaussian model

The Gaussian model has enjoyed great popularity in the past and continues to do so, mainly owing to its mathematical simplicity. Before we go on to employ it for the purposes of SCFT, we should reflect on its properties and point out its limitations:

- It represents ideal chains in the continuous limit. Ideal chains are infinitely flexible, and segments may overlap with each other. The exponent \( \nu = 1/2 \) is only true for ideal chains. A model that prohibits this overlap is the self-avoiding walk (SAW), for which \( \nu \) has a different value (with mean-field arguments one gets: \( \nu = 3/5 \), from computer simulations or experiments: \( \nu \approx 0.588 \)).

- In dense melts of strongly interacting polymers, which are considered here, chains have many contacts with other chains in all directions. The resulting superposition of all excluded-volume interactions thus effectively reduces to an inconsequential constant potential term, and the chains become ideal [1].

- In dilute melts, there is an interplay between excluded-volume interactions on the one hand and temperature-dependent repulsive interactions of polymer segments with solvent particles on the other. Solvents for which these two interactions cancel out are called Θ solvents. Thus, at the Θ temperature, flexible chains display ideal behavior, i.e., \( \nu = 1/2 \), and the Gaussian model applies.

- The Gaussian model reflects perfectly flexible chains. It does not feature a bending penalty (only an entropic stretching penalty) and has no orientation dependence. However, semiflexible polymers, for which the Kratky-Porod [17, 18], or wormlike chain, model is appropriate, effectively become Gaussian for very large ratio of the chain and persistence lengths.
2.2 SCFT for Gaussian chains

Consider a melt of monodisperse polymers contained in a fixed volume V. Let us write:

\[ n_j \equiv \text{number of chains of type } j \]
\[ N_j := \rho_0 V C_j \] with \[ N_j \equiv \text{number of monomers in a chain of type } j \text{ (polymerization index)} \]
\[ V C_j \equiv \text{volume of a chain of type } j \]

The following derivation of SCFT for Gaussian chains is for a ternary blend consisting of diblock copolymers AB and homopolymers A and B. An adaptation to other constellations is straightforward, including to systems with more than two monomer species. Monomers of type A and B shall occupy the same volume, \( \rho_0 \). In the previous section, we saw that the Gaussian distribution is the continuous limit of a random walk on an arbitrary lattice. Although its contour length is subject to statistical fluctuations, we may introduce a contour variable, \( s \), which shall run from 0 (starting point) to \( N_j \) (end point).

In a melt, we will find many intertwined Gaussian chains forming a fluid, with each chain contributing to the overall monomer densities. For this fluid to be incompressible, and normalizing the overall relative density to unity, we define the relative density operators \( \hat{\phi}_{A,B} \):

\[ \hat{\phi}_{\alpha}(r) := \frac{1}{\rho_0} \sum_j \sum_{i,j=1}^{n_j} \int_0^{N_j} ds \delta(r - R_{ij}(s)) \gamma_{\alpha,j}(s), \] \hspace{1cm} (2.13)

where

\[ \rho_0 := \frac{\text{number of monomers}}{V} = \frac{\Sigma_i n_i N_i}{V}. \] \hspace{1cm} (2.14)

Note that in order for the melt to be incompressible, the chains have to occupy an intrinsic volume, \( V C_j \), which corresponds to the volume per monomer and is related to our choice of \( \rho_0 \). Moreover,

\[ \gamma_{\alpha,j}(s) := \begin{cases} 0, & \text{portions of } j \text{ that contain no } \alpha \\ 1, & \text{portions of } j \text{ that contain } \alpha \end{cases}. \] \hspace{1cm} (2.15)

The interaction energy functional is taken to be

\[ V_{\text{inter}} = k_B T \chi \rho_0 \int dr \hat{\phi}_A(r) \hat{\phi}_B(r), \] \hspace{1cm} (2.16)

where \( \chi \) is the Flory-Huggins interaction parameter [1]. In most cases, \( \chi \) is positive, corresponding to repulsion between unlike monomers. Note that Flory and Huggins had introduced their interaction term

\[ E_{\text{site}} = k_B T \chi \phi_A(r) \phi_B(r) \] \hspace{1cm} (2.17)

in a lattice context, where the \( \phi \)'s were the relative densities at a discretized site, \( r \). The entire energy of the system was thus the sum over all energies per site. The energy functional (2.16) is a continuous version of this formula. Furthermore, both formulae require the densities to be normalized, i.e., \( \int dr (\hat{\phi}_A(r) + \hat{\phi}_B(r)) = V \).
2.2. SCFT for Gaussian chains

2.2.1 Canonical ensemble

As a starting point in the derivation of the theory, we consider the partition function of the system. By convention, we shall put $k_B T \equiv 1$. In the canonical ensemble, the partition function is given by

$$Z_C = \left[ \prod_j \frac{1}{n_j!} \int D_j \{ \cdot \} \right] \exp \left[ -\chi \rho_0 \int \text{d}r \frac{\phi_A(r) \phi_B(r)}{\phi_A(r) + \phi_B(r) - 1} \right]$$

with

$$\int D_j \{ \cdot \} \equiv \int D_j \{ R(\cdot) \} \cdot P \{ R(\cdot) \},$$

where

$$P \{ R(\cdot) \} = N \exp \left[ -\sum_\alpha \frac{3}{2b_\alpha^2} \int_0^{N_j} \text{d}s \left| \frac{\text{d}R(s)}{\text{d}s} \right|^2 \gamma_{\alpha, j}(s) \right]$$

is the Wiener statistical weight for a particular conformation of the system with corresponding Kuhn lengths, $b_\alpha$, of each monomer species. Here, we shall put $b_\alpha = b$ for all $\alpha$. The delta functional is introduced to implement the incompressibility constraint. The partition function is thus a functional, or path, integral over all conformations of the Gaussian chains with constant density and is weighted with two factors: the energy functional (2.16) and the statistical weight for the conformations. Let us henceforth write:

$$\left[ \prod_j \frac{1}{n_j!} \int D_j \{ \cdot \} \right] =: \int D \{ \cdot \}$$

(2.21)

to obtain a more compact formula for the partition function:

$$Z_C = \int D \{ \cdot \} \exp \left[ -\chi \rho_0 \int \text{d}r \frac{\phi_A(r) \phi_B(r)}{\phi_A(r) + \phi_B(r) - 1} \right].$$

(2.22)

Now, we make use of the identity:

$$1 = \int D \{ \phi_\alpha \} \delta \left( \phi_\alpha - \phi_\alpha \right)$$

$$= \int D \{ \phi_\alpha \} \int_\infty \mathcal{D} \{ (\rho_0 \omega_\alpha) \} \exp \left[ \int \text{d}r (\rho_0 \omega_\alpha)(r) \left( \phi_\alpha(r) - \phi_\alpha(r) \right) \right]$$

(2.23)

with the standard representation of the delta functional on the right-hand side. We insert this identity twice into the partition function (once for each monomer species), thereby introducing the scalar functions $\phi_\alpha$ and auxiliary fields $\omega_\alpha$. The factor $\rho_0$ is needed if we are to interpret $\omega_\alpha$ as the potential seen by monomers of type $\alpha$ and which is generated by all other monomers in the system. From now on, we will write $\phi_\alpha \equiv \phi_\alpha(r)$, $\hat{\phi}_\alpha \equiv \hat{\phi}_\alpha(r)$, $\omega_\alpha \equiv \omega_\alpha(r)$, etc. Note that the absolute density is

$$\rho_\alpha = \rho_0 \phi_\alpha.$$

(2.24)
Therefore, we might as well have written (2.23) as

\[ 1 = \int \mathcal{D}\{\rho_\alpha\} \int_\infty \mathcal{D}\{\omega_\alpha\} \exp \left[ \int \text{d}r \: \omega_\alpha(r) (\rho_\alpha(r) - \hat{\rho}_\alpha(r)) \right]. \tag{2.25} \]

However, we should stick to (2.23) because this will allow us to replace the operators \(\hat{\phi}_\alpha\) with the functions \(\phi_\alpha\) elsewhere in the partition function. Spelling out monomer species A and B, we obtain:

\[
Z_C = \int \mathcal{D}\{\cdot\} \int \mathcal{D}\phi_A \int_\infty \mathcal{D}\omega_A \int \mathcal{D}\phi_B \int_\infty \mathcal{D}\omega_B \exp \left\{ -\chi \rho_0 \int \text{d}r \: \phi_A \phi_B \right\}
\exp \left\{ \int \text{d}r \: \left( \rho_0 \omega_A (\phi_A - \hat{\phi}_A) + \rho_0 \omega_B (\phi_B - \hat{\phi}_B) \right) \right\} \cdot \delta(\phi_A + \phi_B - 1) \tag{2.26}
\]

\[
= \int \mathcal{D}\phi_A \int \mathcal{D}\phi_B \int_\infty \mathcal{D}\omega_A \int_\infty \mathcal{D}\omega_B \int \mathcal{D}\xi \exp \left\{ \int \text{d}r \: \left( \rho_0 (\omega_A \phi_A + \omega_B \phi_B) \right) - \left[ \chi \rho_0 \int \text{d}r \: \phi_A \phi_B \right] + \int \text{d}r \: \xi(r)(\phi_A + \phi_B - 1) \right\}
\cdot \int \mathcal{D}\{\cdot\} \exp \left[ -\int \text{d}r \rho_0 \omega_A \hat{\phi}_A - \int \text{d}r \rho_0 \omega_B \hat{\phi}_B \right], \tag{2.27}
\]

where \(\xi\) is a Langrangian multiplicator field for the incompressibility constraint. We may now reformulate \(Z_C\) in terms of a weight functional \(\mathcal{F}_C\):

\[
Z_C = \int \mathcal{D}\phi_A \int_\infty \mathcal{D}\omega_A \int \mathcal{D}\phi_B \int_\infty \mathcal{D}\omega_B \int \mathcal{D}\xi \exp \left\{ -\mathcal{F}_C[\phi_A, \phi_B, \omega_A, \omega_B, \xi] \right\} \tag{2.28}
\]

with

\[
\mathcal{F}_C = \chi \rho_0 \int \text{d}r \: \phi_A \phi_B - \rho_0 \int \text{d}r \: \left( \omega_A \phi_A + \omega_B \phi_B \right) - \int \text{d}r \: \xi(r)(\phi_A + \phi_B - 1)
- \ln \left[ \int \mathcal{D}\{\cdot\} \exp \left[ -\rho_0 \int \text{d}r \: \omega_A \hat{\phi}_A - \rho_0 \int \text{d}r \: \omega_B \hat{\phi}_B \right] \right]. \tag{2.29}
\]

Remembering definition (2.21), we define \(Q_j\), which is the partition function of a single chain of type \(j\) that moves in fields \(\omega_\alpha\):

\[
Q_j := \int \mathcal{D}_j \{\cdot\} \exp \left[ -\rho_0 \int \text{d}r \: \omega_A \hat{\phi}_{A,j} - \rho_0 \int \text{d}r \: \omega_B \hat{\phi}_{B,j} \right] \tag{2.30}
\]

where

\[
\hat{\phi}_{\alpha,j}(r) := \frac{1}{\rho_0} \int_0^{N_j} ds \: \delta(r - R_j(s)){\gamma}_{\alpha,j}(s) \tag{2.31}
\]

is the partial (relative) density operator for a single chain of type \(j\). Hence,

\[
- \ln \prod_j \frac{1}{n_j!} Q_j^{n_j} = - \sum_j \ln \frac{Q_j^{n_j}}{n_j!} \approx - \sum_j n_j \ln \left( \frac{Q_j}{n_j} \right) + \text{const.}, \tag{2.32}
\]
where Stirling’s approximation:
\[ n_j! \approx \left( \frac{n_j}{e} \right)^{n_j} \sqrt{2\pi n_j} \]  
(2.33)

has been used. Thus,
\[
\mathcal{F}_C = \chi \rho_0 \int dr \phi_A \phi_B - \rho_0 \int dr (\omega_A \phi_A + \omega_B \phi_B) - \rho_0 \int dr \xi(r)(\phi_A + \phi_B - 1)
- \sum_j n_j \ln \left( \frac{Q_j}{n_j} \right) \quad (+ \text{const.})
\]  
(2.34)

Until now, all calculations have been exact (for large \( n_j \)), in the sense that all possible scalar fields \( \rho_\alpha, \omega_\alpha, \) and \( \xi \) contribute to the functional \( \mathcal{F}_C \). In order to evaluate (2.28) further, we must now make a saddle point approximation of the integral. This we do by extremizing \( \mathcal{F}_C \) with respect to its variable functions. We obtain:
\[
\frac{\delta \mathcal{F}_C}{\delta \phi_A} = 0 \quad \Rightarrow \quad \omega_A = \chi \phi_B - \xi,
\]  
(2.35)
\[
\frac{\delta \mathcal{F}_C}{\delta \phi_B} = 0 \quad \Rightarrow \quad \omega_B = \chi \phi_A - \xi,
\]  
(2.36)
\[
\frac{\delta \mathcal{F}_C}{\delta \omega_A} = -\phi_A - \sum_j n_j \frac{\delta \ln Q_j}{\delta \omega_A} = 0 \quad \Rightarrow \quad \phi_A = -\sum_j \frac{n_j}{\rho_0 Q_j} \frac{\delta Q_j}{\delta \omega_A},
\]  
(2.37)
\[
\frac{\delta \mathcal{F}_C}{\delta \omega_B} = -\phi_B - \sum_j n_j \frac{\delta \ln Q_j}{\delta \omega_B} = 0 \quad \Rightarrow \quad \phi_B = -\sum_j \frac{n_j}{\rho_0 Q_j} \frac{\delta Q_j}{\delta \omega_B},
\]  
(2.38)
\[
\frac{\delta \mathcal{F}_C}{\delta \xi} = 0 \quad \Rightarrow \quad \phi_A + \phi_B = 1.
\]  
(2.39)

These are the mean-field equations for the canonical ensemble. \( \xi(r) \) is determined except for a constant. This constant is sometimes chosen such that \( \int dr \xi(r) = 0 \). The fields \( \omega_A \) and \( \omega_B \) were defined as imaginary functions in (2.28), yet Eqs. (2.35) and (2.36) yield real quantities. This is not a contradiction, however, because we can deform the integration path of (2.28) so that it goes through the saddle point, which is on the real axis. Furthermore, we see from the above density equations that
\[
\phi_\alpha = \sum_j n_j \langle \hat{\phi}_{\alpha,j} \rangle_n Q_j = \langle \hat{\phi}_\alpha \rangle_C,
\]  
(2.40)

where \( \langle \ldots \rangle_C \) is the statistical average in a system of \( n_j \) chains of type \( j \). The mean-field approximation thus approximates the local densities with their statistical averages. \textit{A posteriori} we thus justify our interpretation of the \( \hat{\phi}_s \) as densities and of the fields \( \omega \) as local fields. The \( Q_j \) are the partition functions of a single polymer of type \( j \) that propagates in fields \( \omega_A \) and \( \omega_B \), as we switch from space to line integrals:
\[
Q_j = \int D_j \{ \cdot \} \exp \left[ -\sum_\alpha \int dr (\rho_0 \omega_\alpha \hat{\phi}_{\alpha,j}) \right]
\]  
(2.41)
\[
= \int D_j \{ \cdot \} \exp \left[ -\sum_\alpha \int_0^{N_j} ds \omega_\alpha (R_j(s)) \cdot \gamma_{\alpha,j}(s) \right].
\]  
(2.42)
Now, we make a parameter transformation from \( s \in [0, N] \) to \( t \in [0, 1] \). For block copolymers, we also define a parameter \( f \) such that in portions of a chain where \( t \leq f \), we will find A monomers, otherwise B monomers. Moreover, we introduce the relative polymer lengths \( \nu_j \): \[ \nu_A \equiv \frac{N_A}{N}, \quad \nu_B \equiv \frac{N_B}{N}, \quad N \equiv N_{CP} \] (2.43)
The indices A, B, and CP denote A and B homopolymers, and copolymers, respectively. Thus, we obtain for copolymers:

\[ Q_{CP} = \int D_N \{ \cdot \} \exp \left[ -N \int_0^f dt \omega_A(r(t)) - N \int_f^1 dt \omega_B(r(t)) \right] \] (2.44)

and for homopolymers:

\[ Q_A = \int D_{N_A} \{ \cdot \} \exp \left[ -N \int_0^{\nu_A} dt \omega_A(r(t)) \right], \quad Q_B = \int D_{N_B} \{ \cdot \} \exp \left[ -N \int_0^{\nu_B} dt \omega_B(r(t)) \right]. \] (2.45, 2.46)

Hence,

\[ \frac{\delta Q_{CP}}{\delta \omega_A(x)} = -N \int D_N \{ \cdot \} \int_0^f dt \delta(r(t) - x) \cdot \exp \left[ -N \int_0^f dt \omega_A(r(t)) - N \int_f^1 dt \omega_B(r(t)) \right], \] (2.47)

\[ \frac{\delta Q_{CP}}{\delta \omega_B(x)} = -N \int D_N \{ \cdot \} \int_f^1 dt \delta(r(t) - x) \cdot \exp \left[ -N \int_0^f dt \omega_A(r(t)) - N \int_f^1 dt \omega_B(r(t)) \right], \] (2.48)

\[ \frac{\delta Q_A}{\delta \omega_A(x)} = -N \int D_{N_A} \{ \cdot \} \int_0^{\nu_A} dt \delta(r(t) - x) \cdot \exp \left[ -N \int_0^{\nu_A} dt \omega_A(r(t)) \right], \] (2.49)

\[ \frac{\delta Q_B}{\delta \omega_B(x)} = -N \int D_{N_B} \{ \cdot \} \int_0^{\nu_B} dt \delta(r(t) - x) \cdot \exp \left[ -N \int_0^{\nu_B} dt \omega_B(r(t)) \right], \] (2.50)

\[ \frac{\delta Q_A}{\delta \omega_B(x)} = \frac{\delta Q_B}{\delta \omega_A(x)} = 0. \] (2.51)

To understand \( Q_j \) better, consider the propagator \( G_j(x', s'; x, s) \), which gives the probability that if chain coordinate \( s' \) is located at space coordinate \( x' \), we will find chain coordinate \( s \) at \( x \) \((x, x' \in \mathbb{R}^3)\). \( G_j \) is also called the Green’s function of a Gaussian chain.

\[ G_j(x', s'; x, s) = \int D\bar{x}(s) \delta(\bar{x}(s) - x) \delta(\bar{x}(s') - x') \cdot \exp \left( -\int_0^N ds \left[ \frac{3}{2b^2} \left( \frac{d\bar{x}}{ds} \right)^2 + \phi(\bar{x}(s)) \right] \right). \] (2.52)
\( \phi \) is an external field. \( k_B T = 1 \). As before, we rescale \( s \) such that \( s = N_j \Rightarrow \tau = \nu_j \) and \( s = 0 \Rightarrow \tau = 0 \), and switch to curves \( r(\tau) \): \( r(\nu_j) = x(N_j) \), \( r(0) = x(0) \). Furthermore,

\[
ds = N \cdot d\tau, 
\]

(2.53)

\[
\frac{d\mathbf{x}}{ds} = \frac{1}{N} \frac{d\mathbf{r}}{d\tau}. 
\]

(2.54)

Setting \( t' = 0 \) and \( q_j(x, x', t) := G_j(x', 0; x, t) \) we obtain:

\[
q_j(x, x', t) = \int D\mathbf{r}(t) P[r(\tau); 0, t] \cdot \delta(\mathbf{r}(t) - x) \delta(\mathbf{r}(0) - x') \exp \left( -\int_0^t d\tau N\phi \right) 
\]

(2.55)

with \( 0 \leq t \leq \nu_j \),

where

\[
P[r(\tau); 0, t] \equiv \exp \left( -\int_0^t d\tau \frac{3}{2N\sigma^2} \left( \frac{d\mathbf{r}}{d\tau} \right)^2 \right) 
\]

(2.56)

is just the Wiener weight. Identifying

\[
\phi \equiv \sum_\alpha \omega_\alpha(r(\tau))\gamma_{\alpha,j}(\tau) 
\]

(2.57)

(with \( \gamma(s) \Rightarrow \tilde{\gamma}(\tau) \equiv \gamma(\tau) \)), we see that this formalism is exactly the same as for quantum-mechanical particles and their propagators. Due to this analogy, the same solution applies and \( q \) satisfies a diffusion equation similar to the Schrödinger equation:

\[
\dot{q}_j(x, x', t) = \frac{N\sigma^2}{2d} \Delta q_j(x, x', t) - N \sum_\alpha \omega_\alpha(x)\gamma_{\alpha,j}(x) 
\]

(2.58)

with \( 0 \leq t \leq \nu_j \), \( d \): number of dimensions

Integrating over all \( x' \), we get \( q_j(x, t) \), for which Eq. (2.58) holds. Furthermore,

\[
q_j(x, t) = \int D_N \{ \cdot \} \delta(\mathbf{r}(t) - x) \cdot \exp \left( -N \int_0^t d\tau \sum_\alpha \omega_\alpha(\mathbf{r}(\tau))\gamma_{\alpha,j}(\mathbf{r}(\tau)) \right), 
\]

(2.60)

\[
\int D_N \{ \cdot \} \equiv \int D\mathbf{r}(\tau) P[r(\tau); 0, t] 
\]

(2.61)

for a single polymer. For the copolymers, this yields:

\[
q_{CP}(x, t) = \begin{cases} 
\int D_N \{ \cdot \} \delta(\mathbf{r}(t) - x) \cdot \exp \left( -N \int_0^t d\tau \omega_A(\mathbf{r}(\tau)) \right) & , 0 \leq t \leq f \\
\int D_N \{ \cdot \} \delta(\mathbf{r}(t) - x) \cdot \exp \left( -N \int_0^t d\tau \omega_A(\mathbf{r}(\tau)) - N \int_0^t d\tau \omega_B(\mathbf{r}(\tau)) \right) & , f < t \leq 1 
\end{cases} 
\]

(2.62)

Analogously, we define the conjugate propagator for propagation from the other end of the polymer:

\[
q_{CP}^\dagger(x, t) = \begin{cases} 
\int D_N \{ \cdot \} \delta(\mathbf{r}(t) - x) \cdot \exp \left( -N \int_0^t d\tau \omega_A(\mathbf{r}(\tau)) - N \int_t^1 d\tau \omega_B(\mathbf{r}(\tau)) \right) & , 0 \leq t \leq f \\
\int D_N \{ \cdot \} \delta(\mathbf{r}(t) - x) \cdot \exp \left( -N \int_t^1 d\tau \omega_B(\mathbf{r}(\tau)) \right) & , f < t \leq 1 
\end{cases} 
\]
For the homopolymers, the propagators are their own conjugates:

\[ q_A(x, t) = \int D_{N_A} \{ \cdot \} \exp \left( -N \int_0^t d\tau \omega_A(r(\tau)) \right) \cdot \delta(r(t) - x), \quad 0 \leq t \leq \nu \]

\[ q_B(x, t) = \int D_{N_B} \{ \cdot \} \exp \left( -N \int_0^t d\tau \omega_B(r(\tau)) \right) \cdot \delta(r(t) - x), \quad 0 \leq t \leq \nu \]

Comparing these equations with Eqs. (2.44) ff., we find that

\[ Q_j = \int dx \, q_j(x, \nu) \]  

(2.64)

Now, we can calculate the densities (via Eqs. (2.37), (2.38), (2.47) ff.):

\[ \frac{\delta Q_j}{\delta \omega_\alpha(x)} = \int D_N \{ \cdot \} \int_0^{\nu_j} d\tau \left( -N \int_0^{\nu_j} d\tau' \omega_\alpha(r(\tau)) \gamma_{\alpha,j}(x) \right) \cdot \exp \left( -N \int_0^{\nu_j} d\tau' \omega_\alpha(r(\tau)) \gamma_{\alpha,j}(r(\tau)) \right) \]

\[ = -N \int_0^{\nu_j} d\tau \gamma_{\alpha,j}(x) \cdot q_j(x, \tau) q_j^\dagger(x, \tau) \]  

(2.65)

Thus (\( q \equiv q_{CP} \)),

\[ \phi_A(x) = \frac{V_{CP}}{Q_{CP}} \int_0^1 dt \, q(x, t) q_j^\dagger(x, t) + \frac{V_A}{\nu_A Q_A} \int_0^{\nu_A} dt \, q_A(x, t) q_A(x, \nu_A - t), \]  

(2.66)

\[ \phi_B(x) = \frac{V_{CP}}{Q_{CP}} \int_0^1 dt \, q(x, t) q_j^\dagger(x, t) + \frac{V_B}{\nu_B Q_B} \int_0^{\nu_B} dt \, q_B(x, t) q_B(x, \nu_B - t), \]  

(2.67)

with the partial volumes

\[ V_{CP} := n_{CP} \frac{N}{\rho_0}, \quad V_A := n_A \nu_A \frac{N}{\rho_0}, \quad V_B := n_B \nu_B \frac{N}{\rho_0}. \]  

(2.68)

How many free parameters does our theory have? To answer this question, let us take another look at the diffusion equation 2.58, which we will have to evaluate numerically. First, we need to define a length scale. It is convenient to choose the (unperturbed) radius of gyration \( R_g := \sqrt{N b^2/(2d)} \) as the unit length. Hence,

\[ \dot{q}_j(x, t) = \Delta q_j(x, t) - N \sum_\alpha \omega_\alpha(x) \gamma_{\alpha,j}(x). \]  

(2.69)

Moreover, we do not know \( N \) and therefore make another variable substitution:

\[ W_\alpha = N \omega_\alpha. \]  

(2.70)

These \( W \)'s are the quantities that will be calculated, and the diffusion equation becomes:

\[ \dot{q}_j(x, t) = \Delta q_j(x, t) - \sum_\alpha W_\alpha(x) \gamma_{\alpha,j}(x), \quad 0 \leq t \leq \nu_j. \]  

(2.71)
The remaining two mean-field equations transform into:

\[ W_A(x) = \chi N \phi_B(x) - \xi'(x), \quad (2.72) \]
\[ W_B(x) = \chi N \phi_A(x) - \xi'(x). \quad (2.73) \]

The weight functional now reads:

\[ F_C = \frac{\rho_0}{N} \left[ \chi N \int \text{d}r \, \phi_A \phi_B \right. - \int \text{d}r \left( W_A \phi_A + W_B \phi_B \right) - \sum_j \frac{V_j}{\nu_j} \ln Q_j \biggr] \quad (2.74) \]

We have omitted the integral over \( \xi \) as it vanishes anyway once the fields and densities have been determined self-consistently. Note that \( F_C \) is invariant under different choices for the offset constant in \( \xi \). We find that our theory has the following four parameters in the canonical ensemble:

1. The partial volumes, \( V_j \):
   \[ \sum_j V_j = V \]
2. The relative lengths of the polymer species, \( \nu_j \)
3. The interaction parameter, \( \chi N \)
4. The dimensionless chain density, \( C := \frac{\rho_0}{N} R^d \)

The last parameter, the chain density, deserves some attention. Consider

\[ \frac{\rho_0}{N} = \frac{\sum_j n_j \nu_j}{V}. \quad (2.75) \]

If \( \nu_j \) were one for all \( j \), then the chain density would just be the number of polymers per unit volume. If some of the \( \nu_j \) are not equal to one, however, we have to think of \( \frac{\rho_0}{N} \) as the number of polymers of length \( N \) that would fit into a unit volume if it contained only them. \( C \) is also a measure for the length of the polymers. In \( d \) dimensions we get:

\[ V_0 = \frac{V}{\sum_j n_j \nu_j} = Nv_0 \equiv \text{volume per polymer of length one} \quad (2.76) \]

\( v_0 \): volume per monomer

\[ C \propto \frac{\sqrt{N} b^d}{N v_0} = \frac{b^d}{v_0} N^{d-1} \begin{cases} \frac{b^2}{v_0}, & d=2 \\ \frac{b^3}{v_0} \sqrt{N}, & d=3 \end{cases} \quad (2.77) \]

In two dimensions, \( C \) does not depend on \( N \), but we see a square-root dependence in three dimensions. Whereas in two dimensions, different values for \( C \) correspond to different statistical Kuhn lengths and monomer volumes (only), we may imagine these quantities constant with varying \( N \) in three dimensions. In any case, \( C \) defines the ratio of the volume explored by a chain and the volume which it occupies and is in this sense an overlap parameter.
2.2.2 Grand canonical ensemble

In the grand canonical ensemble, we retain overall incompressibility but allow for the percentages of each polymer species to adapt according to the corresponding chemical potential. Most of the conceptual framework presented in the last section still holds, and modifications occur wherever the polymer numbers \( n_j \) appear in the canonical ensemble. The partition function still looks as in (2.28), except that now the path integral is defined as:

\[
\int \mathcal{D} \{ \cdot \} = \prod_j \sum_{n_j=0}^{\infty} \frac{\mu_j^{n_j}}{n_j!} \prod_j n_j \int \mathcal{D} \{ \cdot \}.
\]  

(2.78)

\[
Z_{GC} = \int \mathcal{D} \{ \cdot \} \exp \left( -\chi \rho_0 \int dr \phi_A(r) \phi_B(r) \right) \cdot \delta \left( \dot{\phi}_A(r) + \dot{\phi}_B(r) - 1 \right).
\]  

(2.79)

Note that \( e^{\mu_j} \equiv e^{\beta \mu_j} \). When calculating

\[
\mathcal{F}_{GC} = \chi \rho_0 \int dr \phi_A \phi_B - \int dr \rho_0 (\omega_A \phi_A + \omega_B \phi_B) - \int dr \xi(r) (\dot{\phi}_A + \dot{\phi}_B - 1)
\]

\[- \ln \left( \int \mathcal{D} \{ \cdot \} \exp \left[ -\int dr \rho_0 (\omega_A \phi_A - \int dr \rho_0 \omega_B \phi_B) \right] \right),
\]  

(2.80)

we obtain

\[
- \ln \left( \prod_j \sum_{n_j=0}^{\infty} \frac{\mu_j^{n_j}}{n_j!} \prod_j n_j Q_j \right) = - \sum_j e^{\mu_j} Q_j,
\]  

(2.81)

and from there:

\[
\mathcal{F}_{GC} = \frac{\rho_0}{N} \left[ \chi N \int dr \phi_A \phi_B - \int dr (W_A \phi_A + W_B \phi_B) - \sum_j \frac{N}{\rho_0} e^{\mu_j} Q_j \right].
\]  

(2.82)

The mean field equations now read:

\[
W_A(x) = \chi N \phi_B(x) - \xi'(x),
\]  

(2.83)

\[
W_B(x) = \chi N \phi_A(x) - \xi'(x),
\]  

(2.84)

\[
\phi_A = - \sum_j \frac{N}{\rho_0} \frac{e^{\mu_j} \delta Q_j}{\delta W_A} = \langle \hat{\phi}_A \rangle_{GC},
\]  

(2.85)

\[
\phi_B = - \sum_j \frac{N}{\rho_0} \frac{e^{\mu_j} \delta Q_j}{\delta W_B} = \langle \hat{\phi}_B \rangle_{GC},
\]  

(2.86)

\[
\phi_A(x) + \phi_B(x) = 1,
\]  

(2.87)

which translates into these equations for the densities:

\[
\phi_A(x) = \frac{N}{\rho_0} e^{\mu_C} \int_0^f dt \, q(x,t) q_A(x,t) + \frac{N}{\rho_0} e^{\mu_A} \int_0^\nu_A dt \, q_A(x,t) q_A(x,\nu_A - t),
\]  

(2.88)

\[
\phi_B(x) = \frac{N}{\rho_0} e^{\mu_C} \int_f^1 dt \, q(x,t) q_A(x,t) + \frac{N}{\rho_0} e^{\mu_B} \int_0^\nu_B dt \, q_B(x,t) q_B(x,\nu_B - t).
\]  

(2.89)

The field \( \xi \) is now determined through the offset in the chemical potentials, and the parameters of the model are:
2.3. The phase diagram for ternary AB+A+B melts

1. The chemical potentials, $\mu_j$

2. The relative lengths of the polymer species, $\nu_j$

3. The interaction parameter, $\chi N$

4. The dimensionless chain density, $C := \frac{\rho_0 N}{R_g}$

It is convenient to set $e^{\mu_{CP}} = \frac{\rho_0 N}{V}$ and $e^{\mu_A - \mu_{CP}} = e^{\mu_B - \mu_{CP}} := z$ (if the homopolymers are to be treated symmetrically) so that we get:

$$\phi_A(x) = \int_0^f dt q(x,t)q_A^\dagger(x,t) + z \int_0^{\nu_A} dt q_A(x,t)q_A(x,\nu_A - t), \quad (2.90)$$

$$\phi_B(x) = \int_f^1 dt q(x,t)q_B^\dagger(x,t) + z \int_0^{\nu_B} dt q_B(x,t)q_B(x,\nu_B - t). \quad (2.91)$$

With this choice of $\mu_j$, $F_{GC} / (\frac{\rho_0 N}{V})$ is the free energy per polymer, and

$$F_{GC} = \frac{\rho_0 N}{V} \left[ \chi N \int d\mathbf{r} \left( \phi_A \phi_B - \int d\mathbf{r} \left( W_A \phi_A + W_B \phi_B \right) - Q_{CP} - zQ_A - zQ_B \right) \right] \quad (2.92)$$

Note that in the grand canonical ensemble, our choice for the chemical potentials determines $\xi$ (and hence $\xi' = N\xi$) completely, including the offset constant.

2.3 The phase diagram for ternary AB+A+B melts

I now use SCFT to calculate the phase diagram of symmetric ternary melts of diblock copolymers (AB) plus homopolymers A and B, i.e., in the above notation, $\nu_{CP} = 1$, $\nu_A = \nu_B = \alpha$. The general idea is to find the configuration with lowest free energy per chain for a given set of parameters. In the canonical ensemble, which was used to establish the Leibler and Scott lines, these are $\chi N$ and $\phi_H$. The first-order transition from lamellar to phase-separated region was established by matching free energies per chain as well as chemical potentials. In the lamellar phase, this also includes an optimization with respect to the lamellar periodicity.

On the copolymer-rich side of the diagrams shown in Figs. 2.1 and 2.2, respectively, a line of order-disorder transitions (ODT) separates a disordered region at low interaction strengths, $\chi N$, from a periodically ordered (i.e., lamellar) phase at higher $\chi N$. For pure copolymers ($\phi_H = 0$), this transition is predicted within mean-field theory to occur at $\chi N = 10.495$, and the line of ODTs, often referred to as Leibler line, represents second-order transitions. Based on an analysis by Brazovskii [35], Leibler [31] later clarified this transition to indeed be weakly first-order if fluctuations are accounted for. Fredrickson and Helfand [36] showed that these fluctuations shift the transition to higher incompatibilities, $\chi N$, and specified an analytic expression that depends on the dimensionless polymer density, $C$, introduced above. $C$ acts as a Ginzburg parameter [33], and we recall that in the limit $C \to \infty$, the mean-field solution is recovered.

On the homopolymer-rich side of the diagram, we find a line of second-order transitions from a disordered phase at low $\chi N$ to a region of two coexisting homogeneous
Chapter 2. Self-Consistent Field Theory (SCFT) for Gaussian Chains

18

liquid phases at high \( \chi_N \). For the pure homopolymer system \( \phi_H = 1 \), the upper critical consolute point is found by mean-field Flory-Huggins theory to be at \( \alpha \chi N = 2 \). This line of continuous transitions is referred to as Scott line [34].

The point where the Leibler and Scott lines meet is found, again within mean-field theory, to be an isotropic Lifshitz critical point (LP) [11], which in the case of \( \alpha = 1 \) becomes a Lifshitz tricritical point. According to Broseta and Fredrickson [37], it occurs at a total homopolymer fraction of \( \phi_{LP} = 1/(1 + 2\alpha^2) \) and an incompatibility of \( (\chi N)_{LP} = 2(1 + 2\alpha^2)/\alpha \). The separation into microphases (i.e., lamellae) observed along the Leibler line displays a steady increase in the lamellar periodicity until it finally diverges at the LP, giving rise to macrophase separation along the Scott line. Below the LP in the ordered regime, the system undergoes a first-order transition from the lamellar (L) to the two-phase (A+B) region along the axis of homopolymer concentration, \( \phi_H \).

To calculate the phase diagrams for \( \alpha = 0.2 \) and 0.5 (Figs. 2.1 and 2.2, respectively), I implemented in one dimension the Matsen-Schick Fourier-space approach to SCFT [38], which uses a restricted orthonormal set of harmonic basis functions to approximate the density profiles. These functions are eigenfunctions of the Laplacian, so the solution of the diffusion equation (2.58) reduces to matrix operations. Unless the lamellar periodicity, \( L_{MF} \), exceeds around 10 \( R_g \) (e.g., approaching the LP or for very high \( \chi N \)), I found approximately 30 basis functions to be sufficient for the free energy to be calculated to within 1 part in \( 10^{10} \).

The resulting phase diagrams feature a first-order transition with a three-phase coexistence region (L+A+B) reaching all the way up to the LP for any \( \alpha \). This is in contrast to a previous publication [71], in which this transition was displayed as a continuous un-binding transition (this figure was shown again in Ref. [73]). The correctness of my calculation was confirmed to me, however, by the original author of the faulty diagram [39]. It is further corroborated by recent results of Naughton and Matsen [40], who found a qualitatively similar result for \( \alpha = 1 \). Also, Fig. 4 of Ref. [41] shows a coexistence region for a symmetric ternary blend with \( \alpha = 0.3 \) at \( \chi N = 11 \).

Fig. 2.3 displays the lamellar periodicities, \( L_{MF} \), for \( \chi N = 11.5, 12, \) and 12.5 for \( \alpha = 0.2 \), as obtained from a minimization of the free energy per chain. I have also examined \( L_{MF} \) along the low \( \phi_H \) border of the L+A+B coexistence region. The result is shown in the inset of Fig. 2.3: approaching both the LP and high \( \chi N \), \( L_{MF} \) increases sharply (on theoretical grounds it should actually diverge in these limits); for moderate \( \chi N \), somewhat away from the LP, there is a minimum of \( L_{MF} \) along the binodal. Fig. 2.4 shows typical density profiles at moderate parameters in the lamellar phase.
2.3. The phase diagram for ternary AB+A+B melts

Figure 2.1: Mean-field phase diagram for a ternary AB+A+B blend with $\alpha = 0.2$, as obtained from a Fourier-space implementation of SCFT. $2\phi$ denotes a region of two-phase coexistence between an A-rich and a B-rich phase, $3\phi$ one of three-phase coexistence between an A-rich, a B-rich, and a lamellar phase.

Figure 2.2: Mean-field phase diagram for ternary AB+A+B blend with $\alpha = 0.5$. Notation as in Fig. 2.1.
Figure 2.3: Lamellar periodicities according to the mean-field equations for $\chi N = 12.5$. The inbox shows the end points of these graphs at the boundary of the three-phase (L+A+B) coexistence region. The gap in the $\chi N = 11.5$ curve corresponds to a region in the disordered phase.

Figure 2.4: A and B densities in a lamellar solution of the mean-field equations at $\phi H = 0.5$, $\chi N = 12.5$, $L_{MF} = 3.92$. 
Chapter 3

Entropy-Induced Smectic Phases in Rod-Coil Polymers

Parts of this chapter have been published together with D. E. Sullivan as Ref. [42]. The research reported here was done in large part at the Department of Physics, University of Guelph, Guelph, Ontario, Canada.

In Chapter 2, Self-Consistent Field Theory (SCFT) was introduced for Gaussian chains, and the mean-field equations were derived. Alternatively, in this chapter, I present a theory for semiflexible rod-coil polymers based on the wormlike chain model. Entropic effects stemming from a steric interaction will be shown to suffice for the formation of the smectic phase, which is the liquid-crystalline variety of the lamellar phase. By contrast, for Gaussian chains, a repulsive Flory-Huggins interaction was responsible for microdomain separation.

3.1 Introduction

The characteristic feature of liquid crystals is their hybrid nature between the amorphous liquid and the crystalline solid. They are formed by strongly anisotropic particles, like disks or rods. Fig. 3.1 displays the most common liquid-crystalline phases of rigid rods: (a) In the isotropic phase, neither orientational nor positional order is present. (b) The nematic phase is characterized by a high degree of orientational order – the particles are on average aligned along a vector called the director – but there is no long-range positional order. (c) In the smectic phases, both orientational and positional ordering is observed such that the particles form periodic layers. Depending on the average angle of the rods with the plane dividing the layers, smectic-A (untilted) and smectic-C (tilted) are distinguished. Apart from these main phases, variations may exist for particular rods. Cholesterol, for example, features a ‘cholesteric’ phase in which the nematic director varies according to a helical structure.

Many organic compounds are liquid crystals, e.g., the tobacco mosaic virus. Its length 3000 Å and width ∼ 200 Å are about ten times those of a number of synthetic polypeptides [43]. The latter have found widespread use in industrial applications, notably in color displays for computer and television screens where their capability to change the
polarization of light is exploited. Over the past few decades, considerable research has
been conducted on liquid crystals [43, 44, 45] as well as melts of block copolymers, which
have also been shown to exhibit a multitude of phases of varying complexity [46]. This
chapter will be concerned with liquid-crystalline copolymers, i.e., copolymers which exhibit
phases commonly associated with liquid crystals. One such class of polymers is that of
rod-coil polymers, where a flexible coil is attached to an already mesogenic rod. Doing so,
however, greatly aids the formation of smectic phases, as we shall see in this chapter. The
isotropic, nematic, smectic monolayer, and smectic bilayer phases of rod-coil polymers are
shown schematically in Fig. 3.2.

As we saw in Chapter 2, SCFT [6, 7, 8, 9, 10] is a very powerful tool. However,
one need not restrict oneself to the particular model employed there. In general, imple-
mentations of SCFT differ mainly in two regards: the flexibility of the polymer chain and
the interactions between chain segments. Most commonly, polymers have been modeled as
perfectly flexible Gaussian chains (as in Chapter 2), which assumes that there is no energy
penalty for local bending. Alternatively, the “wormlike” chain model [17, 18] does intro-
duce such a bending penalty and is thus the appropriate model for semiflexible polymers.
Concerning the interactions, most previous investigators have invoked Flory-Huggins-type
repulsive interactions between unlike chain segments in order to induce microphase sepa-
ration and hence the formation of various mesoscopic phases [10, 47, 48].

The aim of the present chapter is to develop a modified SCFT suited to examining
the formation of liquid-crystalline phases in athermal solutions of diblock copolymers each
consisting of a rigid (“rod”) and a flexible (“coil”) part. Both parts are modeled as
wormlike chains, but are characterized by different rigidities. For the interactions between
3.1. Introduction

Figure 3.2: Liquid-crystalline phases of rod-coil block copolymers (schematic): (a) isotropic, (b) nematic, (c) smectic-$A_1$ (monolayer), (d) smectic-$A_2$ (bilayer). Rods are black, coils are red.

any two chain segments, here I do not distinguish between the two parts, i.e., I assume that the same type of interaction applies to any pair of chain segments. I take this interaction to be the limit of the Onsager excluded-volume interaction for thin rods, which favors local alignment of the chains. In the case of homogeneous chains, characterized throughout by the same rigidity, the present model reduces to that applied by Chen et al. [49, 50] to the study of nematic ordering and the isotropic-nematic interface of semiflexible polymers. As is well known [49, 51], the Onsager model is based on a second-virial approximation to the free energy, and hence is strictly valid only for dilute polymer solutions. Here I show that the generalized model with different rigidities for two parts of a polymer is able to account for the formation of lamellar smectic-A phases. With this minimal model, the only possible sources for the formation of such phases are entropic effects. Recently, both smectic monolayer and bilayer phases (traditionally denoted $A_1$ and $A_2$) have been found in an SCFT-based approach which includes Flory-Huggins interactions and treats the rigid sections of the polymers as perfectly rigid and perfectly aligned in the same orientation [52, 53]. In contrast, here I find only one type of smectic-A phase, which would most accurately be classified as a partial bilayer phase (denoted $A_d$). In addition, as in Refs. [49, 50], my model accounts for the disordered isotropic phase.

Liquids of fairly short rod-coil molecules have been examined recently by computer simulations [54, 55, 56, 57, 58, 59]. Generally, these computer studies have shown that the addition of flexible segments to otherwise rigid molecules stabilizes the smectic-A phase with respect to the nematic phase, consistent with experimental results for thermotropic non-polymeric liquid crystals [44, 45]. This behavior is also shown by the present model, although, in contrast to some studies [55, 56, 58], I do not find that the nematic phase is entirely suppressed with respect to the isotropic and smectic phases. My finding that the
only stable smectic-A of the present model is the partial bilayer $A_d$ phase is in agreement with a density-functional treatment by Holyst [60] of rigid “nail-shaped” molecules, and, somewhat more tentatively, with Monte Carlo studies by Mazars et al. [56] of a rod-coil model.

Experimental studies by J. T. Chen et al. [61] demonstrated both monolayer and bilayer smectic ordering in rod-coil diblock copolymers consisting of a highly rigid polyhexylisocyanate block joined to a flexible polystyrene coil. However, these results were complicated by the simultaneous occurrence of tilted (or smectic-C) ordering and crystallization of the rigid blocks. These two effects are not considered here.

### 3.2 Theory

#### 3.2.1 General three-dimensional theory

Consider a monodisperse fluid of $n$ rod-coil diblock copolymers of total contour length $L$, polymerization index $N$, and fixed segment length $a$ such that $L = Na$, occupying a total volume $V$. Any solvent present is considered to be structureless. A fraction $f$ of the total contour length of each copolymer is occupied by relatively rigid (rod) segments, and the remaining fraction by more flexible (coil) segments. The average number density $n/V$ of copolymers here is denoted by $\rho$, but coincides with $C \equiv \rho_0/N$ of Chapter 2. It is related to the monomer number density via $\rho = \rho_0/N$. In accordance with the wormlike chain model for semiflexible chains [17, 18, 47, 48, 49, 50], which is applied here to both sections of the chains, polymers will be treated as space curves $r_i(t)$ characterized by dimensionless unit tangent vectors $u_i(t)$. Hence, microscopic contour-averaged density operators may be defined as

$$\hat{\phi}_{\text{rigid}}(r, u) = \frac{1}{\rho} \sum_{i=1}^{n} \int_0^f dt_i \delta (r - r_i(t_i)) \delta (u - u_i(t_i)),$$

$$\hat{\phi}_{\text{flex}}(r, u) = \frac{1}{\rho} \sum_{i=1}^{n} \int_f^1 dt_i \delta (r - r_i(t_i)) \delta (u - u_i(t_i)),$$  \hspace{1cm} (3.1)

which satisfy the normalization conditions

$$\int dr \, du \, \hat{\phi}_{\text{rigid}}(r, u) = fV,$$

$$\int dr \, du \, \hat{\phi}_{\text{flex}}(r, u) = (1 - f)V.$$  \hspace{1cm} (3.2)

Since these partial densities are only determined $a \ posteriori$ and otherwise do not enter the calculations, I shall instead use the total density,

$$\hat{\phi}(r, u) = \hat{\phi}_{\text{rigid}}(r, u) + \hat{\phi}_{\text{flex}}(r, u).$$  \hspace{1cm} (3.3)

The interaction potential between any two segments, i.e., rod-rod, rod-coil, or coil-coil, is taken to be the Onsager excluded-volume interaction, which in the limit of very thin polymers ($L, a \gg$ diameter $D$) reduces to [50]

$$v(r_1, u_1; r_2, u_2) = L^2 D \delta(r_1 - r_2) \left| u_1 \times u_2 \right|,$$  \hspace{1cm} (3.4)
where the factor $L^2 D$ appears in order to be consistent with the definition of the density operators as contour-averaged polymer densities. This interaction is minimized when contacting segments are parallel, i.e., $|\mathbf{u}_1 \times \mathbf{u}_2| = 0$. Since Eq. (3.4) describes the effects of hard-core repulsion between chain segments, at least at the level of a second-virial approximation [51], here I shall not impose an additional incompressibility constraint to account for such effects, as is commonly done in the theory of dense copolymer melts [9, 10].

The partition function in the canonical ensemble has the form

$$Z = \int \mathcal{D}_n \{\cdot\} \exp \left( -\rho G \int \mathrm{d}\mathbf{r} \int \mathrm{d}\mathbf{u} \int \mathrm{d}\mathbf{u}' \, \hat{\phi}(\mathbf{r}, \mathbf{u}) \, \hat{\phi}(\mathbf{r}, \mathbf{u}') \, |\mathbf{u} \times \mathbf{u}'| \right), \quad (3.5)$$

where

$$G = L^2 D \rho \quad (3.6)$$

is proportional to the average polymer number density $\rho$. Note that the structure of Eq. (3.5) is analogous to that of Eq. (2.18) for Gaussian chains. Moreover, the factor $\rho G \equiv \rho_0 \chi$ here corresponds to the factor $\rho_0 \chi N$ in the theory of Chapter 2. Unlike there, however, the polymer density $\rho$ appears both as a prefactor of the free energy and as a factor in the interaction parameter, $G$. Therefore, $\rho \equiv C \equiv \rho_0 / N$ here is not an independent global prefactor of the free energy functional and hence does not act as a Ginzburg parameter. In Eq. (3.5),

$$\int \mathcal{D}_n \{\cdot\} = \frac{1}{n!} \prod_{i=1}^{n} \int \mathcal{D} \{r_i, u_i\} \mathcal{P} \{r_i, u_i[0, 1]\}, \quad (3.7)$$

where

$$\mathcal{P} \{r_i, u_i[s_1, s_2]\} \propto \prod_{t=s_1}^{s_2} \delta \left[ (u_i(t))^2 - 1 \right] \delta \left[ \frac{r_i(t) - r_i(s_1)}{L} \right] \exp \left[ -\frac{1}{2N} \int_{s_1}^{t} \mathrm{d}s \, \frac{\left[ \frac{\mathrm{d}u_i}{\mathrm{d}t} \right]^2}{\kappa(t)} \right] \quad (3.8)$$

is the statistical weight of a given path and $\kappa(t)$ is a dimensionless bending modulus. According to the model considered here,

$$\kappa(t) = \kappa_{\text{rigid}}, \quad 0 \leq t < f$$
$$\kappa(t) = \kappa_{\text{flex}}, \quad f < t \leq 1. \quad (3.9)$$

As in Chapter 2 with the Gaussian chain model, we now multiply the partition function by $1 = \int \mathcal{D} \{\phi(\mathbf{r}, \mathbf{u})\} \delta(\phi(\mathbf{r}, \mathbf{u}) - \hat{\phi}(\mathbf{r}, \mathbf{u}))$, which allows us to replace the operator $\hat{\phi}$ with the function $\phi$. Then, using the exponential representation of the delta function, the partition function can be rewritten as $Z \propto \int \mathcal{D} \{W\} \int \mathcal{D} \{\phi\} \exp(-\mathcal{F}[W, \phi])$ with

$$\mathcal{F} = \rho \left[ G \int \mathrm{d}\mathbf{r} \int \mathrm{d}\mathbf{u} \int \mathrm{d}\mathbf{u}' \, \phi(\mathbf{r}, \mathbf{u}) \phi(\mathbf{r}, \mathbf{u}') |\mathbf{u} \times \mathbf{u}'| \right.$$  
$$\left. - \int \mathrm{d}\mathbf{r} \int \mathrm{d}\mathbf{u} \, W(\mathbf{r}, \mathbf{u}) \phi(\mathbf{r}, \mathbf{u}) \right] - \ln \left( \frac{Q^n}{n!} \right), \quad (3.10)$$
where
\[ Q = \int \mathcal{D}_1 \{ \cdot \} \exp \left( - \int_0^1 dt \ W(\mathbf{r}(t), \mathbf{u}(t)) \right) \]  (3.11)
is the single-chain partition function. The function \( W(\mathbf{r}, \mathbf{u}) \) is identified with the potential energy, or field, generated by all polymers in the system as seen by a “test” polymer.

Taking the saddle point of the free energy (3.10) with respect to \( \phi(\mathbf{r}, \mathbf{u}) \) and \( W(\mathbf{r}, \mathbf{u}) \), we obtain the mean-field equations:
\[ W(\mathbf{r}, \mathbf{u}) = 2G \int d\mathbf{u}' \phi(\mathbf{r}, \mathbf{u}')|\mathbf{u} \times \mathbf{u}'|, \]  (3.12a)
\[ \phi(\mathbf{r}, \mathbf{u}) = -\frac{V}{Q} \frac{\delta Q}{\delta W(\mathbf{r}, \mathbf{u})}. \]  (3.12b)

In this approximation, the function \( \phi(\mathbf{r}, \mathbf{u}) \) equals the statistical average \( \langle \hat{\phi}(\mathbf{r}, \mathbf{u}) \rangle \) of the microscopic density. In order to solve these equations for the density, we need to express \( Q[W] \) in terms of the end-segment distribution function defined by
\[ q(\mathbf{r}, \mathbf{u}, t) = \int \mathcal{D}\{\mathbf{r}_i, \mathbf{u}_i\} \mathcal{P}\{\mathbf{r}_i, \mathbf{u}_i; [0, t]\} \delta(\mathbf{r} - \mathbf{r}_i(t))\delta(\mathbf{u} - \mathbf{u}_i(t)) \exp \left[ -\int_0^t ds W(\mathbf{r}_i(s), \mathbf{u}_i(s)) \right]. \]  (3.13)
The function \( q^\dagger(\mathbf{r}, \mathbf{u}, t) \) is defined as the end-segment distribution function starting from the opposite end of the polymer. Hence,
\[ Q = \int d\mathbf{u} \int d\mathbf{r} q(\mathbf{r}, \mathbf{u}, t)q^\dagger(\mathbf{r}, \mathbf{u}, t), \]  (3.14)
where the contour variable \( t \) is arbitrary. With the above definitions of \( Q, q, \) and \( q^\dagger \), Eq. (3.12b) yields for the local density:
\[ \phi(\mathbf{r}, \mathbf{u}) = \frac{V}{Q} \int_0^1 dt q(\mathbf{r}, \mathbf{u}, t)q^\dagger(\mathbf{r}, \mathbf{u}, t). \]  (3.15)
The average rigid and flexible densities, \( \phi_{\text{rigid}}(\mathbf{r}, \mathbf{u}) \) and \( \phi_{\text{flex}}(\mathbf{r}, \mathbf{u}) \), are given by expressions analogous to (3.15) on replacing the limits of integration over \( t \) as in Eq. (3.1). The end-segment distribution functions, or propagators, satisfy diffusion-like equations:
\[ \frac{\partial}{\partial t} q(\mathbf{r}, \mathbf{u}, t) = \left[ -L \mathbf{u} \cdot \nabla \mathbf{r} + \frac{1}{2\xi(t)} \nabla^2 \mathbf{u} - W(\mathbf{r}, \mathbf{u}) \right] q(\mathbf{r}, \mathbf{u}, t), \]
\[ \frac{\partial}{\partial t} q^\dagger(\mathbf{r}, \mathbf{u}, t) = \left[ -L \mathbf{u} \cdot \nabla \mathbf{r} - \frac{1}{2\xi(t)} \nabla^2 \mathbf{u} + W(\mathbf{r}, \mathbf{u}) \right] q^\dagger(\mathbf{r}, \mathbf{u}, t), \]  (3.16)
with initial conditions \( q(\mathbf{r}, \mathbf{u}, 0) = 1 \) and \( q^\dagger(\mathbf{r}, \mathbf{u}, 1) = 1 \). Here, I have defined the rigidity parameter \( \xi(t) \equiv \kappa(t)/N \) depending on the chain contour variable \( t \): \( \xi(t) \) equals the persistence length of the corresponding chain section in units of the total contour length \( L \) [47]. This is where the difference between the rigid and flexible parts of the copolymer is accounted for.
To proceed with further analysis of the mean-field equations, we represent the orientational \((u)\) dependencies of the functions \(\phi, W, q\) and \(q^\dagger\) using spherical-harmonic series:

\[
\phi(r, u) = \sum_{l,m} \phi_{lm}(r) Y_{l,m}(u),
\]

\[
W(r, u) = \sum_{l,m} W_{lm}(r) Y_{l,m}(u),
\]

\[
q(r, u, t) = \sum_{l,m} q_{lm}(r, t) Y_{l,m}(u),
\]

\[
q^\dagger(r, u, t) = \sum_{l,m} q^\dagger_{lm}(r, t) Y_{l,m}(u).
\] (3.17)

Since these are all real functions, the expansion coefficients must obey the following conditions:

\[
\phi_{lm}(r) = \phi^*_{l,-m}(r)(-1)^m
\] (3.18)

etc. Next, we expand the kernel \(|u \times u'|\) by use of the addition theorem for spherical harmonics [50],

\[
|u \times u'| = \sum_{l,m} \frac{4\pi}{2l+1} d_l Y_{l,m}(u) Y^*_{l,m}(u')
\] (3.19)

with

\[
d_l = 0, \ l \ odd,
\]

\[
d_0 = \frac{\pi}{4},
\]

\[
d_{2k} = -\frac{\pi(4k+1)(2k)!(2k-2)!}{2^{4k+1}(k-1)!k!(k+1)!}, k = 1, 2, 3, ....
\] (3.20)

Inserting these formulae into the free energy (3.10), the latter can be expressed as

\[
\mathcal{F} = \rho \sum_{l,m} \int dr \left[ \frac{4\pi G}{2l+1} d_l |\phi_{lm}(r)|^2 - \text{Re} \left( W_{lm}(r) \phi^*_{lm}(r) \right) \right] - \ln Q^n, \quad (3.21)
\]

where “Re” denotes real part. The mean-field equations (3.12a) and (3.15) now read

\[
W_{lm}(r) = \frac{8\pi}{2l+1} d_l G \phi_{lm}(r),
\] (3.22)

\[
\phi_{lm}(r) = \frac{V}{Q} \int_0^1 dt \sum_{\ell', m'} \sum_{\ell'' m''} q^\dagger_{\ell', m'}(r, t) q_{\ell'', m''}(r, t) \int du Y_{\ell', m'}(u) Y^*_{\ell'', m''}(u) Y^+_l(u)
\]

\[
= \frac{V}{Q} \int_0^1 dt \sum_{\ell', m'} \sum_{\ell'' m''} q^\dagger_{\ell', m'}(r, t) q_{\ell'', m''}(r, t) \sqrt{\frac{(2\ell' + 1)(2\ell'' + 1)}{4\pi(2l+1)}} C^{\ell''}_{0,0,0} C^{\ell', m'}_{m', m, l} \quad (3.23)
\]

with

\[
Q = \sum_{l,m} \int dr \ q_{lm}(r, t) q^\dagger_{l,-m}(r, t)(-1)^m.
\] (3.24)
The $C_{m', m''}^{l', l}$ are Clebsch-Gordan coefficients, and I have used a result for the integral of three spherical harmonics [62]. The corresponding projections $\phi_{\text{rigid}, l, m}$ and $\phi_{\text{flex}, l, m}$ are obtained likewise by changing the limits of the line integral in Eq. (3.23). In terms of the projections $q_{l, m}(r, t)$, the diffusion-like equation (3.16) yields the following coupled set of equations:

$$\frac{\partial}{\partial t} q_{l, m}(r, t) = - L \sum_{l', m'} \sqrt{\frac{2l' + 1}{2l + 1}} \left[ C_{0, 0, 0}^{l, l'} \left( - \frac{1}{\sqrt{2}} \left( C_{1, m', m}^{l, l'} + C_{-1, m', m}^{l, l'} \right) \frac{\partial}{\partial x} + C_{l', 0, m'}^{l, l'} \frac{\partial}{\partial y} \right) q_{l', m'}(r, t) \right.$$  

$$- \frac{1}{2\xi(t)} (l + 1) q_{l, m}(r, t)$$

$$- \sum_{l''} \sqrt{\frac{(2l'' + 1)(2l' + 1)}{4\pi(2l + 1)}} W_{l''}^{l'} q_{l'', m''}(r, t) C_{m'', m' \cdot m''}^{l'', l'}$$

(3.25)

with initial conditions

$$q_{0, 0}(r, 0) = \sqrt{4\pi},$$
$$q_{l, m}(r, 0) = 0,$$  otherwise.

(3.26)

### 3.2.2 One-dimensional theory

For simple applications of the general theory presented above, we now specialize to situations where the densities vary in only one spatial dimension, which for convenience is chosen to be the $z$ direction. Furthermore, we will restrict analysis to phases that exhibit no azimuthal orientation dependence about the $z$-axis, thereby excluding the possibility of smectic-C phases [53, 57]. Then the only nonzero projections of any angular functions are those with $m = 0$, so that we will subsequently drop the $m$ indices, denoting $\phi_l = \phi_{l, m=0}$, etc. Another consequence is that now all spherical-harmonic expansion coefficients are real. The free energy Eq. (3.21) becomes

$$\mathcal{F} = \rho A \sum_l \int dz \left[ \frac{4\pi G}{2l + 1} d_l \phi_l^2(z) - W_l(z) \phi_l(z) \right] - \ln \frac{Q^n}{n!},$$

(3.27)

where $A$ is the cross-sectional area of the system in the $x$ and $y$ directions. The mean-field equations (3.22) and (3.23) are:

$$W_l(z) = \frac{8\pi}{2l + 1} d_l \phi_l(z),$$

(3.28a)

$$\phi_l(z) = \frac{V}{Q} \sum_{l''} \int_0^1 dt q_{l''}^\dagger(z, t) q_{l''}(z, t) \sqrt{\frac{(2l'' + 1)(2l' + 1)}{4\pi(2l + 1)}} \left( C_{0, 0, 0}^{l'', l'} \right)^2$$

(3.28b)

with

$$Q = A \sum_l \int dz q_{l}^\dagger(z, t) q_l(z, t).$$

(3.29)
Finally, the diffusion-like equation (3.25) becomes

$$\frac{\partial}{\partial t} q_l(z,t) = - L \sum_{l'} \sqrt{\frac{2l' + 1}{2l + 1}} \left( C_{0,0,0}^{1,l',l} \right)^2 \frac{\partial}{\partial z} q_{l'}(z,t) - \frac{1}{2\xi(t)} l(l + 1) q_l(z,t)$$

$$- \sum_{l'',l'''} \sqrt{\frac{(2l'' + 1)(2l''' + 1)}{4\pi(2l + 1)}} \left( C_{0,0,0}^{l'',l',l} \right)^2 W_{l'}(z) q_{l''}(z,t)$$

(3.30)

with initial conditions

$$q_0(z,0) = \sqrt{4\pi},$$

$$q_l(z,0) = 0, \quad l > 0.$$  

(3.31)

In the following, I shall truncate the interaction expansion (3.19) after \( l = 2 \), (equivalent to a “Maier-Saupe” interaction), retaining only the terms with coefficients \( d_0 = \frac{\pi}{4} \) and \( d_2 = -\frac{5\pi}{32} \).

I shall now briefly discuss the computational methods used in solving the theory. The fields and densities are determined self-consistently according to Eqs. (3.28a), (3.28b), and (3.30) using a fixed-point iteration algorithm with variable mixing parameters for successive iterations. An iteration consists of: (a) given the functions \( W_l(z) \), solving Eq. (3.30) and its counterpart for \( q^1_l \); (b) calculating the set of functions \( \phi_l(z) \) from Eq. (3.28b); (c) obtaining a new set of functions \( W'_l(z) \) from Eq. (3.28a); (d) mixing \( W'_l(z) \) and \( W_l(z) \) according to a fixed-point algorithm, which yields a new \( W_l(z) \) for the next iteration.

Concerning step (a), solutions of the diffusion-like equation (3.30) were discretized in time and space according to a Forward Time Centered Space (FTCS) scheme [63], which is explicit and straightforward to implement. As explained in Appendix A, this is a reliable discretization scheme for solving partial differential equations although its accuracy is only first order in the contour discretization \( dt \). For the corresponding diffusion equation in the case of Gaussian chains, a number of more sophisticated schemes such as Crank-Nicholson and DuFort-Frankel [64] are available to improve on stability and accuracy (the accuracy becomes second order in \( dt \) in both cases), and thus to lower the contour resolution needed for a given desired level of accuracy, a key factor in computing times. Due to the coupled nature of Eq. (3.30), however, it seems daunting to apply similarly efficient discretization schemes to the wormlike chain model.

Concerning step (d), the fixed-point iteration algorithm used in finding self-consistent solutions of Eqs. (3.28a) and (3.28b) is a tried and tested method. Here viable alternatives do exist, however, first and foremost being Newton-Raphson-type algorithms like Broyden’s method [63] (see Appendix A). In the present chapter, I have restricted myself to using the fixed-point algorithm, since the primary aim was to develop the framework presented above and illustrate it with a few demonstrative results.

For the rod-coil project, calculations were performed on a one-dimensional grid with periodic boundary conditions, a spatial discretization of \( dz = 0.02 \), and a contour discretization of \( dt = 1/1500 \). The spherical-harmonic expansions of the propagators and densities were truncated after \( l = 12 \). With these parameters, \( \mathcal{F}/V \) can be determined to within a numerical inaccuracy of less than 1%, requiring up to 500 iterations, or 3-4 hours on a Pentium II processor. Note that although the interaction kernel (3.19) is truncated
after \( l = 2 \), one cannot truncate the propagators at the same value of \( l \), since the coupling in (4.21) renders these higher-order projections nonzero.

Once Eqs. (3.28a), (3.28b) and (3.30) have been solved self-consistently, the free energy can be rewritten as

\[
\mathcal{F} = -4\pi G\rho A \sum_l \frac{d_l}{2l+1} \int \phi_l^2(z) - \ln \frac{Q^n}{n!}
\]

where \( I \) have used Stirling’s approximation for the factorial. In a smectic phase, all projections of the local densities and propagators are taken to be periodic functions of \( z \) with period (or layer spacing) \( d \). All integrations over \( z \) can then be evaluated as

\[
\frac{A}{V} \int \frac{dz}{d} = \frac{1}{d} \int_0^d dz
\]

The location of the second-order nematic-smectic transition (cf. Section 3.3) is determined by the behavior of the smectic order parameter defined as:

\[
O_{sm} = \left[ \frac{1}{d} \int \frac{dz}{d} \left( |\sqrt{4\pi} \phi_{\text{rigid},0} - f|^2 + |\sqrt{4\pi} \phi_{\text{flex},0} - (1-f)|^2 \right) \right]^{1/2}
\]

The parameter \( O_{sm} \) vanishes in the isotropic and nematic phases, while it is nonzero in the smectic phase. For a “perfect” smectic phase in which the profiles \( \sqrt{4\pi} \phi_{\text{rigid},0} \) and \( \sqrt{4\pi} \phi_{\text{flex},0} \) have rectangular shapes of widths \( fd \) and \( (1-f)d \), respectively, \( O_{sm} \) has the value \( \sqrt{2f(1-f)} \). The equilibrium period is that value of \( d \) which minimizes the free energy per volume.

### 3.3 Results

The results presented here apply to a fluid of polymers with \( \xi_{\text{rigid}} = 10 \), and \( \xi_{\text{flex}} = 0.1 \). These values more or less represent the feasible bounds on \( \xi \) for performing numerical calculations. To begin with, I examine a system with \( f = 2/3 \).

Fig. 3.3 shows the rigid and flexible density profiles (i.e., zeroth-order projections \( \phi_{\beta,l=0}(z), \beta \in \{\text{rigid,flex}\} \)) as well as the total density profile \( \phi_0(z) \) in a smectic configuration at \( G = 20 \), just above the nematic-smectic transition. We see a clearly defined region of predominantly rigid chain segments and a less pronounced region where the flexible parts are dominant. The individual density variations are nearly pure sine functions, as expected close to a second-order transition. The total density in this lamellar structure exhibits maxima in the rigid regions and minima in the flexible regions. This indicates
3.3. Results

Figure 3.3: Density profiles of a smectic configuration with \( f = 2/3 \), \( \xi_{\text{rigid}} = 10 \), \( \xi_{\text{flex}} = 0.1 \), \( G = 20 \), and period \( 1.3L \). The solid, dotted, and dashed lines correspond to \( (4\pi)^{1/2} \phi_0 \), \( (4\pi)^{1/2} \phi_{\text{rigid},0} \), and \( (4\pi)^{1/2} \phi_{\text{flex},0} \), respectively.

Figure 3.4: Orientational order parameters \( \bar{P}_{1,\text{rigid}} \) (dotted line) and \( \bar{P}_{2,\text{rigid}} \) (solid line) for the same case as in Fig. 1.
Figure 3.5: Order parameters $\bar{P}_{1,\text{flex}}$ (dotted line) and $\bar{P}_{2,\text{flex}}$ (solid line) for the same case as in Fig. 1.

Figure 3.6: Density profiles as in Fig. 1 but for $f = 2/3$, $G = 30$, and period $1.24L$. 
3.3. Results

Figure 3.7: Orientational order parameters of rigid segments for the same case as in Fig. 4.

Figure 3.8: Orientational order parameters of flexible segments for the same case as in Fig. 4.
that the rigid segments pack more efficiently than flexible ones due to their greater susceptibility to local orientational alignment. The non-constant value of the total density $\phi_0(z)$ also reflects the fact that I have not imposed an incompressibility constraint, as discussed above.

More insight can be gained from the orientational projections of the densities, $\phi_{\beta,l}(z)$. Here, I express these in terms of the $z$-dependent order parameters defined as, for each degree $l$,

$$\bar{P}_{l,\beta}(z) = \frac{1}{\sqrt{2l+1}} \frac{\phi_{\beta,l}(z)}{\phi_{\beta,0}(z)},$$

(3.36)
equivalent to the average of the $l^{th}$ Legendre polynomial $P_l(\cos \theta)$ per segment of species $\beta$, where $\theta$ is the angle between a segment axis and the $z$ axis. The lowest-order functions $\bar{P}_{1,\beta}(z)$ and $\bar{P}_{2,\beta}(z)$ corresponding to the configuration of Fig. 3.3 are shown in Figs. 3.4 and 3.5. The second-order distribution $\bar{P}_{2,\beta}$ characterizes the overall degree of orientational order of species $\beta$, while $\bar{P}_{1,\beta}$ indicates the average spatial direction in which segments $\beta$ are oriented. The behavior of the functions $\bar{P}_{1,\beta}(z)$ shows that both the rigid and flexible regions are divided in the middle into domains of positive and negative orientation along $z$, in the manner of a bilayer. We see that both $\bar{P}_{2,\text{rigid}}$ and $\bar{P}_{2,\text{flex}}$ exhibit maxima (minima) in the regions of high (low) rod density (cf. Fig. 3.3). The minimum in the flexible coil distribution $\bar{P}_{2,\text{flex}}$ is more pronounced, while its maximal region is a slightly modulated
plateau within the domain of high rigid density, suggesting that coil segments are oriented by interactions with neighboring rigid segments. Overall, the coils exhibit a much weaker degree of orientational ordering than the rods.

The configuration shown in Figs. 3.3, 3.4 and 3.5 has an optimal period (i.e., yielding the minimum free energy per volume) of $d = 1.30L$ for the chosen parameters $f = 2/3$ and $G = 20$. Keeping the same $f$ but increasing $G$ to 30 results in a decrease of the optimal period to $d = 1.24$. This is accompanied by stronger segregation of the rod and coil regions and less purely sinusoidal behavior of the densities, as shown in Fig. 3.6. The corresponding orientational order parameters are given in Figs. 3.7 and 3.8, and are qualitatively similar to those for $G = 20$. I found that the optimal periods range from 1.1$L$ for $f = 0.3$ to 1.3$L$ for $f = 0.75$ for values of $G$ close to the $N - A_d$ transition. Smectic phases with values of $f$ outside this interval could not be examined due to numerical difficulties. Attempts to generate solutions corresponding to monolayer smectic structures having $d \leq L$ always converged to a uniform (isotropic or nematic) phase.

Fig. 3.9 shows the phase diagram in the $G - f$ plane for rod-coil polymers with $\xi_{\text{rigid}} = 10$ and $\xi_{\text{flex}} = 0.1$. At low mean densities $G$, the steric interactions do not suffice to generate an ordered phase: the system is isotropic. Upon increasing $G$, one encounters a first-order isotropic-nematic ($I - N$) transition. The two branches delineating the coexistence region in the diagram were determined by constructing tangents to the $F/V$ graphs for the isotropic and nematic solutions with a spline interpolation, which amounts to calculating the values of $G$ at equal chemical potentials and pressures. The lower $f$, the higher is the value of $G$ at which the $I - N$ transition occurs: this transition is driven primarily by the tendency of the rigid-rod segments to align and thus requires higher densities if the
proportion of the rods is lowered. For \( f = 1 \), I can compare my data with that of Chen [49], who uses the full kernel (3.19) and obtains an isotropic density of \( G_{\text{iso}} = 4.18 \) and a nematic density of \( G_{\text{nem}} = 5.33 \). If I calculate this transition retaining the kernel up to \( l = 12 \), I obtain good agreement: \( G_{\text{iso}} = 4.16 \) and \( G_{\text{nem}} = 5.29 \). With the truncated kernel \( (l \leq 2) \) used in the majority of this work, the coexistence region becomes more narrow: \( G_{\text{iso}} = 4.89 \) and \( G_{\text{nem}} = 5.35 \).

A second transition, from the nematic to the smectic \((A_d)\) phase, occurs at higher values of \( G \). This transition is second-order within numerical uncertainties, as is indicated by the behavior of the smectic order parameter \( O_{\text{sm}} \), Eq. (3.35). Fig. 3.10 shows that \( O_{\text{sm}} \) starts to grow from zero at a well-defined critical value of the density \( G \). There is a minimum of the \( N - A_d \) transition density at roughly \( f = 0.55 \). On moving toward both lower and higher values of \( f \), the critical values of \( G \) increase sharply until numerical difficulties prevent us from extending the graph further. Obviously, a well-balanced proportion of rod-like vs. coil-like segments facilitates forming a smectic phase: the former stabilize the rod-dominated portion of the density profile (the nematic microdomain, as it were), and the latter, the coil-dominated portion. Deviations from the optimal \( f \) to higher values of \( f \) decrease the entropic advantage of the coils, which has to be compensated for by an increased density. Lower values of \( f \), on the other hand, destabilize the nematic microdomain ordering of the rods. This is why, for even lower values of \( f \), the \( I - N \) and \( N - A_d \) phase boundaries approach each other until intersecting at \( f = 0.32 \). Below this value of \( f \), the \( I - N \) transition is preempted by a first-order \( I - A_d \) transition. For \( f = 0.3 \), we find a coexistence region between \( G_{\text{iso}} = 29.3 \) and \( G_{\text{smec}} = 31.4 \). Unfortunately, numerical problems prevented us from examining even lower values of \( f \): it thus remains an open question by how much the \( I - A_d \) transition deviates from the preempted \( I - N \) transition.

### 3.4 Summary and conclusions

I have presented a self-consistent field theory for semiflexible copolymers in a general three-dimensional as well as a one-dimensional form without azimuthal orientational dependence. Using this framework, numerical calculations have been performed to investigate the occurrence of a smectic-A phase in systems of rod-coil copolymers where the two parts of each molecule differ only in their rigidities.

A phase diagram was established for \( \xi_{\text{rigid}} = 10 \) and \( \xi_{\text{flex}} = 0.1 \), which correspond to very stiff rods and very flexible coils. At low reduced polymer density \( G \), an isotropic phase is present, which becomes nematic for larger values of \( G \) via a first-order transition. At even larger values of \( G \), a partial bilayer smectic-A phase forms, whose period increases with increasing rod fraction \( f \). The latter effect, while small in the examined parameter range, indicates that the coils can overlap with other coil segments as well as with the rods, resulting in interdigitation of the layers. This feature is also suggested by the rather diffuse, sinusoidal variation of the local density profiles \( \phi_{\beta,0} \). The shape of the smectic phase boundary in Fig. 3.9, exhibiting a minimum mean density \( G \) near a rod fraction \( f = 0.55 \), is plausible and consistent with experimental deductions [57]. (See also [66].) Preliminary calculations indicate that lowering the rod rigidity \( \xi_{\text{rigid}} \) shifts all transitions to higher densities \( G \).
In the present work, only anisotropic excluded-volume interactions between chain segments, assumed to be the same for the rigid and flexible portions of each molecule, are taken into account. To generate smectic phases, I have shown that it is sufficient to distinguish rigid and flexible parts of a polymer by means of their rigidities or reduced persistence lengths $\xi$, and thus have demonstrated that lamellar ordering is a purely entropic phenomenon. This is consistent with the findings of computer simulations [54, 56, 57, 58, 59] and a density-functional treatment [60] of diblock models employing only anisotropic repulsive intermolecular forces. In contrast, the smectic ordering found in the SCFT treatments of Refs. [52, 53] is driven primarily by isotropic Flory-Huggins interactions between unlike chain segments, which may well be present in more realistic models. Unlike here, the latter works predicted the formation of both monolayer and bilayer smectic phases as well as strong segregation between rigid and flexible domains, which can be attributed both to the quite different nature of the interactions adopted and to the allowance of local compressibility in the present theory: as conjectured in Ref. [53], compressibility effects “could dramatically stabilize the bilayer phase.”

Due to my use of the Onsager second-virial approximation for treating repulsive interactions between polymers, the present theory is valid only for dilute solutions in the limits $\rho \to 0$ and $L/D \to \infty$ such that $G$ remains finite and nonzero [51]. In previous studies employing the Onsager approximation [49, 50, 51], these limits have been applicable only to the isotropic-nematic phase transition. The phase diagram of Fig. 3.9 shows that smectic ordering induced by differing rigidities can occur under the same limiting conditions. Fig. 3.9 also indicates that the values of $G$ at the nematic-smectic transition increase sharply with increasing $f$, that is, approaching the limit of homogeneous and fairly rigid chains. This is consistent with studies showing the existence of smectic phases in such fluids at nonzero volume fraction $\rho V_{\text{mol}}$, where $V_{\text{mol}} \propto L D^2$ is the molecular volume [67, 68, 69, 70]. In this limit, the Onsager theory should still be qualitatively valid, but the spatially local form adopted in Eq. (3.4) should be generalized to a non-local interaction in order to generate smectic phases in fluids of homogeneous wormlike chains [69].
Chapter 4

Fluctuations in the Ternary AB+A+B System

Parts of this chapter are contained in an article by D. Duchs, V. Ganesan, G. H. Fredrickson, and F. Schmid which at the time of writing is in preparation.

4.1 Introduction

The self-consistent field theory discussed in Chapter 2 is a mean-field approximation which generally produces very good results in regions of a phase diagram that are somewhat removed from phase boundaries. In the vicinity of such boundaries, however, fluctuation effects cannot be neglected. Fredrickson and Helfand [36] showed that in the particular case of incompressible copolymer melts, the order-disorder transition is shifted to higher segregations, $\chi_N$, while the order of the transition changes from second-order to weakly first-order. In the phase diagrams for ternary blends (Figs. 2.1 and 2.2), this corresponds to the ODT at homopolymer volume fraction $\phi_H = 0$. In this chapter, I present a Monte Carlo approach to including thermal fluctuations in an SCFT-based framework and apply it to a ternary AB+A+B blend at homopolymer/copolymer length ratio $\alpha = 0.2$.

This study is largely motivated by a recent series of experiments by Bates et al. [71, 72, 73, 74] on various realizations of the ternary AB+A+B system with $\alpha \approx 0.2$. A combination of neutron scattering, dynamical mechanical spectroscopy, and transmission electron microscopy (TEM) was used to examine the region where mean-field theory predicts a Lifshitz point (LP) (cf. Fig. 2.1). In these experiments, the LP was destroyed by fluctuations, giving rise to a bicontinuous microemulsion phase in its vicinity. At lower $\chi N$, the L and A+B phases were conjectured to be separated by a narrow channel of this phase stretching down from the Lifshitz region, while the Leibler and Scott lines were shifted from their mean-field locations to lower temperatures (i.e., higher $\chi N \propto 1/T$). It seems that as the lamellar periodicity increases along the $\phi_H$ axis, the persistence length of composition fluctuations along the microdomain boundaries decreases until at some point the two become comparable in size and the lamellae begin to rupture. The resulting experimental phase diagrams deviate most clearly from the mean-field diagram for the PEE/PDMS/PEE-PDMS system (Fig. 4.1), in which the molecular weights were lower than in the other systems. This situation corresponds to a low value of the Ginzburg
parameter, C.

Apparently, thermal fluctuations not accounted for within mean-field theory give rise to these changes. To understand and quantify the shift in both the Leibler and Scott lines and to examine the formation of the microemulsion is our primary goal in this chapter. Bicontinuous structures such as the one suggested by the above experiments are particularly interesting from an application point of view in that they are able to impart many useful properties on polymer alloys. For example, if one blend component is conducting or gas-permeable, these properties will be passed on to the entire alloy. Also, mechanical properties such as the strain at break and the toughness index have been observed to exhibit maxima with co-continuous structures [5].

The remainder of this chapter is organized as follows. After a brief discussion of other authors’ contributions to the treatment of fluctuations, we outline our simulation method. Two fluctuating fields appear in the theory: $W_+$ (corresponding to the total density fluctuations) and $W_-$ (corresponding to the composition fluctuations). It is argued heuristically that the fluctuations in $W_+$ are inconsequential for a system subjected to overall incompressibility.

![Experimental phase diagram of the PDMS-PEE/PDMS/PEE blend with $\alpha \approx 0.2$ [73] - Reproduced by permission of The Royal Society of Chemistry.](image)
4.2 Related research

Several authors have attempted extensions of the mean-field approach presented in Chapter 2 of this thesis. Some have done so to implement dynamics in the theory, thereby introducing non-equilibrium treatments, while the aim of others was to examine fluctuations around the saddle point in equilibrium.

Shi, Noolandi, and Desai [75] introduced a general theoretical framework to study anisotropic composition fluctuations about an ordered block copolymer phase with broken symmetry. In their approach, the free energy functional in the partition function is self-consistently expanded around the saddle point using an asymptotic steepest-descent method. The expansion is then cut off after the (second-order) Gaussian term, limiting its applicability to the regime of very high dimensionless polymer densities, , where the amplitudes of the fluctuations are small. Furthermore, obtaining the Gaussian correction term is rather expensive from a computational standpoint because it requires the previous calculation of the relevant saddle points and because it involves anisotropic correlation functions for the system.

J. G. E. M. Fraaije et al. introduced a generalized time-dependent Ginzburg-Landau theory for conserved order parameters as a dynamic variant of mean-field density functional theory (DSCFT) and showed its utility by examining the mesoscopic dynamics of quenched block copolymer melts [76, 77]. Numerically, it involves the integration of (real) Langevin equations for the densities containing a noise term. While this work and extensions thereof [78, 79, 80, 81, 82] used Onsager kinetic coupling coefficients for the density currents, Maurits and Fraaije [83] applied the more general nonlocal kinetic coefficients developed by Kawasaki and Sekimoto [84, 85, 86]. In order to evaluate this model, they reformulated DSCFT in terms of Langevin equations for the external potential fields (i.e., and ), while neglecting fluctuations. This method is referred to as external potential dynamics (EPD).

Reister and Müller [87] used both DSCFT and EPD to investigate spinodal decomposition in a binary polymer mixture as well as compared these results to complementary Monte Carlo simulations using the bond fluctuation model [14, 15]. As regards DSCFT, they used both local and nonlocal Onsager kinetic coefficients. For the EPD, they demonstrated that through a noise term in the dynamic real Langevin equations, random fluctuations can be modeled, thus extending the versatility of the approach. Further, they concluded that EPD is much more feasible from a computational standpoint than DSCFT. These authors used EPD to study dynamical effects. However, the same formalism can be used to simulate fluctuations in equilibrium. In this function, it is equivalent to the method presented in this chapter if fluctuations are discarded.

By contrast, Ganesan and Fredrickson [88, 89, 90] used a complex Langevin method to sample the fluctuations of both and numerically. The densities in this approach are fluctuating complex quantities that only by averaging become real. They used this method to examine the shift of the order-disorder transition in a two-dimensional melt of symmetric diblock copolymers, and found good agreement with an analytical expression derived by Fredrickson and Helfand [36] by means of a Brazovskii [35] analysis. While their method can be applied to the ternary AB+A+B system under consideration in this chapter, its applicability is best for low homopolymer volume fractions, . At higher , it becomes less feasible numerically because the larger length scales in this regime in
4.3 The field-theoretic model

In the following, the Gaussian chain model for ternary blends of Chapter 2 is used to describe the polymers. Let us briefly recall its salient features: in the canonical ensemble, we have a mixture of \( n_A \) homopolymers of type A, \( n_B \) homopolymers of type B, and \( n_{AB} \) symmetric block copolymers in a volume \( V \). The polymerization indices are \( N \) for the copolymers and \( N_A = N_B = \alpha N \) for the homopolymers, and \( f = 1/2 \) denotes the portion of A monomers in the copolymer. We restrict our attention to the concentration isopleth, where the homopolymers have equal volume fractions \( \phi_H = \frac{1}{2} \). Both A and B segments shall be incompressible and occupy the volume \( \frac{1}{\rho_0} \). In this notation, we have

\[
\begin{align*}
n_A &= n_B = \frac{\phi_H \rho_0 V}{2\alpha N}; & n_{AB} &= \frac{(1 - \phi_H) \rho_0 V}{N}.
\end{align*}
\] (4.1)

In a Gaussian chain, the segments are assumed to be perfectly flexible and here are assigned a common statistical Kuhn length, \( b \). The polymers are represented as continuous space curves, \( R_i^j(s) \), where \( i \) denotes the polymer species (A, B, or AB), \( j \) the different polymers of a component, and \( s \) is an arc length variable measured along the chain contour such that \( 0 \leq s \leq \nu_j \). Conformations of non-interacting polymers are given a Gaussian statistical weight, \( \exp(-H_0) \), with a harmonic stretching energy (in units of \( k_B T \)) of,

\[
H_0[R] = \frac{3}{2N^{b^2}} \sum_i \sum_{j=1}^{n_i} \int_0^{\nu_i} ds \left\lVert \frac{dR_i^j(s)}{ds} \right\rVert^2.
\] (4.2)

Interactions between A and B segments are modeled as local contact interactions by a pseudopotential with a Flory-Huggins quadratic form in the microscopic volume fractions,

\[
H_I[R] = \chi \rho_0 \int dr \hat{\phi}_A(r) \hat{\phi}_B(r)
\] (4.3)

which with the substitutions \( \hat{m} = \hat{\phi}_A - \hat{\phi}_B \), \( \hat{\phi} = \hat{\phi}_A + \hat{\phi}_B \) transforms into

\[
H_I[R] = \frac{\chi \rho_0}{4} \int dr (\hat{\phi}^2 - \hat{m}^2).
\] (4.4)

\( \chi \) denotes the Flory-Huggins parameter, and the microscopic density operators \( \hat{\phi}_A \) and \( \hat{\phi}_B \) are defined by

\[
\begin{align*}
\hat{\phi}_A(r) &= \frac{N}{\rho_0} \sum_{j=1}^{n_{AB}} \int_0^f ds \delta(r - R_{AB}^j(s)) + \frac{N}{\rho_0} \sum_{j=1}^{n_A} \int_0^\alpha ds \delta(r - R_A^j(s))
\hat{\phi}_B(r) &= \frac{N}{\rho_0} \sum_{j=1}^{n_{AB}} \int_1^f ds \delta(r - R_{AB}^j(s)) + \frac{N}{\rho_0} \sum_{j=1}^{n_B} \int_0^\alpha ds \delta(r - R_B^j(s)).
\end{align*}
\] (4.5)
Furthermore, the incompressibility constraint is implemented by the insertion of a Dirac delta functional, and we obtain the canonical partition function of the blend, $Z_C$:

$$Z_C \propto \int D\mathbf{R} \prod_i \prod_j D\mathbf{R}_i^j(s) \delta(\hat{\phi} - 1) \exp(-H_0 - H_I), \quad (4.6)$$

where the $\int D\mathbf{R}$ denote functional integrations over chain conformations.

A Hubbard-Stratonovich transformation [92] transforms the term

$$\exp\left[ + \chi N \int d\mathbf{r} \omega_-(\mathbf{r})^2 - \omega_-(\mathbf{r})\hat{m}(\mathbf{r}) \right] (4.7)$$

into

$$\int D\omega_- \exp\left[ \int d\mathbf{r} \left( -\frac{1}{\chi N} \omega_-(\mathbf{r})^2 - \omega_-(\mathbf{r})\hat{m}(\mathbf{r}) \right) \right] \int D\omega_- \exp\left[ -\frac{1}{\chi N} \int d\mathbf{r} \omega_-(\mathbf{r})^2 \right], \quad (4.8)$$

whereby we obtain a new auxiliary field, $\omega_-$. Note the plus sign in front of the quadratic term in the argument of the exponential to be transformed. This insures that the auxiliary field is real. The denominator in (4.8) is only a constant factor and may be dropped. We are free to apply any coefficient to $\omega_-$ so for consistency I set

$$\omega_-(\mathbf{r}) := \frac{\rho_0}{N} W_-(\mathbf{r}). \quad (4.9)$$

Now, I insert the identity $1 = \int D\phi \delta(\phi - \hat{\phi})$ into Eq. (4.6) and replace the two delta functionals with their standard exponential representation, thereby introducing the two fields $W_+(\mathbf{r})$ and $\xi(\mathbf{r})$. We thus obtain

$$Z_C \propto \int D\mathbf{R} \int_\infty DW_- \exp\left[ -\frac{\rho_0}{N} \frac{1}{\chi N} \int d\mathbf{r} W_-^2 \right] \cdot \int D\phi \int_\infty DW_+ \exp\left[ \frac{\rho_0}{N} \int d\mathbf{r} W_+(\phi - \hat{\phi}) \right] \cdot \exp\left[ -\frac{\rho_0}{N} \int d\mathbf{r} \hat{m} \right] \cdot \int D\xi \exp\left[ \frac{\rho_0}{N} \int d\mathbf{r} \xi(\phi - 1) \right] \quad (4.10)$$

This can be written in terms of a weight functional, $H_C$:

$$Z_C \propto \int_\infty DW_- \int_\infty DW_+ \int D\xi \exp[-H_C(W_-, W_+, \xi)] \quad (4.11)$$

with

$$H_C = \frac{\rho_0}{N} \left[ \int d\mathbf{r} W_-^2 - \int d\mathbf{r} W_+ \phi + \frac{\chi N}{4} \int d\mathbf{r} \phi^2 - \int d\mathbf{r} \xi(\phi - 1) \right] - \ln \left( \int D\mathbf{R} \exp\left[ -\frac{\rho_0}{N} \int d\mathbf{r} \left( W_- \hat{m} + W_+ \hat{\phi} \right) \right] \right). \quad (4.12)$$

On making the following substitutions:

$$W_+ = \frac{1}{2} (W_A + W_B), \quad (4.13)$$

$$W_- = \frac{1}{2} (W_A - W_B), \quad (4.14)$$
we find that
\[ W_- m + W_+ \phi = W_A \phi_A + W_B \phi_B, \tag{4.15} \]
and can write
\[ H_C = \frac{\rho_0}{N} \left[ \frac{1}{\chi N} \int dr \: W_-^2 - \int dr \: W_+ \phi + \frac{\chi N}{4} \int dr \: \phi^2 - \int dr \: \xi(\phi - 1) - \sum_j \frac{V_j}{\nu_j} \ln Q_j \right] \tag{4.16} \]
By virtue of the incompressibility constraint, \( \phi = 1 \), which yields
\[ Z_C \propto \int_\infty^\infty DW_- \int_\infty^\infty DW_+ \exp[-H_C(W_-,W_+)] \tag{4.17} \]
with
\[ H_C(W_-,W_+) = C \left[ \frac{1}{\chi N} \int dr \: W_-^2 - \int dr \: W_+ - V(1 - \phi_H) \ln Q_{AB} - \frac{V \phi_H}{2\alpha} \ln Q_A - \frac{V \phi_H}{2\alpha} \ln Q_B \right], \tag{4.18} \]
\[ C = \frac{\rho_0}{N} R_{g0}^d. \tag{4.19} \]
As in previous chapters of this thesis, all lengths are expressed in units of the unperturbed radius of gyration, \( R_{g0}^d = b(N/(2d))^{1/2} \), where \( d \) is the space dimension. The parameter \( C = \frac{\rho_0}{N} R_{g0}^d \), in being a global prefactor to \( H_C \), acts as a Ginzburg parameter [33] such that in the limit \( C \to \infty \) the partition function (4.17) is reduced to its saddle point and the mean-field solution becomes exact. In (4.18), \( i \equiv \sqrt{-1} \), and \( Q_A, Q_B, \) and \( Q_{AB} \) denote the single chain partition functions for A, B, and AB chains, respectively, in the potential fields \( W_-(r) \) and \( W_+(r) \). Note that \( W_- \) is conjugate to the difference in A and B densities, \( \hat{m} \), and \( W_+ \) to the total density, \( \hat{\phi} \). Moreover, \( W_- \) is real, whereas \( W_+ \) is imaginary. This renders \( H_C \) complex. The single chain partition functions can be expressed in terms of the Feynman-Kac formulae [6] as:
\[ Q_i = \int dr \: q_i(r, \nu_i), \tag{4.20} \]
where the propagators \( q_i \) satisfy the diffusion equation
\[ \frac{\partial}{\partial s} q_i(r,s) = \Delta q_i(r,s) - q_i U_i; \quad q_i(r,0) = 1 \tag{4.21} \]
\[ U_A = W_+ + W_-, \quad 0 \leq s \leq \alpha \]
\[ U_B = W_+ - W_-, \quad 0 \leq s \leq \alpha \]
\[ U_{AB} = \begin{cases} \quad U_A, \quad 0 \leq s < f \\ \quad U_B, \quad f \leq s \leq 1 \end{cases} \tag{4.22} \]
An analogous diffusion equation applies to the conjugate propagators, \( q_i^\dagger \), which propagate from the opposite end of a polymer. Due to their symmetry, the propagators of the
homopolymers are their own conjugates. Hence, we can calculate density functions, \( \bar{\phi}_A \) and \( \bar{\phi}_B \), from

\[
\bar{\phi}_A(r) := V(1-\phi_H) \frac{Q_{AB}}{Q_A} \int_0^f ds q_{AB}(r,s)q_{AB}^\dagger(r,1-s) + \frac{V\phi_H}{2\alpha Q_A} \int_0^\alpha ds q_A(r,s)q_A(r,\alpha-s),
\]

(4.23)

\[
\bar{\phi}_B(r) := V(1-\phi_H) \frac{Q_{AB}}{Q_B} \int_f^1 ds q_{AB}(r,s)q_{AB}^\dagger(r,1-s) + \frac{V\phi_H}{2\alpha Q_B} \int_0^\alpha ds q_B(r,s)q_B(r,\alpha-s).
\]

(4.24)

These density functions depend on \( W_- \) and \( W_+ \) and are thus in general complex. However, their ensemble averages yield real densities:

\[
\bar{\phi}_{A,B} = \langle \bar{\phi}_{A,B} \rangle = \langle \hat{\phi}_{A,B} \rangle.
\]

The above derivation was for the canonical ensemble. A similar set of formulae is obtained in the grand-canonical ensemble:

\[
H_{GC}(W_-,W_+) = C \left[ \frac{1}{\chi N} \int dr W_+^2 - \int dr W_+ - Q_{AB} - zQ_A - zQ_B \right],
\]

(4.25)

where \( z = \exp((\Delta \mu/k_BT)) \), and \( \Delta \mu \equiv \mu_A - \mu_{AB} = \mu_B - \mu_{AB} \) is the difference in the chemical potentials of homopolymers and copolymers. For the density functions, we get:

\[
\bar{\phi}_A(r) = \int_0^f ds q_{AB}(r,s)q_{AB}^\dagger(r,1-s) + z \int_0^\alpha ds q_A(r,s)q_A(r,\alpha-s),
\]

(4.26)

\[
\bar{\phi}_B(r) = \int_f^1 ds q_{AB}(r,s)q_{AB}^\dagger(r,1-s) + z \int_0^\alpha ds q_B(r,s)q_B(r,\alpha-s).
\]

(4.27)

In the self-consistent mean-field theory (SCFT) of Chapter 2 [29, 9, 93], the integrand of (4.17) is approximated by saddle points \((W_+^*, W_-^*)\) (cf. (2.35) f.) such that

\[
\frac{\delta H}{\delta W_+} \bigg|_{W_+^*, W_-^*} = 0, \quad \frac{\delta H}{\delta W_-} \bigg|_{W_+^*, W_-^*} = 0.
\]

(4.28)

A homogeneous saddle point then corresponds to the disordered phase, whereas every ordered phase has a unique inhomogeneous saddle point. At finite \( C \) parameters, however, thermal fluctuation effects in the vicinity of phase boundaries disvalidate the mean-field results in these regions, and strategies for evaluating (4.17) must be found. There exist perturbation calculations in the literature [94], but they suffer from the necessary cut-off that has do be done at some finite order. By contrast, the method presented in this chapter, like the complex Langevin method of Ganesan and Fredrickson [88, 89, 90], is non-perturbative in nature, allowing for a more complete account of fluctuation effects. As discussed in Section 4.2, there also exist real Langevin methods in the literature which have examined the \( W_- \) fluctuations [87].

### 4.4 The field-theoretic Monte Carlo method

My approach to sampling (4.17) is based on the Monte Carlo method. We recall that \( W_- \) is real and \( W_+ \) imaginary, and that the fluctuations in \( W_+ \) correspond to those of the total density. Although the incompressibility constraint implemented in our model
effectively only constrains \( \phi \equiv \phi_A + \phi_B = \langle \phi_A + \phi_B \rangle \), we can still surmise the effect of the \( W_+ \) fluctuations that govern them to be rather small compared to the composition fluctuations governed by \( W_- \). Therefore, I shall tackle \( W_- \) and \( W_+ \) in two distinct steps, after which I will be able to argue that the effect of the \( W_- \) fluctuations is indeed the dominant one.

### 4.4.1 \( W_- \) fluctuations

To simulate the fluctuations in \( W_- \), I pick a starting density configuration and use the SCFT mean-field equations (2.35) f. to calculate a seed \( W_- \) and \( W_+ \). Next, the algorithm proceeds according to the following scheme:

(a) Make a global move in \( W_- \), i.e., tentatively add a random number \( \in [-1, +1] \) times a stepwidth parameter to every \( W_-(r) \), and denote by \( W_0^- \) the old \( W_- \).

(b) Calculate a self-consistent \( \bar{W}_+ \) that is a partial saddle point with the tentative new \( W_- \) fixed; to do this, we need to satisfy the condition

\[
\bar{\phi}_A(r; [\bar{W}_+, W_-]) + \bar{\phi}_B(r; [\bar{W}_+, W_-]) = 1. \tag{4.29}
\]

Technically, I iteratively solve the equations

\[
\bar{W}_+(r) = \frac{\chi N}{2} \phi - \xi(r), \tag{4.30}
\]

\[
\phi = 1. \tag{4.31}
\]

Note that \( \xi \), which is a Lagrangian multiplier field for the incompressibility constraint, is fully determined by \( \bar{W}_+ \) once self-consistency is established. The initial guess for \( \bar{W}_+ \) (and thus \( \xi \)) in this iteration, \( \bar{W}_0^+ \), is taken to be the old \( \bar{W}_+ \) from the last Monte Carlo cycle. The iteration, which employs a two-step Anderson mixing scheme (cf. Appendix A) [99], is terminated once \( \Delta H \equiv H[W_-, \bar{W}_+] - H[W_0^-, \bar{W}_0^+] \) is determined to within 0.001, or a few parts in \( 10^4 \). Every step in this iteration requires the solution of the diffusion equation (4.21) on the entire simulation lattice, and this is where practically all the computing time is spent. The number of iterations to find \( \bar{W}_+ \) in most cases is roughly around 10.

(c) Accept or reject the move according to a standard Metropolis criterion, i.e., if the resulting difference in \( H \), \( \Delta H \), is negative, or otherwise with a probability \( \exp(-\Delta H) \).

(d) Go back to (a) unless the maximum number of Monte Carlo steps has been reached.

### 4.4.2 \( W_+ \) fluctuations

As a result from SCFT, we know that the saddle point \( W_+^* \) of \( W_+ \) as well as the partial saddle point \( \bar{W}_+[W_-] \) are purely real, despite the fact that the integration path of \( W_+ \) in (4.17) is along the imaginary axis. Neglecting surface terms and because we do not cross any singularities, we can therefore deform the integration path and represent \( W_+ \) as

\[
W_+(r) = \bar{W}_+[W_-](r) + i\tilde{\omega}_+(r) \tag{4.32}
\]
Furthermore, we can split the argument of the integral in the partition function into a real part and a complex reweighting factor:

\[ Z_C \propto \int dW_- \int d\tilde{\omega}_+ \exp \left( -H^R_C \right) \cdot I_C, \quad (4.33) \]

\[ H^R_C = C \left[ \frac{1}{\chi N} \int dr \, W^2 - \int dr \, \tilde{W}_+ - \sum_i \left( \frac{V_i}{\nu_i} \text{Re} (\ln Q_i [W_-, \tilde{\omega}_+]) \right) \right], \quad (4.34) \]

\[ I_C = \exp \left( C \left[ i \int dr \tilde{\omega}_+ + i \sum_i z_i \text{Im} (\ln Q_i [W_-, \tilde{\omega}_+]) \right] \right), \quad (4.35) \]

And in the grand canonical ensemble:

\[ Z_{GC} \propto \int dW_- \int d\tilde{\omega}_+ \exp \left( -H^R_{GC} \right) \cdot I_{GC}, \quad (4.36) \]

\[ H^R_{GC} = C \left[ \frac{1}{\chi N} \int dW^2 - \int d\tilde{W}_+ - \sum_i z_i \text{Re} (Q_i [W_-, \tilde{\omega}_+]) \right], \quad (4.37) \]

\[ I_{GC} = \exp \left( C \left[ i \int d\tilde{\omega}_+ + i \sum_i z_i \text{Im} (Q_i [W_-, \tilde{\omega}_+]) \right] \right), \quad (4.38) \]

where \( z_{AB} = 1 \) and \( z_A = z_B = z \).

We can thus in principle simulate both \( W_- \) and \( W_+ \) fluctuations, e.g., by alternating a move in \( W_- \) with a number of moves in \( \tilde{\omega}_+ \) and replacing \( \triangle H_{C,GC} \) with \( \triangle H^R_{C,GC} \) in the Metropolis criterion. Each configuration then needs to be weighted with the factor \( I_{C,GC} \). We note that the propagators and the field term in the diffusion equation (4.21) are now complex, but the averages of all physical quantities will approach real values in equilibrium. Evidently, the \( W_+ \) fluctuations exert their influence on the simulation in two ways: (a) through the complex reweighting factor \( I_{C,GC} \) and (b) through the difference of \( \triangle H^R \) from \( \triangle H \). We define \( \triangle H^* = \triangle H^R - \triangle H \) and so can interpret \( \sqrt{\langle (\triangle H^*)^2 \rangle} \) as a measure for the deviation of the simulation from a reference case in which only \( W_- \) fluctuations are simulated.

In the canonical ensemble, a phase factor \( e^{i\psi} \) that conserves \( H_C \) (and hence \( |Q_i| \) and \( H^R \)) is introduced if \( \tilde{\omega}_+ \) is shifted by an offset constant, \( \psi \), which we are free to choose in the canonical ensemble. \( \psi \) will freely move in canonical simulations unless constrained, e.g., by demanding that \( \int dr \, \tilde{\omega}_+ (r) = 0 \). By contrast, in the grand canonical ensemble, \( Q^R \) and hence \( H^R \) are at lowest approximation quadratic functions around \( \tilde{\omega}_+ = 0 \). As a result, the shape of the \( H^R(W_+) \) energy landscape is much steeper in the grand canonical ensemble than in the canonical, and no constraint is needed.

Therefore, I choose the grand canonical ensemble for testing my simulation technique of \( W_+ \) fluctuations. Before starting a run at \( z = 2 \), \( \chi N = 12 \), \( \alpha = 0.2 \), and \( C = 50 \), I fully equilibrated a simulation of \( W_- \) fluctuations only with the same parameters, which started from a disordered configuration and ran a million Monte Carlo steps. This set of parameters is very close to the (fluctuation-corrected) order-disorder transition in the phase diagram, on the disordered side. The average homopolymer fraction, \( \bar{\phi}_H \), is at 0.65. On a \( 32 \times 32 \) lattice with periodic boundary conditions, I used \( dx = 0.78125 \). I then took
Chapter 4. Fluctuations in the Ternary AB+A+B System

Figure 4.2: Re(H) in a simulation of \( W^- \) and \( W^+ \) against number of Monte Carlo steps in \( W^- \) for \( C = 50, \chi N = 12, z = 2 \) (grand canonical ensemble), started from a configuration equilibrated in \( W^- \). Each \( W^- \) step alternates with ten steps in \( \tilde{\omega}^+ \). This equilibrated configuration and switched on the \( W^+ \) fluctuations, again making ten (global) moves in \( W^+ \) after each move in \( W^- \) so as to roughly spend equal amounts of computing time for the two types of moves. Note in Fig. 4.2 how \( H_{GC} \) quickly increases and then plateaus off, indicating that the \( W^+ \) fluctuations have become saturated. For the same run, I recorded the previously introduced \( \Delta H^* \), which is displayed in Fig. 4.3. This simulation was done with a chain contour discretization of \( ds = 0.01, \) which allows for the calculation of \( \Delta H \) to within 0.01. Typically, i.e., in simulations of \( W^- \) only, however, I used \( ds = 0.05 \) in order to save computing time. The discretization error of \( \Delta H \) in this case is increased to \( \lesssim 0.08 \). Considering that typical values of \( \Delta H \) are on the order of \( \pm 3 \), this appears like a very moderate numerical error. At the same time, it becomes clear that the effect of the fluctuations in \( W^+ \) expressed through \( \Delta H^* \) is well below our standard numerical accuracy: \( \sqrt{\langle (\Delta H^*)^2 \rangle} \approx 0.022 \). Furthermore, the average of \( \Delta H^* \) very closely approximates zero, so there is no drift of \( \Delta H \) as a result. I therefore conclude that contributions of the \( W^+ \) fluctuations to \( H^R \) cannot be distinguished as such from numerical inaccuracies.

Next, I examine the reweighting factor, \( I_{GC} \), in the particular simulation discussed above. Numerically, I must evaluate:

\[
I_{GC}^R = \cos \left( C \left[ \int \text{d}r \tilde{\omega}^+ + \sum_i z_i Q_i^l \right] \right) \equiv \cos(A)
\]

The two terms in the argument, \( A \), of the cosine in (4.39) have opposite signs and largely cancel out each other. However, in practice, the numerical application of \( I_{GC} \) to the
4.4. The field-theoretic Monte Carlo method

Figure 4.3: $\Delta H^*$ (difference of $\Delta H$ and a reference value at $\tilde{\omega}_+$ fluctuations switched off) in the same simulation as in Fig. 4.2.

Figure 4.4: The argument, $A$, of the cosine in the reweighting factor in the same simulation as in Fig. 4.2.
Figure 4.5: Autocorrelation function of $W_-$ in the same simulation as in Fig. 4.2.

Figure 4.6: Correlation functions $C[I, I]$ (dashed), $C[\tilde{\omega}_+, \tilde{\omega}_+]$ (dashed-dotted), $C[I, W_-]$ (left inset), and $C[\tilde{\omega}_+, W_-]$ (right inset) in the same simulation as in Fig. 4.2. Note the different scale from Fig. 4.5.
4.4. The field-theoretic Monte Carlo method

Simulation is impeded greatly by the fact that $A$ is still not a small number. $I_{GC}$ is a non-positive-definite weighting factor, echoing the “sign problem” encountered in fermionic lattice gauge theories in elementary particle physics [103]. Fig. 4.4 displays part of a time series of $A$ in our above test run. It performs fast oscillations around a very small absolute mean value which within numerical accuracy closely approximates zero when averaged over the entire run, although it is expected from theory to be finite. This is the essence of ‘my’ sign problem: the statistical deviation from the mean completely dominates $A$.

At this point, I seem to have arrived at a dead end, having to concede that I cannot numerically implement the reweighting factor. However, we still may get a sense of the impact of the $\tilde{\omega}_+^{+}$ fluctuations from examining pairwise correlation functions of $\tilde{\omega}_+^{+}$, $W_-$, and $I_{GC}$. The correlation function is defined as

$$C[A,B;t] = \frac{\langle A(t')B(t'+t) \rangle - \langle A \rangle \langle B \rangle}{\sigma_A \sigma_B^2},$$

where $\sigma$ denotes standard deviation, and $t$ and $t'$ are numbers of moves in $W_-$. If $A$ or $B$ is a field, I average over all lattice points $r$. Fig. 4.5 shows $C[W_-W_-]$, and in Fig. 4.6, $C[I,I]$, $C[\tilde{\omega}_+^{+},\tilde{\omega}_+^{+}]$, $C[I,W_-]$, and $C[\tilde{\omega}_+^{+},W_-]$ are displayed. On comparing the various correlation times, an important qualitative difference is observed: while the autocorrelation time of $W_-$ is on the order of $10^5$ Monte Carlo steps in $W_-$ (subsequently denoted MCS), those of $\tilde{\omega}_+^{+}$ and $I$ are only on the order of a few hundred MCS, i.e., several orders of magnitude smaller. Furthermore, the correlation functions of $W_-$ with both $\tilde{\omega}_+^{+}$ and $I$ are virtually flat lines, indicating the absence of any cross-correlations. The same is obtained for $C[\tilde{\omega}_+^{+},W_-^2]$. With this result in mind, consider again the partition function (4.36).

I demonstrated before that we may essentially replace $H_R[W_-,\tilde{\omega}_+^{+}]$ with $H[W_-]$ and so rewrite $Z_{GC}$ as

$$Z_{GC} \propto \int D W_- \exp \left( -H_{GC} \right) \int D \tilde{\omega}_+ I_{GC}[W_-,\tilde{\omega}_+].$$

Since, as we have seen, $I$ and $W_-$ are virtually uncorrelated, I argue that the results of the simulation of $W_-$ cannot possibly be altered by the reweighting factor in a significant way.

At this point, a remark concerning my numerical methods is due: the single most time-consuming part in the simulations is the solution of the diffusion equation (4.21). Initially, I variably used the Crank-Nicholson (CN) and DuFort-Frankel (DF) finite differencing schemes [63, 64] as explained in Appendix A to effectuate this task. As it turned out, however, a pseudo-spectral method that had been used successfully in the past to solve the mathematically very similar Schrödinger equation [100] and that had recently been applied to SCFT [101] was far better suited for my needs. It is explained in detail in Appendix A. Typically, it allowed for an order of magnitude higher values of $d$ than either CN or DF to achieve the same accuracy. Implementing the scheme with highly optimized Fast Fourier Transform routines like the publicly available FFTW library [102], the computing time per contour step is only roughly double that needed for DF and half that needed for CN on a single processor. I therefore regard it as the method of choice in the context of polymer field theory. For the larger system sizes to be discussed in Section 4.6, a parallel version of the code was used.
4.5 Symmetries of the complex partition function

I consider first $I_{GC}$ and to this end split the complex diffusion equation into two real equations for the real and imaginary parts of the propagator for polymers of type $i$, denoted $q_i^R$ and $q_i^I$ respectively. Note the symmetry properties of this coupled set of equations with respect to a sign change in $\tilde{\omega} +$. For $W_+ = W_+ \pm i\tilde{\omega}_+$, I obtain

$$\frac{\partial}{\partial s} q_i^R = \Delta q_i^R - U_i q_i^R \pm \tilde{\omega}_+ q_i^I,$$
$$\pm \frac{\partial}{\partial s} q_i^I = \pm \Delta q_i^I \mp U_i q_i^I - \tilde{\omega}_+ q_i^R.$$

We see that $q_i^R$ remains unchanged whereas $q_i^I$ goes to $-q_i^I$. Thus, for the corresponding single chain partition function $Q_i = Q_i^R + iQ_i^I$, $Q_i^R$ is symmetric in $\tilde{\omega}_+$, and $Q_i^I$ antisymmetric. Furthermore, the exponent of the reweighting factor $I_{GC}$ is antisymmetric in $\tilde{\omega}_+$ (cf. (4.38)). Since in the partition function (4.36) we integrate over all possible configurations of $\tilde{\omega}_+$, the imaginary parts of $I_{GC}$ cancel out for each antisymmetric pair, and consequently, we effectively only need to apply the real part of $I_{GC}$:

$$I_{GC}^R = \cos \left( C \left[ \int d\tilde{\omega}_+ + \sum_i z_i Q_i^I [W_-, \tilde{\omega}_+] \right] \right).$$

These symmetries reveal that in a functional expansion of $Q_i$ in $\tilde{\omega}_+$ around the partial saddle point, $\tilde{Q}_i[W_-]$, the even-order terms coincide with the expansion of $Q_i^R$, and the odd-order terms with that of $Q_i^I$. Moreover,

$$\frac{dQ_i}{dW_+} = \frac{dQ_i^I}{d\tilde{\omega}_+},$$

and

$$\sum_i z_i \frac{dQ_i^I}{d\tilde{\omega}_+} = \sum_i z_i \frac{dQ_i}{dW_+} = -(\hat{\rho}) = -1$$

by virtue of the incompressibility constraint. Hence,

$$\sum_i z_i Q_i^I = -\int d\tilde{\omega}_+ + \mathcal{O}(\tilde{\omega}_+^3),$$

and finally

$$I_{GC} = \cos \left( \mathcal{O}(\tilde{\omega}_+^3) \right) = 1 - \mathcal{O}(\tilde{\omega}_+^2),$$

$$Q_i^R = \tilde{Q}_i[W_-] + \mathcal{O}(\tilde{\omega}_+^2).$$

The above derivation of Eqs. (4.47) and (4.48) was for the grand canonical ensemble but holds true if one is to derive analogous relations in the canonical ensemble, as well, because the leading term of $\ln Q_i$ is linear in $Q_i$, as in the grand canonical ensemble. Based on these analytical results, we might have expected $I_{GC}$ to deviate only very slightly from unity. Hence, the impact of the $W_+$ fluctuations should have materialized solely as a (real) $\mathcal{O}(\tilde{\omega}_+^2)$ contribution to $H_{GC}$. However, as demonstrated in the last section, this assumption is not confirmed in actual simulations, where the deviations of $I_{GC}$ from unity are not negligible. The reason for this is that the argument of the cosine not only scales with $C$ (a large number) but is also an extensive quantity.
4.6 The order-disorder transition

In evaluating our simulations of the ternary A+B+AB system, we must establish two phase boundaries: first, the shifted order-disorder transition, and second, the onset of the phase-separated region. For the former, two parameters were used which I shall define in the following.

A given distribution $W_-(r)$ is translated via (4.24) into an image $\bar{\phi}_A(r)$. All simulations performed with the field-theoretic Monte Carlo method that are presented in this section took only $W_-$ fluctuations into account. Hence, $\bar{\phi}_A(r)$ is real. Lattice points are denoted black if $\bar{\phi}_A(r) < 0.5$, and white if $\bar{\phi}_A(r) \geq 0.5$. $\langle \max(l_a) \rangle$ is then the maximal length of either black or white sections along a one-dimensional cross section of the image in direction $a$, averaged over all offsets along the $x$ axis (or equivalently the $y$ axis). The direction persistence parameter, $\Lambda$, is defined for a two-dimensional black-and-white image as

$$\Lambda := \left( \frac{\langle \max(l_{||}) \rangle}{\langle \max(l_{\perp}) \rangle} \right)_{\text{all directions}} - 1 ,$$

where $l_{||}$ denotes the direction being averaged over, and $l_{\perp}$ the direction perpendicular to a given $l_{||}$. Please consult Appendix A on how to calculate it. $\Lambda$ is 0 in a disordered configuration and nonzero-positive in a lamellar configuration, and thus constitutes a measure for the lamellarness of a configuration. Furthermore, it is dimensionless and does not directly depend on the lattice size. $\Lambda$ measures lamellarness from a local perspective. Complementarily, the parameters $F_2(q)$ and $F_4(q)$ measure the anisotropy of the Fourier transform, $F(q)$, of an image and thereby look at periodicity from a more macroscopic point of view:

$$F_n(q) := \frac{1}{2\pi} \left[ \int_0^{2\pi} d\phi |F(q)|^2 e^{in\phi} \right] ,$$

$F_2(q)$ and $F_4(q)$, too, are identically 0 in a disordered (isotropic) configuration and > 0 otherwise. A more convenient way to analyze $F_n(q)$ is to calculate a normalized ratio of the integral of $F_n(q)$ and the standard deviation (denoted $\sigma$) of $q$ under the distribution $F_n(q)$, which yields a single dimensionless number:

$$\bar{F}_n := \frac{\int dq F_n(q)}{\sigma(q)}_{F_n} ,$$

$$\sigma(q)|_{F_n} \equiv \left( \frac{\int dq q^2 F_n(q)}{\int dq F_n(q)} - \left( \frac{\int dq q F_n(q)}{\int dq F_n(q)} \right)^2 \right)^{1/2} .$$

$\bar{F}_n$, like $F_n(q)$, is 0 for disordered and nonzero for lamellar configurations. Note that $F(q)$ is not the structure factor of the melt but simply the Fourier transform of $\bar{\phi}_A(r)$. In fully equilibrated simulations, the structure factor can be calculated from averaged expressions containing the fields $W_-$ and $W_+$ [89]. This will be done in Chapter 5 for a few sample parameters.

At dimensionless number density $C=50$, the shift of the order-disorder transition was examined in the canonical ensemble on a $32\times32$ lattice with $dx=0.625$ for $\phi_H = 0$ and 0.2, and $dx=0.78125$ for $\phi_H = 0.4$, 0.6, and 0.7. These $dx$ values were chosen such that a single lamella was several pixels wide, which is a precondition for the evaluation
Figure 4.7: Snapshots at C=50, $\phi_H = 0$, and $\chi N = 11.1, 11.2, 11.3, 11.4$, and 11.5. White points: $0 < \phi_A < 0.49$, red points: $0.49 \leq \phi_A < 0.51$, black points: $0.51 \leq \phi_A \leq 1$. All runs were started from disordered configurations.

Figure 4.8: Direction persistence as measured by the $\Lambda$ parameter vs. $\chi N$ for different homopolymer volume fractions $\phi_H$ at the order-disorder transition in simulations on $32 \times 32$ lattices. For $\phi_H = 0$ and 0.2, the discretization $dx = 0.625$ was used, otherwise $dx = 0.78125$.

of the $\Lambda$ parameter to work properly. For simplicity, I shall subsequently denote runs that started from a disordered configuration “D-started” and those that started out of a lamellar configuration, “L-started.” In the latter, the lamellar periodicities of the starting configurations reflect the mean-field values at the given parameters. Moreover, in L-started runs, $dx$ was chosen to allow for an integer number of lamellae to fit in the simulation box.

In the case of copolymers only ($\phi_H = 0$), D-started runs at various $\chi N$ yield the snapshots of Fig. 4.7, which were taken after approximately 1 million Monte Carlo steps and after the simulations were equilibrated. It is clearly visible from the images that above a certain threshold $\chi N$, a lamellar phase begins to form spontaneously. I did not observe any hysteresis effects. To quantify the transition, I have plotted $\Lambda$ in Fig. 4.8 and $\bar{F}_2/\bar{F}_4$ in Fig. 4.9. These plots also include curves for the other runs described above, i.e., of $\phi_H$ up to 0.7 and for all of them display a jump in both the $\Lambda$ and $\bar{F}_2/\bar{F}_4$ parameters at the fluctuation-corrected critical $\chi N$. For $\phi_H = 0$, I find the transition to be shifted
4.6. The order-disorder transition

Figure 4.9: $\bar{F}_2$ (solid) and $\bar{F}_4$ (dashed) vs. $\chi N$ for different homopolymer volume fractions $\phi_H$ at the order-disorder transition in the same simulations as in Fig. 4.8.

from the mean-field value of 10.495 to 11.3(1), which is in qualitative agreement with the theoretical prediction of Fredrickson and Helfand [36] as well as in good quantitative agreement with a result obtained earlier by Ganesan and Fredrickson using their complex Langevin method [88]. It is also further evidence that the $W_+$ fluctuations contribute only a minor correction to the partition function.

For homopolymer concentrations $\phi_H$ above 0.7, the configurations did not spontaneously assemble into a lamellar phase for any $\chi N$ in D-started simulations on 32×32 lattices. The configurations can at best be described as very defective lamellae. However, in L-started runs the configurations only broke up at very low $\chi N$ values. For example, at $\phi_H = 0.85$, D-started runs were done for $\chi N \leq 13$, all of which stayed disordered, yet in L-started runs the lamellae only broke up for $\chi N < 12.1$. This obvious discrepancy can be made plausible by two arguments: First, in the D-started runs, as we approach higher $\chi N$ values (which correspond to lower temperatures), the simulations get trapped in configurations with a certain degree of local periodicity yet cannot move through configuration phase efficiently enough to assemble into real lamellae. Second, in the L-started runs, in order to keep $d\chi$ reasonably low (i.e., $\lesssim 0.8$), one can only put a small number of lamellae in the box. In the case of $\phi_H = 0.85$, that number is 3. But this in combination with the periodic boundary conditions imposed on the box in turn seems to artificially stabilize the lamellar phase, which is why I do not see a breaking-up. Evidently, I must use bigger simulation boxes for $\phi_H \gtrsim 0.7$.

Because of the slow ordering at high $\chi N$ described above, I restrict myself to L-started runs in examining the order-disorder transition on 48×48 lattices. Even so, I find that the simulations take an increasingly greater number of Monte Carlo steps to equilibrate the higher $\chi N$. For $\phi_H = 0.7$, a time series of $\bar{F}_2$ is displayed in Fig. 4.14, respectively.
Chapter 4. Fluctuations in the Ternary AB+A+B System

Figure 4.10: Snapshots (48×48 lattice, L-started) at $C = 50$, $\phi_H = 0.7$ and $\chi N = 12.2$, 12.3, 12.4, and 12.5 (from left to right).

Figure 4.11: Snapshots (48×48 lattice, L-started) at $C = 50$, $\phi_H = 0.8$ and $\chi N = 12.5$, 12.7, and 13.0 (from left to right).

Figure 4.12: Snapshots (48×48 lattice, L-started) at $C = 50$, $\phi_H = 0.85$ and $\chi N = 12.5$, 12.7, 13.0, 13.2, and 13.4 (from left to right).

The plot indicates that the transition must be between $\chi N = 12.3$ and 12.5: there occurs a substantial jump in both parameters, caused by a breaking-up of the initial lamellar configuration. This is in good agreement with the result from the 32×32 lattice.

The $\bar{F}_2$ graphs at the transition, especially for $\chi N = 12.3$, display a special feature: a periodic oscillatory motion once the run is equilibrated. The time scale of this oscillation can be explained as follows. Close to the transition, the lamellae do not easily dissolve completely. Instead, defects develop which may open and close periodically, often alternating the continuation of a given contour length between two defective lamellar branches. This oscillation is reflected in $\bar{F}_2$.

For $\phi_H = 0.8$, the corresponding time series are displayed in Figs. 4.15 and 4.16. $\chi N = 12.5$ is clearly disordered, and the transition is inferred to occur below $\chi N = 13$. Another run for $\phi_H = 0.85$ indicates that that transition is at approximately $\chi N = 13$, though even after 3 million Monte Carlo steps the configurations retained a strong correlation to the seed configuration (see Figs. 4.17 and 4.18). The slope of the resulting fluctuation-corrected order-disorder transition is in good qualitative agreement with the experimental result of Hillmyer et al. for a PEE/PDMS/PEE-PDMS system of intermediate molecular weights [74]. In particular, the shape of the cusp-like region of bicontinuous microemulsion identified therein is here reproduced computationally for the first time.
4.6. The order-disorder transition

Figure 4.13: Direction persistence, $\Lambda$, at $\phi_H = 0.7$ in simulations on a $48 \times 48$ lattice that started from lamellar configurations.

Figure 4.14: $\bar{F}_2$ at homopolymer volume fraction $\phi_H = 0.7$ for different $\chi N$ in simulations on a $48 \times 48$ lattice that started from lamellar configurations.
Chapter 4. Fluctuations in the Ternary AB+A+B System

Figure 4.15: $\Lambda$ at $\phi_H = 0.8$ in simulations on a 48×48 lattice that started from lamellar configurations.

Figure 4.16: $\bar{F}_2$ at homopolymer volume fraction $\phi_H = 0.8$ for different $\chi N$ in simulations on a 48×48 lattice that started from lamellar configurations.
4.6. **The order-disorder transition**

Figure 4.17: $\Lambda$ at $\phi_H = 0.85$ in simulations on a $48 \times 48$ lattice that started from lamellar configurations.

Figure 4.18: $\bar{F}_2$ at homopolymer volume fraction $\phi_H = 0.85$ for different $\chi N$ in simulations on a $48 \times 48$ lattice that started from lamellar configurations.
I conclude that on the one hand, using a bigger lattice has alleviated the artificial stabilizing effect of the periodic boundary conditions on the lamellae. On the other hand, however, I still encounter numerical freezing effects for $\chi N \gtrsim 13$ at $C = 50$, which prohibit the examination of this region of the phase diagram.

### 4.7 The demixing transition

Next, I examine the onset of the phase-separated region. This is best done in the grand canonical ensemble; a D-started run at parameters in the phase-separated region will eventually undergo a spontaneous symmetry breaking and end up in a configuration dominated by either A or B segments. Moreover, it is sufficient to do these runs on $32 \times 32$ lattices as no complex morphologies are investigated here. To quantify this behavior, I analyze the total difference of normalized A and B densities:

$$\Delta \phi := \frac{1}{V} \left| \int \mathrm{d}r (\bar{\phi}_A(r) - \bar{\phi}_B(r)) \right|.$$  \hspace{1cm} (4.53)

First, I look at the demixing in a system consisting only of A and B homopolymers ($\phi_H = 1$). In this case, Fig. 4.19 shows how the order parameter $\Delta \phi$ grows from zero above approximately $2.04/\alpha$, shifted up from the mean-field value of $2/\alpha$.

![Figure 4.19: Difference of A and B monomer densities, $\Delta \phi$, vs. $\chi N$ in a binary system of A and B homopolymers in the grand canonical ensemble.](image)

For various $\chi N$, Fig. 4.20 displays $\Delta \phi$ for relative homopolymer fugacities $z$ (corresponding to $\phi_H < 1$ in the canonical ensemble) around the transition. When translated into $\phi_H$ values in the canonical ensemble, the numerical accuracy obtained in the present
Finally, the above results for the order-disorder (ODT) and disorder/two-phase transitions are put together in the fluctuation-corrected phase diagram for $C=50$ (Fig. 4.23). As concerns the order of the revised ODT, I did sweeps in the grand canonical ensemble across the ODT for homopolymer fugacities $z=1$ (Fig. 4.21) and $z=2$ (Fig. 4.22). Both plots are consistent with a very weakly first-order transition. In either case, the coexistence region has a width of no more than approximately 0.001 in $\phi_H$, which is comparable to the line thickness of the ODT in the phase diagram (4.23).
Figure 4.21: Homopolymer concentration, \( \phi_H \), vs. \( \chi N \) at \( z = 1 \) across the order-disorder transition. The dotted line indicates the \( \chi N \) coordinate of the ODT \( \pm 0.1 \).

Figure 4.22: Homopolymer concentration, \( \phi_H \), vs. \( \chi N \) at \( z = 2 \) across the order-disorder transition. The dotted line indicates the \( \chi N \) coordinate of the ODT \( \pm 0.1 \).
4.9 Summary and conclusions

In this chapter, I have introduced a Monte Carlo method of evaluating the full partition function of a system of polymeric blends subject to thermal fluctuations. Small lattices (32×32) sufficed up to $\phi_H = 0.7$, above which bigger (48×48) lattices had to be used. In the low $\phi_H$ region, an alternative complex Langevin method introduced by Ganesan and Fredrickson [88] is consistent with our results, which, if corroborated, indicates that the $W_+$ fluctuations neglected in my simulations contribute only a minor correction. I have presented further evidence for this circumstance obtained from empirical test runs. Regrettably, a full analysis of the $W_+$ fluctuations was rendered impossible by the occurrence of a non-positive-definite weighting factor (the sign problem). A fluctuation-corrected phase diagram has been presented for C=50, which corresponds well to the PEE/PDMS/PEE-PDMS system of intermediate polymerization indices examined in a series of recent experiments [71, 72, 73, 74], providing further corroboration of the existence of a region of bicontinuous microemulsion between the lamellar and phase-separated regions.

Figure 4.23: The fluctuation-corrected phase diagram at C=50. The dotted lines indicate the mean-field result of Fig. 2.1. The order-disorder transition is weakly first-order, but the coexistence region is too small to be displayed.

4.9 Summary and conclusions

In this chapter, I have introduced a Monte Carlo method of evaluating the full partition function of a system of polymeric blends subject to thermal fluctuations. Small lattices (32×32) sufficed up to $\phi_H = 0.7$, above which bigger (48×48) lattices had to be used. In the low $\phi_H$ region, an alternative complex Langevin method introduced by Ganesan and Fredrickson [88] is consistent with our results, which, if corroborated, indicates that the $W_+$ fluctuations neglected in my simulations contribute only a minor correction. I have presented further evidence for this circumstance obtained from empirical test runs. Regrettably, a full analysis of the $W_+$ fluctuations was rendered impossible by the occurrence of a non-positive-definite weighting factor (the sign problem). A fluctuation-corrected phase diagram has been presented for C=50, which corresponds well to the PEE/PDMS/PEE-PDMS system of intermediate polymerization indices examined in a series of recent experiments [71, 72, 73, 74], providing further corroboration of the existence of a region of bicontinuous microemulsion between the lamellar and phase-separated regions.
Chapter 5

The Formation and Structure of the Microemulsion Phase

5.1 Introduction

Having established my simulation method for fluctuations in polymer melts along with a revised phase diagram for a two-dimensional ternary AB+A+B blend at an intermediate dimensionless polymer density of $C=50$ and a homopolymer vs. copolymer length ratio of $\alpha = 0.2$ in the last chapter, the focus in this chapter shall be on the properties of the microemulsion phase. I will first elucidate the mechanism by which it forms by means of a curvature analysis of the microdomain boundaries. In this context, evidence is presented to the effect of a subclassification of the microemulsion. Then, I will introduce parameters derived from Minkowski functionals to further characterize the morphological behavior in this region. Finally, two-dimensional structure factors for the lamellar, microemulsion, and (structureless) disordered regimes will be shown. The methods of analysis presented in this chapter all require as input configurations containing density (or field) distributions. Those based on density distributions should in principle also be applicable to digitized snapshots obtained from experiments. Here, however, I restrict myself to using as input data results produced with the field-theoretic Monte Carlo method presented in Chapter 4.

5.2 The formation of the microemulsion phase

As a starting point in undertaking graphical analyses of images, we need to find the preferential length scale of a density distribution $\tilde{\phi}_A(r)$. This is equivalent to finding the preferential wave vector of its Fourier transform, $F(q)$. To this end, I use the anisotropy function of definition (4.50) with anisotropy index $n = 0$:

$$F_0(q) := \frac{1}{2\pi} \int_0^{2\pi} d\phi |F(q)|^2. \tag{5.1}$$

The technicalities of calculating $F_0(q)$ require $q$ to be binned in a histogram and thus approximated by discrete values (in practice, by multiples of $2\pi/L$, where $L$ is the box length). This makes the $q$ coordinate of the maximum of $F_0(q)$ a poor candidate to be
taken as the preferential wave vector. A smoother procedure is to use
\[ \bar{q} = \frac{\int dq \, q F_0(q)}{\int dq \, F_0(q)}. \] (5.2)

Now, the preferential length scale, \( L_0 \), is defined as
\[ L_0 := \frac{2\pi}{\bar{q}}. \] (5.3)

Further, I define the mean curvature diameter, \( D_C \), of the boundaries of A and B microdomains in a black-and-white image, in which a lattice point is painted black (“A”) if \( 0.5 < \bar{\phi}_A \leq 1 \), and white (“B”) if \( 0 \leq \bar{\phi}_A \leq 0.5 \):
\[ D_C := 2 \left[ \frac{1}{L_c} \int ds \left( \frac{dt}{ds} \right)^2 \right]^{-\frac{1}{2}}. \] (5.4)

\( L_c \) is the sum of all contour lengths of the microdomain boundaries, and \( t \) is the tangent vector at a given coordinate \( s \) along the contour. For a detailed explanation of the algorithm I used to calculate \( D_C \), see Appendix A.

Following the text of this section, configurational snapshots and plots of equilibrated averages of \( L_0 \) and \( D_C \) are displayed for various homopolymer volume fractions, \( \phi_H \), at \( \chi N = 12.5, 12, \) and \( 11.5 \). Note that red pixels in the snapshots denote points for which \( 0.49 < \bar{\phi}_A < 0.51 \). Also, the deviation of \( L_0 \) from the corresponding mean-field values, \( L_{MF} \), (cf. Chapter 2) is shown. All plots are for a dimensionless polymer number density of \( C = 50 \). The observations from these plots are:

- In all three cases, i.e., \( \chi N = 12.5, 12, \) and \( 11.5 \), \( D_C \) is in good approximation constant over an extended \( \phi_H \) range (Figs. 5.5, 5.9, and 5.13). This is plausible because the curvature is induced by the copolymers, which are located predominantly at the interfaces between microdomains. It should thus be independent of the amount of homopolymers. Nevertheless, \( D_C \) gets somewhat smaller as \( \chi N \) is decreased; the higher temperatures that correspond to lower \( \chi N \) facilitate the bending of the microdomain interfaces.

- At \( \chi N = 12.5 \), coming from the lamellar phase at low \( \phi_H \), \( D_C \) becomes comparable in size to the preferential as well as mean-field length scales, \( L_0 \) and \( L_{MF} \), respectively, around \( \phi_H \approx 0.75 \), which is when the lamellae begin to break up. This is in good quantitative agreement with the location of the fluctuation-corrected order-disordered transition of Fig. 4.23. At \( \chi N = 12 \) and \( 11.5 \), \( D_C \) is well below both the mean-field periodicities, \( L_{MF} \), and the preferential length scales, \( L_0 \), in the \( \phi_H \) range examined: these configurations are in the disordered phase. I have thus identified the mechanism by which fluctuations generate the microemulsion.

- At \( \chi N = 12.5 \) and \( 12 \), \( L_0 \) begins to deviate from \( L_{MF} \) at a certain \( \phi_H \), signaling the departure from defect-driven behavior and the onset of a more genuine morphology within the microemulsion phase. The effect is stronger for \( \chi N = 12.5 \) than for \( \chi N = 12 \). At \( \chi N = 11.5 \), it is barely noticeable anymore, and \( L_0 \) coincides within error bars with \( L_{MF} \). However, the higher \( \chi N \), the more smeared out \( L_0 \) becomes.
5.2. **The Formation of the Microemulsion Phase**

- According to a popular definition of a microemulsion [95, 96], it is characterized by the existence of a preferential length scale within the disordered phase. In this sense, all configurations displayed in Figs. 5.2, 5.6, and 5.10 are microemulsions. The previous result, however, suggests a more diversified classification into (a) a defect-driven region and (b) a “genuine” microemulsion. I shall denote them \( D_{\mu E} \) and \( G_{\mu E} \), respectively. In the below redrawing (Fig. 5.1) of the phase diagram Fig. 4.23, I have indicated the approximate location of the conjectured transition between the two, as marked by the occurrence of nonzero values for \( L_0 - L_{MF} \) in Figs. 5.4 and 5.8. Presumably, the \( G_{\mu E} \) phase coincides with the “bicontinuous microemulsion” \( (B_{\mu E}) \) phase indicated in the experimental three-dimensional phase diagram (Fig. 4.1). However, for topological reasons, bicontinuity can only be approximated in two dimensions, hence the different names.

![Figure 5.1: Substructure of the microemulsion. \( D_{\mu E} \): defect-driven, \( G_{\mu E} \): genuine microemulsion morphology, \( 2\phi \): phase-separated region.](image)

- The attentive reader will notice a certain similarity of some configurations, especially those at higher \( \chi_N \). The corresponding simulations were indeed started independently of each other, but with the same seed number for the random number generator (which is perfectly fine). The Monte Carlo simulations in this chapter were run up to approximately 1.5 million Monte Carlo steps, including equilibration times of up to a few 100,000 steps. As a general rule, the more disordered the configurations the shorter was the equilibration phase.
\( \chi N = 12.5 \)

Figure 5.2: Snapshots of D-started runs at \( C = 50, \chi N = 12.5, \phi_H = 0.74, 0.78, 0.8, \) and 0.82 (first row from left), and \( \phi_H = 0.84, 0.86, 0.88, \) and 0.9 (second row from left).

Figure 5.3: Preferential length scale, \( L_0 \), at \( \chi N = 12.5 \), averaged over the equilibrated parts of simulations on a 48\( \times \)48 lattice. Filled and empty dots correspond to disordered and lamellar initial conditions, respectively. For error bars, see Fig. 5.4. The solid line shows the lamellar periodicities according to mean-field theory (cf. Fig. 2.1).
5.2. The formation of the microemulsion phase

Figure 5.4: Difference of the preferential length scales, $L_0$, from the corresponding mean-field periodicities at $\chi N = 12.5$ (equilibrated averages over the same simulations as in Fig. 5.3).

Figure 5.5: Curvature diameter, $D_C$, at $\chi N = 12.5$ vs. homopolymer volume fraction, $\phi_H$ (equilibrated averages over the same simulations as in Fig. 5.3). Filled and empty dots correspond to disordered and lamellar initial conditions, respectively. The solid line shows the lamellar periodicities according to mean-field theory.
$\chi N = 12$

Figure 5.6: Snapshots of D-started runs at $C = 50$, $\chi N = 12$, $\phi_H = 0.74$, 0.78, 0.8, and 0.82 (first row from left), and $\phi_H = 0.84$, 0.86, 0.88, and 0.9 (second row from left).

Figure 5.7: Preferential length scale, $L_0$, at $\chi N = 12$, averaged over the equilibrated parts of D-started simulations on a $48\times48$ lattice. The solid line shows the lamellar periodicities according to mean-field theory (cf. Fig. 2.1).
5.2. The formation of the microemulsion phase

Figure 5.8: Difference of the preferential length scales, \( L_0 \), from the corresponding mean-field periodicities at \( \chi N = 12 \) (equilibrated averages over the same simulations as in Fig. 5.7).

Figure 5.9: Curvature diameter, \( D_C \), at \( \chi N = 12 \) vs. homopolymer volume fraction, \( \phi_H \) (equilibrated averages over the same simulations as in Fig. 5.7). The solid line shows the lamellar periodicities according to mean-field theory.
\( \chi N = 11.5 \)

Figure 5.10: Snapshots of D-started runs at \( C = 50, \chi N = 11.5, \phi_H = 0.82, 0.83, 0.84, \) and 0.85 (first row from left), and \( \phi_H = 0.86, 0.87, 0.88, \) and 0.89 (second row from left).

Figure 5.11: Preferential length scale, \( L_0 \), at \( \chi N = 11.5 \), averaged over the equilibrated parts of D-started simulations on a 48\( \times \)48 lattice. The solid line shows the lamellar periodicities according to mean-field theory (cf. Fig. 2.1).
5.2. The formation of the microemulsion phase

Figure 5.12: Difference of the preferential length scales, $L_0$, from the corresponding mean-field periodicities at $\chi N = 11.5$ (equilibrated averages over the same simulations as in Fig. 5.11).

Figure 5.13: Curvature diameter, $D_C$, at $\chi N = 11.5$ vs. homopolymer volume fraction, $\phi_H$ (equilibrated averages over the same simulations as in Fig. 5.11). The solid line shows the lamellar periodicities according to mean-field theory.
5.3 The Minkowski functionals with inflated pixels

5.3.1 The Minkowski functionals

The Minkowski functionals [97, 98] of an image are the area covered by black pixels, $A$, the combined circumference, $L_c$, of the black areas (or interchangeably of the white areas), and the Euler characteristic, $\chi_E$, of the image. The latter is a topological property known from differential geometry to equal the number of objects minus the number of holes in the objects. It is invariant under reshaping of the image as long as the topology is left unchanged, or, put more technically, under a retriangulation of the associated simplicial complex. The area, $A$, is straightforward to calculate – simply count the black pixels. For the Euler characteristic, I used a routine by K. Michielsen [98].

As far as the circumference is concerned, one needs to exercise some care. A naive circumference algorithm would be one which simply integrates over pieces of the lattice that border the black areas. In this algorithm, the direction of segments is always either horizontal or vertical. If, for instance, the bitmap lattice is meant to display a circle of radius $r$, it will calculate the circumference to be $8r$, not $2\pi r$ as it should. This result is independent of the lattice resolution! A popular method of remedying this artifact is to use this algorithm anyway and then apply a factor $\pi/4$ to the final result. However, for the purposes of distinguishing lamellar-like structures from more disordered ones, this approach is clearly unsatisfactory, as it would not produce invariant results for identical but rotated images. A variety of more sophisticated circumference algorithms exist. Unfortunately, many of them were developed for commercial software packages and so are not easily accessible to the common researcher. The procedure outlined in Appendix A is my own attempt and was used in the present work.

In what follows, the Minkowski functionals shall be evaluated for images in which all black pixels have been replaced with disks of variable radii (“inflated”). To do so, the lattice grid must first be made finer so the disks will be well-defined, i.e., we must replace each pixel with a sublattice of a certain size (I used $16\times16$). To prepare for inflating, each black pixel in the original image is copied to an arbitrary (yet the same for all pixels) position within its corresponding sublattice. Then, disks of any radius can easily be drawn whose center is at the position of the initial black pixels.
5.3. The Minkowski functionals with inflated pixels

5.3.2 Area at inflation radius $L_0/8$, $A_{3/4}$

The first modified Minkowski functional to be looked at I have called $A_{3/4}$. It measures the area fraction occupied by black pixels on the fine grid when an inflation radius of $(L_0/8 + dx/2)$ is used, $dx$ being the side length of a sublattice. In the limit of infinite resolution of the original image, the required inflation radius goes to $L_0/8$. For a perfectly lamellar configuration, it is 0.75, and bigger otherwise. This is because deviations from the lamellar pattern are tantamount to some black pixels reaching into an otherwise mostly white area, which, when painted over by the inflated disks, will disproportionately see its white pixels turned into black. Figs. 5.14, 5.15, and 5.16 show $A_{3/4}$ for various $\phi_H$ at $\chi N = 12.5$, 12, and 11.5, respectively. At $\chi N = 12.5$, we see $A_{3/4}$ starting near 0.75 at $\phi_H = 0.74$, which is very close to both the lamellar phase and the proposed transition between the $D\mu E$ and $G\mu E$ subphases. The more $\phi_H$ is increased, the further away we get from either, and $A_{3/4}$ increases. In Fig. 5.15, $A_{3/4}$ is consistent with a constant in the $D\mu E$ regime, and a positive slope in the $G\mu E$ region. We note that the slope is not as steep for $\chi N = 12$ as for $\chi N = 12.5$, and that $A_{3/4}$ is significantly higher in the former than in the latter, illustrating that we have gone deeper into the disordered phase. Finally, the curve for $\chi N = 11.5$ shows even higher values of $A_{3/4}$ at an even smaller slope than for $\chi N = 12$.

![Figure 5.14: $A_{3/4}$ at $C = 50$ and $\chi N = 12.5$.](image-url)
Figure 5.15: $A_{3/4}$ at $C = 50$ and $\chi N = 12$.

Figure 5.16: $A_{3/4}$ at $C = 50$ and $\chi N = 11.5$. 
5.3.3 Normalized mean circumference, $U$

Next, I define a dimensionless parameter, $U$, to examine the circumference of microdomain boundaries for different morphologies:

$$U := \frac{L_c}{2L} \cdot \frac{L_0}{L},$$

(5.5)

where $L_c$ is the absolute microdomain circumference (for the algorithm I used to calculate it, see Appendix A), $L$ is the box length, and $L_0$ the preferential length scale of a configuration averaged over the equilibrated parts of a simulation. The definition is construed such that $U$ is 1 in a perfectly lamellar configuration and greater than 1 otherwise. Figs. 5.17 and 5.18 are consistent with a nearly constant $U$ in the $D\mu E$ region (only). At the high $\phi_H$ end of the $G\mu E$ region, we observe a positive slope of $U$ against $\phi_H$, signaling that the phase-separated region is approached. Like $A_{3/4}$, the $U$ values are greater while the positive slope discussed above is flatter for $\chi N = 12$ than for $\chi N = 12.5$. At $\chi N = 11.5$ (Fig. 5.19), large errors prevent any classification. Interestingly, at $\chi N = 12$, $U$ features a negative slope between the defect-driven region and that close to phase separation behavior, i.e., fewer and fewer defects contribute less and less to the circumference, $L_c$, while at the same time the increase in $L_0$ is still moderate.

![Figure 5.17: $U$ at $C = 50$ and $\chi N = 12.5$.](image-url)
Figure 5.18: $U$ at $C = 50$ and $\chi N = 12$.

Figure 5.19: $U$ at $C = 50$ and $\chi N = 11.5$. 
5.3.4 Euler characteristic

The preceding parameters were averaged over the equilibrated parts of simulations. By contrast, the following two parameters, which make use of the Euler characteristic, $\chi_E$, of an image, have been evaluated for single snapshots (of equilibrated configurations) only.

In a perfectly lamellar configuration, the inflation radius at which two opposite fronts of disks meet is $L_{MF}/4$. Because of the underlying lattice structure, there will be many small white holes between adjacent disks. In this situation, the number of black objects is 1, and the number of holes increases sharply (Fig. 5.20). This corresponds to a (negative) peak in $\chi_E(r)$. In analogy to my definition (5.2) of $\bar{q}$, I introduce the preferential inflation radius at which the disks touch as

$$\bar{r}_\chi = \frac{\int dr \ r \chi_E(r)}{\int dr \chi_E(r)}.$$  \hspace{1cm} (5.6)

For comparison with lamellar periodicities, the relevant quantity is

$$L_{\bar{r}_\chi} := 4 \left( \bar{r}_\chi - \frac{d}{2} \right).$$  \hspace{1cm} (5.7)

From Fig. 5.21 we see that $L_{\bar{r}_\chi}$ increases steadily with increasing $\phi_H$ along the examined range. Moreover, it deviates the more from $L_{MF}$, and thus also $L_0$, (Figs. 5.3, 5.7, and 5.11) the lower $\chi N$. This result reflects the fact that the configurations become more disordered, and hence smaller inflation radii are needed to make the disks touch.

![Figure 5.20: Euler characteristic, $\chi_E$, vs. inflation radius, r, for a sample configuration at $\chi N = 12.5$ and $\phi_H = 0.82$.](image-url)
Figure 5.21: $L_{\bar{r}_\chi}$ vs. homopolymer volume fraction, $\phi_H$. Dotted lines correspond to mean-field periodicities.

Figure 5.22: $A_{\bar{r}_\chi}$ vs. homopolymer volume fraction, $\phi_H$. 
The area fraction of the lattice occupied by black pixels at inflation radius \( \bar{r}_\chi \), denoted \( A_{\bar{r}_\chi} \), is displayed in Fig. 5.22. It was extrapolated from discrete \( A(r) \) functions and approximated by the nearest multiple of 0.01. We observe a drop as we approach the phase-separated region, while smaller \( \chi N \) translate into smaller overall \( A_{\bar{r}_\chi} \) levels. The drop at high \( \phi_H \) illustrates the greater variance of length scales associated with this regime.

### 5.4 The Lifshitz line

The Lifshitz line [95, 96] is where the peak in the structure factor begins to deviate from zero, i.e., for \( \chi N \) below this line, there is no preferential length scale in the system. Although structure factors of sample configurations will be presented in the next section, their calculation is time-consuming and does not lend itself to being used many times over. Therefore, I shall instead examine the existence of a preferential length scale with the anisotropy parameter, \( \bar{F}_n \), as introduced in (4.51), setting \( n = 0 \). This yields the following definition for \( \bar{F}_0 \):

\[
\bar{F}_0 \ := \frac{\int dq F_0(q)}{\sigma(q) |F_0|},
\]

where \( F_0(q) \) is the function defined in (5.1).

From Fig. 5.24, in which \( F_0(q) \) is displayed for various \( \chi N \) at \( \phi_H = 0.82 \) (cf. Fig. 5.23), it can be seen that \( F_0(q) \) has a pronounced peak for \( \chi N \sim 11 \), below which point the curves become markedly flatter in the low \( q \) region, indicating the location of the Lifshitz line. The corresponding plots for \( \bar{F}_0 \) are shown in Fig. 5.25: the curves for the \( \phi_H \) values examined all start to deviate from zero in the region around \( \chi N = 11 \). Furthermore, a plot of \( \bar{F}_0 \) vs. \( \phi_H \) at \( \chi N = 11.5 \) suggests that the Lifshitz line has a convex shape in the phase diagram of Figs. 5.1 and 4.23.

![Figure 5.23: Snapshots of D-started runs at \( C = 50, \phi_H = 0.82 \), and \( \chi N = 12.5, 12, 11.5, \) and 11 (first row from left), and \( \chi N = 10.5, 10, 9, \) and 8 (second row from left).]
Figure 5.24: Equilibrated averages of $F_0(q)$ over the same simulations as in Fig. 5.23.

Figure 5.25: Equilibrated averages of $\bar{F}_0$. 
5.5 Density profiles

In Figs. 5.26 and 5.27, sample one-dimensional cross sections of individual snapshots are shown for the lamellar and disordered phases, respectively. The lamellar profile indicates good agreement with the corresponding mean-field periodicity while incorporating an incidental defect in the center. From the profiles in the disordered phase, we see that the difference in A and B densities, and hence the difference of $\phi_A$ from 0.5, become less and less pronounced the deeper one goes into the disordered regime.

5.6 Structure factors

The structure factor of a configuration, which in principle is different from the Fourier transform of a density distribution, can be calculated from the Fourier transform of the density correlation function, which in turn can be obtained from the fluctuating $W_A(r) \equiv W_A^+(r) + W_A^-(r)$ field via a formula derived in [89]. Thus,

$$S(q) = \mathcal{F} \left[ \frac{1}{V} \int dr_0 \left\langle \hat{\rho}(r_0) \hat{\rho}(r_0 + r) \right\rangle - \left\langle \hat{\rho}(r_0) \right\rangle \left\langle \hat{\rho}(r_0 + r) \right\rangle \right],$$

(5.9)

with

$$\left\langle \hat{\rho}(r) \hat{\rho}(r') \right\rangle - \left\langle \hat{\rho}(r) \right\rangle \left\langle \hat{\rho}(r') \right\rangle = \frac{2C}{\chi N} \delta(r-r') - \frac{4C^2}{(\chi N)^2} \left[ \left\langle W_A^+(r) W_A^-(r') \right\rangle - \left\langle W_A^+(r) \right\rangle \left\langle W_A^{-}(r') \right\rangle \right].$$

(5.10)

The delta term is a spurious self-interaction term which is discarded.

In Figs. 5.28-5.31, two-dimensional structure factors for lamellar, microemulsion, and (structureless) disordered configurations are shown. The lamellar structure factor features a double peak characteristic of linearly ordered shapes. Fig. 5.29 illustrates that in a microemulsion, anisotropy is lost yet a preferential length scale does exist, as evidenced by a ring-shaped region of maxima. As we progress deeper into the disordered phase at lower $\chi N$, the ring is first smeared out (Fig. 5.30) before finally giving way to a shapeless low-amplitude noise (Fig. 5.31).

5.7 Summary and conclusions

In this chapter, I have identified the mechanism underlying the formation of the microemulsion phase in ternary AB+A+B polymeric blends: the lamellae break up when the curvature diameter of the microdomain boundaries becomes comparable to the periodicity of the lamellar phase. I have further shown that the preferential length scale in the system deviates from its mean-field equivalent in a parameter subspace of the microemulsion only, demonstrating that the microemulsion region is divided into a defect-driven and a “genuine” part. The approximate location of the Lifshitz line, i.e., the locus of points where the peak in the structure factor starts to deviate from zero (and which signals the onset of microemulsion behavior), was argued to be at segregations, $\chi N$, well below the range examined in the rest of this chapter (and in the previous). Moreover, parameters based on Minkowski functionals were introduced to examine morphological properties of the microemulsion phase.
84 Chapter 5. The Formation and Structure of the Microemulsion Phase

Figure 5.26: 1d cross section of $\bar{\phi}_A$ at $\phi_H = 0.7$ and $\chi N = 12.5$ (black line), mean-field profile with same parameters (red line).

Figure 5.27: 1d cross sections of $\bar{\phi}_A$ at $\phi_H = 0.82$. 
Figure 5.28: Structure factor of the lamellar phase at $C=50$, $\phi_H = 0.7$, and $\chi N = 12.5$. 
Figure 5.29: Structure factor of the microemulsion phase at \( C = 50 \), \( \phi_H = 0.82 \), and \( \chi_N = 12.5 \).
Figure 5.30: Structure factor of the disordered phase at $C=50$, $\phi_H = 0.82$, and $\chi N = 11$. 
Figure 5.31: Structure factor of the disordered phase at $C=50$, $\phi_H = 0.82$, and $\chi N = 10$. 
Chapter 6

Summary

In the present thesis, two extensions of the self-consistent field theory (SCFT) for polymer blends were introduced, evaluated, and the results analyzed. Throughout, polymers were modeled as continuous paths in space. At the outset of SCFTs, the partition functions of the models contain microscopic density operators and interaction functionals based on them. By applying appropriate transformations, however, the representation of the system is changed from a density-based to a field-based one, i.e., in the transformed partition function, individual polymers interact with effective external potential fields generated by the ensemble of the polymers. The theory thus changes, formally, from a multi-chain to a single-chain formalism. Mean-field approximations of such theories are obtained by discarding all contributions to the partition function but the saddle point of the Hamiltonian with respect to both the fields and the densities, of which in turn one must find self-consistent solutions. The fields and densities are governed by a coupled set of equations whose evaluation requires the numerical solution of diffusion equations akin to the Schrödinger equation used in quantum mechanics.

The first extension of SCFT consisted in applying its concepts to the wormlike chain model for semiflexible chains. I introduced a three-dimensional theory based on spherical-harmonic expansions as well as a one-dimensional theory making use of Legendre expansions of the relevant quantities. Using the latter, I showed that in dilute solutions of rod-coil block copolymers, an Onsager-type excluded-volume interaction term is sufficient in order for smectic phases to form, driven solely by entropic effects, above a critical polymer number density over a wide range of rod fractions. These smectic phases possess an intermediate degree of interdigitation of the layers, placing them between the more commonly known smectic monolayer and bilayer phases. The polymer number density, $G$, in this theory acts both as a global prefactor to the Hamiltonian and as an effective factor in the steric interaction strength. As such, one of its functions is not independent of the other so that it is not a Ginzburg parameter with which fluctuations could be controlled.

The second extension was aimed at including thermal field fluctuations around the saddle point of a Gaussian-chain ternary AB+A+B system. In a first step, I used a Fourier-space implementation of SCFT to calculate mean-field phase diagrams for ternary mixtures with homopolymer vs. copolymer length ratios, $\alpha$, of 0.2 and 0.5, and thus established a region of three-phase coexistence (L+A+B) around where a previous publication had (erroneously) put a continuous unbinding transition. The three-phase region extends up to a Lifshitz critical point (LP), where the length scale of the lamellar mesostructures
diverges. However, as previously demonstrated in a series of experiments on this system, the LP is destroyed by fluctuations and gives way to a region of bicontinuous microemulsion in its vicinity.

In this thesis, I introduced a field-theoretic Monte Carlo simulation method for field fluctuations. In it, two types of fields occur: $W_+$ corresponds to the total density fluctuations, and $W_-$ to the composition fluctuations. The $W_-$ fluctuations are real, whereas those of $W_+$ are imaginary, which leads to a reweighting technique with a non-positive-definite weighting factor. This problem, which has an analog in the sign problem of fermionic quantum field theories, prohibits the Monte Carlo simulation of the full Hamiltonian if $W_+$ fluctuations are to be included. However, in the particular case of (on average) incompressible melts studied here, these total density fluctuations were heuristically argued to decouple from the dominating composition fluctuations.

Proceeding with two-dimensional simulations of $W_-$ fluctuations only, I established a fluctuation-corrected phase diagram at dimensionless polymer density $C = 50$, which was in good qualitative agreement with that obtained from recent experiments on the PDMS-PEE/PDMS/PEE system with intermediate polymerization indices. An exact comparison of theoretical and experimental results is impeded by the difficulties arising from matching the temperatures used in experiments with the (inversely) corresponding theoretical segregation strengths, $\chi N$. Contrary to $G$ in the rod-coil theory, the parameter $C$, though equivalent by definition, here is a Ginzburg parameter in the sense that it can be used to smoothly phase in and out deviations from the saddle point, with the limit $C \to \infty$ corresponding to the mean-field result. The order-disorder transition, which is second-order in the mean-field approximation, is known from a theoretical result to be weakly first-order if fluctuations are accounted for. Nevertheless, the coexistence region was shown here to be very narrow.

In the next step, the mechanism responsible for the formation of the microemulsion was shown to be intimately related to matching length scales of the curvature of microdomain boundaries and of the lamellar periodicity. Moreover, I showed that the microemulsion can be subclassified into a defect-driven region ($D\mu E$), where the preferential length scales of the mesostructures still coincide with the corresponding mean-field length scales, and a “genuine” region ($G\mu E$), at whose boundary the former begin to deviate from the latter. Variations of the two-dimensional Minkowski functionals area, circumference, and Euler characteristic, were introduced and employed for further structural analysis of microemulsions. Finally, the Lifshitz line, i.e., the locus of points at which the peak in the structure factor begins to deviate from zero, marking the occurrence of a preferential length scale, is at $\chi N$ values well below the cusp-like part of the microemulsion region examined in this thesis, which reinforces the validity of the presented results. In a possible extension of this work, one should expect the microemulsion region (“channel”) to widen for polydisperse copolymers. Also, preliminary results have shown that at lower $C$ parameters, the order-disorder transition is shifted to higher segregation strengths, $\chi N$, as would be expected.

The simulation programs as well as most of the graphical analysis tools were developed and coded by myself. They are described and listed in Appendices A and B. A generous allocation of computing time on the Cray T3E-1200 and Cray T3E-600 at Forschungszentrum Jülich is acknowledged. Further, my work was supported through funds from the Deutsche Forschungsgemeinschaft (DFG).
Appendix A

Numerical Methods

This appendix deals with a number of numerical methods employed to obtain the results of this thesis. They include iteration schemes, diffusion-equation solvers, and morphological analysis tools.

We start by looking at the self-consistent field equations (2.66) and (2.72). The general program to solve these coupled equations is as follows.

(0) make an initial guess for the densities and calculate initial fields via (2.72).

(1) from the current fields, calculate the propagators $q_j$

(2) from the propagators, calculate a new set of fields $(W_-, W_+, \xi)$

(3) mix the new and old fields efficiently, then go back to (1)

When running fluctuation simulations, a given $W_-$ is fixed while solving the equations in order to find the partial saddle point in $W_+$. Here, the initial guess is not made via the densities but by starting with the old $W_+$ (and $\xi$) from the last (successful) Monte Carlo step. Hence, in this case:

(0) the initial guess for $W_+$ is the solution for the old $W_-$

(1) from the current $W_+$ and the fixed $W_-$, calculate the propagators $q_j$

(2) from the propagators, calculate a new $W_+$ and $\xi$

(3) mix the new and old $W_+$ efficiently, then go back to (1)

Once the iteration is completed, $W_+$ and $\xi$ differ only by a constant, $\frac{N}{2}$, and a minus sign. There are mainly two problems associated with solving the self-consistent field equations fast yet accurately:

(a) The iteration should converge after as few steps as possible. The essential factor here is the method of mixing the new and old fields in each iteration step. This will form the topic of Section 1 of this appendix.

(b) Calculating the propagators. The problem here is how to best solve a diffusion equation, and various schemes will be discussed in Section 2 of this appendix.
Concerning the analysis of our results, parameters have to be extracted from configurational snapshots that allow for a morphological classification of these configurations. Algorithms to solve this task will be addressed in Section 3 of this appendix.

## A.1 Solution methods for the self-consistent field equations

### A.1.1 Broyden’s method

Broyden’s multisecant method for solving the set of \( 2n \) equations (2.72) for \( n \): number of lattice points in real-space implementations of SCFT, or basis functions in Fourier-space implementations) for the fields \( W_A \) and \( W_B \), is a variant of Newton-Raphson [63]. In that scheme, a set of equations of the form

\[
g(\mathbf{x}) = 0
\]

is solved iteratively by linearization, the linear factor being the Jacobian matrix. Starting from a point (i.e., a given set of values for the fields) \( \mathbf{x}' \) with

\[
g(\mathbf{x}') = y \neq 0
\]

one searches a point \( \mathbf{x} \) for which \( g(\mathbf{x}) = 0 \). Linearization of Eq. (A.1) yields

\[
g(\mathbf{x}) = g(\mathbf{x}' + \delta \mathbf{x}') \simeq g(\mathbf{x}') + \frac{\delta g}{\delta \mathbf{x}}(\mathbf{x}') \cdot \delta \mathbf{x}' = 0.
\]

By inverting the Jacobian matrix \( \frac{\delta g}{\delta \mathbf{x}}(\mathbf{x}') \) we get

\[
\delta \mathbf{x}' = -\left[ \frac{\delta g}{\delta \mathbf{x}}(\mathbf{x}') \right]^{-1} g(\mathbf{x}').
\]

The evaluation of (a finite-difference approximation of) the Jacobian requires the calculation of \( 2n \) sets of \( 2n \) equations, which in total requires the solution of \( 2n \) diffusion equations on the entire lattice. \( \mathbf{x}' \) is then replaced with \( \mathbf{x}'' = \mathbf{x}' + \delta \mathbf{x}' \), and \( g(\mathbf{x}'') \) calculated. This iteration is continued until \( g(\mathbf{x}'') < \epsilon \), where \( \epsilon \) is called the tolerance of the iteration. For \( \epsilon = 10^{-12} \), one needs on the order of a few tens of iteration steps for a full SCFT calculation, and around 10 if one is to calculate the partial saddle point in \( W_+ \) during a simulation (cf. Chapter 4). Instead of calculating the Jacobian at every step, which is necessary in this scheme, one can also iterate it. This is Broyden’s multisecant method (e.g., [63]). It costs much less computing time and yet is accurate enough to converge the iteration in about the same number of steps. It works as follows. One starts with the Jacobian of the first step, \( B_1 \equiv \frac{\delta g}{\delta \mathbf{x}}(\mathbf{x}_1) \) and \( \delta \mathbf{x}_1 = -B_1^{-1} g(\mathbf{x}_1) \), as well as \( \delta g_1 \equiv g(\mathbf{x}_2) - g(\mathbf{x}_1) \) with \( \mathbf{x}_2 = \mathbf{x}_1 + \delta \mathbf{x}_1 \). All subsequent iterations of the Jacobian are given by

\[
B_{i+1} = B_i + \frac{(\delta g_i - B_i \cdot \delta \mathbf{x}_i) \otimes \delta \mathbf{x}_i}{\delta \mathbf{x}_i \cdot \delta \mathbf{x}_i}.
\]

Broyden’s method is a major improvement over Newton-Raphson, but still requires the solution of the initial Jacobian, or of \( 2n \) diffusion equations. On typical lattices with sizes of 1024 or more pixels, this is only feasible for one-time iterations, i.e., mean-field solutions. For the purpose of my Monte Carlo simulation scheme introduced in Chapter 4, however, I have to minimize the computational expense per step. To solve on the order of \( 10^3 \) diffusion equations per Monte Carlo step would clearly be prohibitive. Therefore, I switched to a method which lacks the extreme accuracy of Broyden’s but is much faster.
A.2 Solution methods for parabolic equations

A.1.2 Mixing methods

This alternative way to solve the self-consistent equations is provided by mixing methods. We define the residual vector as

$$F_i := y_i - x_i,$$  \hspace{1cm} (A.6)

where $x_i$ and $y_i$ are the input and output vectors of the $i$th iteration step, respectively. The simplest mixing scheme, aptly called “simple mixing,” now uses a constant mixing parameter, $\beta$, to find $x_{i+1}$:

$$x_{i+1} = x_i + \beta F_i.$$  \hspace{1cm} (A.7)

The optimal $\beta$ needs to be established empirically and is often much smaller than 1. The simple mixing scheme can be improved considerably if one determines a new $\beta$ at every step:

$$x_{i+1} = x_i + \beta_i F_i.$$  \hspace{1cm} (A.8)

In this work, I used the following recipe for determining $\beta_i$. Set $\beta_1 = \beta_{\text{start}}$.

$$\beta_i = \sqrt{\frac{\sum_j (x_j^i - x_j^{i-1})^2}{\sum_j (F_j^i - F_j^{i-1})^2}},$$  \hspace{1cm} (A.9)

where superscript denotes the components of a vector. Moreover, if $\beta_i$ as determined by (A.9) is outside a preset interval [$\beta_{\text{min}}, \beta_{\text{max}}$], it is set to the closest boundary of that interval. Feasible values for $\beta_{\text{start}}, \beta_{\text{min}},$ and $\beta_{\text{max}}$ for the simulation method of Chapter 4 are 0.5, 0.01, and 1, respectively. The wormlike chain calculations of Chapter 3 work best with a fixed $\beta$ of 0.05, however. The tolerance, $\epsilon$, in both cases is around $10^{-3}$. In the former, around 10 iteration steps are needed to achieve this accuracy, whereas in the latter this number goes up to several hundreds (because these are full SCFT solutions, not just updates). Each iteration requires only one solution of a diffusion equation. I therefore trade the superb, though needless, accuracy of Broyden’s method for the speed of mixing schemes. Note that there are limitations to the feasibility of using mixing schemes if a higher accuracy is desired; while $\epsilon \approx 10^{-3}$ is easily obtained, the number of necessary steps for lower $\epsilon$s begins to soar quickly.

The above recipe is similar to a more general scheme known as Anderson mixing. There, $\beta_i$ is calculated from the last $n$ iteration steps by linear optimization, which requires the inversion of an $n \times n$ matrix. In the present work, however, I have found no reduction in the number of iteration steps at $n > 2$. For more information on this scheme, the inclined reader is referred to an article by V. Eyert [99].

A.2 Solution methods for parabolic equations

A.2.1 Finite differencing schemes

Consider the diffusion equation

$$\frac{d}{dt} q(x, t) = \lambda \Delta q(x, t) - W(x)q(x, t).$$  \hspace{1cm} (A.10)

The starting point for any finite differencing scheme is to convert the continuous propagator $q(x, t)$ into a discrete version. This yields $u^n_j$ in one, $u^n_{j,k}$ in two, and $u^n_{j,k,l}$ in
three dimensions. Then, one needs to find a good conversion of the operators in (A.10). The most widely used criterion to decide whether a finite differencing scheme is stable or unstable is that of von Neumann [63]. A scheme is said to be unconditionally stable if the amplification factor (the time evolution of the amplitude of a given Fourier mode) is smaller than 1 without restrictions for all Fourier modes, and conditionally stable if this condition is satisfied only within certain boundaries, e.g., of the discretization parameters ($\Delta t$ and $\Delta x$).

Stability is not to be confused with accuracy. It is conceivable, and indeed demonstrable, that unconditionally stable schemes may feature poor accuracy. The two most efficient schemes presented here, namely Crank-Nicholson and Dufort-Frankel, are unconditionally stable and accurate up to second order in $\Delta t$. In the following presentation of several schemes, we shall give the formulae in one dimension. Generalizations to higher dimensions are straightforward.

**The FTCS scheme**

The simplest way to discretize A.10 is the ‘Forward Time Center Space’ (FTCS) scheme:

$$\frac{1}{\Delta t} \left[ u^{n+1}_j - u^n_j \right] = \frac{\lambda}{(\Delta x)^2} \left[ u^{n+1}_{j+1} - 2u^n_j + u^n_{j-1} \right] - W_j u^n_j$$

(A.11)

or, with $a := \lambda \Delta t / (\Delta x)^2$:

$$u^{n+1}_j = (1 - 2a - W_j) u^n_j + a \left( u^n_{j-1} + u^n_{j+1} \right) .$$

(A.12)

It is stable for

$$2\lambda \frac{\Delta t}{(\Delta x)^2} \leq 1 .$$

(A.13)

The timesteps therefore need to be quite small and the scheme is generally very inefficient. However, it is explicit and can be calculated directly.

**The fully implicit scheme**

Another way of discretizing Eq. (A.10) is not to take the spatial derivative at time $n$ but instead at time $n + 1$. This yields:

$$\frac{1}{\Delta t} \left[ u^{n+1}_j - u^n_j \right] = \frac{\lambda}{(\Delta x)^2} \left[ u^{n+1}_{j+1} - 2u^{n+1}_j + u^{n+1}_{j-1} \right] - W_j u^{n+1}_j$$

(A.14)

or

$$-au^{n-1}_{j+1} + (1 + 2a + W_j) u^{n+1}_j - au^{n+1}_{j-1} = u^n_j$$

(A.15)

or

$$A \cdot u^{n+1} = u^n ,$$

(A.16)

i.e., in order to solve this equation, we have to invert a matrix, $A$. This matrix is either tridiagonal or tridiagonal plus the two opposite diagonal corners, depending on whether we want to solve Eq. (A.16) in a box with walls or with periodic boundary conditions, respectively. For both cases, there are efficient ways to invert $A$ and solve Eq. (A.16). The tridiagonal-plus-opposite-corners case is somewhat more complicated and takes almost
twice as long as the purely tridiagonal case but is still much faster than inverting a general matrix.

The implicit scheme is unconditionally stable for any $\Delta t$ but, due to its lack of time centering, it is only accurate up to $O(\Delta t)$.

**The Crank-Nicholson scheme**

The Crank-Nicholson scheme (CN) combines the stability of the implicit scheme with second-order accuracy in both space and time. One simply averages the explicit and implicit schemes and gets

$$
\frac{1}{\Delta t} \left[ u_{j}^{n+1} - u_{j}^{n} \right] = \frac{\lambda}{2(\Delta x)^{2}} \left[ (u_{j+1}^{n} - 2u_{j}^{n} + u_{j-1}^{n}) + (u_{j+1}^{n+1} - 2u_{j}^{n+1} + u_{j-1}^{n+1}) \right] - \frac{W_{j}}{2} \left[ u_{j+1}^{n+1} + u_{j}^{n+1} \right].
$$

(A.17)

Like Fully Implicit, CN can be solved by the inversion of a tridiagonal matrix. The extension of CN to two-dimensions is called Alternating Direction Implicit (ADI), and is based on the concept of operator-splitting. First, half a timestep is performed in $x$ direction, and then another half-step in $y$ direction [63].

**The DuFort-Frankel scheme**

Since matrix inversion is quite expensive in terms of computing time, one would prefer an explicit scheme, where no such inversion is necessary. A scheme that is explicit but is still second-order accurate in both space and time is the DuFort-Frankel scheme (DF). The tradeoff is that now $u_{j}$ must be known at the last two timesteps. It is therefore referred to as a three-step scheme, as opposed to the two-step schemes presented above. It uses this formula:

$$
\frac{1}{2\Delta t} \left[ u_{j}^{n+1} - u_{j}^{n-1} \right] = \frac{\lambda}{(\Delta x)^{2}} \left[ u_{j+1}^{n} - (u_{j+1}^{n+1} + u_{j-1}^{n+1}) + u_{j-1}^{n} \right] - W_{j}u_{j}^{n}.
$$

(A.18)

DF is unconditionally stable. The first step may be obtained by dividing it into $2^{n}$ substeps (e.g., $n=4$). Then one starts by using FTCS with $\Delta t' = \Delta t / 2^{n}$. Successive doubling of $\Delta t'$ and application of DF is repeated until $\Delta t' = \Delta t$, from whence $\Delta t$ remains constant. In an actual two-dimensional implementation of DF, it turns out that one needs roughly double the amount of timesteps to get the same accuracy as with CN. However, the computing time per step is cut down to one-fourth that of CN.

**A.2.2 Parallelization of finite differencing schemes**

CN/ADI solves (A.10) along one axis at a time and can therefore only be parallelized by slab decomposition (Fig. A.1): every processor computes the solution along one slab. For instance, in two dimensions, $u_{i,j}$ has to be redistributed among the processors after every half-step. This is done by mirroring the matrix along the diagonal and requires a lot of communication between the processors. DF, on the other hand, is explicit and thus allows for geometric decomposition (Fig. A.1). This type of decomposition has the major advantage that every processor is assigned a part of the matrix once and for all. The only
Appendix A. Numerical Methods

Figure A.1: Slab decomposition of a lattice, left, and geometric decomposition, right.

Figure A.2: Relative CPU time per pixel (in an arbitrary time unit) vs. number of pixels in a sublattice for parallel DuFort-Frankel on a Cray T3E-1200.

communication necessary is along the borders of adjacent single-processor submatrices. This can easily be seen from Eq. (A.18).

I implemented DF on a Cray T3E-1200 and found an interesting dependence of the efficiency of the algorithm on the size of the sublattices. In two dimensions, the optimal size is found to be 19×19, and in three dimensions, 7×7×7. In Fig. A.2, the computing time per lattice point is plotted against the size of the sublattice (i.e., the chunk assigned to one processor). From the outset, one would expect this curve to decrease monotonically due to the decreasing communication overhead and to level out for big sublattices. This is by and large the case. However, caching effects can be exploited if the data stored in any one processor does not exceed the size of the cache, and this is the reason why there is a minimum, after all. In actual two-dimensional simulations, I used a sublattice size of 16×16, which is not much slower than 19×19 but better to handle numerically.
A.2.3 The pseudo-spectral method

I initially used the DF scheme to evaluate the diffusion equation (A.10) because of its superiority over CN, let alone the other finite differencing schemes, in terms of speed and parallelizability. However, a different approach turned out to be vastly more accurate. That approach is a pseudo-spectral split-operator method which had been used successfully in the past to solve the time-dependent Schrödinger equation [100], and which was only recently introduced by K. Rasmussen [101] in the context of polymer SCFT. From (A.10) it can be seen that \( q(r,t) \) is related to \( q(r,t+dt) \) by a propagator \( \exp(dt (\triangle - W(r))) \), where \( W(r) \) stands for either \( W_A(r) \) or \( W_B(r) \), depending on the monomer type of a given segment. To \( O(dt^2) \), this can be rewritten as

\[
q(r, t + dt) = \exp(-dt W(r)/2) \exp(dt \triangle) \exp(-dt W(r)/2) + O(dt^3)
\]  
(A.19)

In practice, \( dt \) of course needs to be replaced with some finite \( \Delta t \). From now on, I shall denote both the infinitesimal as well as the finite timestep by \( dt \). The pseudo-spectral method consists of the following steps.

(0) Start from \( q(r, t = 0) = 1 \).

(1) Apply \( \exp(-dt W(r)/2) \) to \( q(r,t) \) in real space at each \( r \).

(2) Fourier transform the result.

(3) Apply \( \exp(dt \triangle) = \exp(-dt (2\pi k/L)^d) \) in \( k \)-space (\( L \): side length of lattice, \( d \): dimensions).

(4) Inverse Fourier transform the result.

(5) Apply the final factor of \( \exp(-dt W(r)/2) \) in real space.

(6) Increment \( t \) by \( dt \) and go back to (1) until \( t \) has reached the end of the interval to be calculated.

This scheme is unconditionally stable in any number of dimensions. Steps (1), (3), and (5) are straightforward and numerically relatively simple. Even here, however, a lot of computing time can be saved by using predefined lists. The factor \( \exp(-dt W(r)/2) \) then only needs to be calculated once after step (0), since \( W(r) \) remains constant for the duration of this scheme. The factor \( \exp(dt \triangle) = \exp(-dt (2\pi k/L)^d) \) of step (3) does not even depend on the fields and so only has to be calculated once at the start of the program.

A more dramatic influence on computing times is exerted by the choice of the Fourier transform algorithm. Many a reader will have heard about Fast Fourier Transform, which has been described in abundance in the literature, e.g., [63]. In one dimension, iterative splitting of the discrete Fourier sum is used to rearrange the input data and apply factors of the form \( \exp(2\pi ik/N) \), \( N \) being the number of lattice points in one direction. This results in the dramatic reduction of the number of operations from \( O(N^2) \) to \( O(N \log_2 N) \). Multi-dimensional FFT algorithms reduce the problem to successive applications of the one-dimensional algorithm. For best efficiency, the lattice size should be a power of 2 in all directions.
Owing to the plethora of applications of FFT, considerable effort has been put into optimizing the algorithm by a number of developers. A freely available software package called FFTW that was developed at MIT sticks out from its competitors by its flexibility and high speed [102]. It is highly optimized with respect to algorithms and adapts itself to a given architecture. Also, there is almost no efficiency loss if the lattice side lengths have one denominator that is not 2, e.g., 48 = 2^4 × 3. FFTW is approximately three times faster than the equivalent Numerical Recipes [63] routine (which does require the lengths to be powers of 2). Parallel routines are provided, as well, which helped in converting our DF-based parallel code to this method. On a 48×48 lattice, I found the following dependence of the number of processors vs. typical times to calculate the diffusion equation.

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>clock time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100.0%</td>
</tr>
<tr>
<td>2</td>
<td>51.9%</td>
</tr>
<tr>
<td>4</td>
<td>21.2%</td>
</tr>
<tr>
<td>8</td>
<td>11.5%</td>
</tr>
<tr>
<td>16</td>
<td>13.5%</td>
</tr>
</tbody>
</table>

Apparently, the routine parallelizes very efficiently up to 8 processors. My implementation is thus for 2×8 processors: one pair of 8 each to calculate the forward and conjugate propagators. 8 jobs for 16 processors each were then combined into a 128-processor job to be run on a Cray T3E-1200 and a Cray T3E-600.

A few words about calculating the complex equivalent of (A.10): While it is marginally faster to use complex FFT routines, this has one big disadvantage: it does not produce purely real propagators (as it should) if the fields are real. This is only fulfilled within numerical accuracy. Therefore, it is better to split the complex diffusion equation into two real ones for the real and imaginary parts, thereby insuring that no artificial imaginary components are produced.

The accuracy of the pseudo-spectral method is astonishing compared to the DF scheme discussed earlier. With all the above strategies implemented, a timestep takes about twice as long as with DF (or half as long as with CN). However, the gain in accuracy more than makes up for that. Testing this method against a Fourier-space implementation of mean-field SCFT, I found that a timestep of Δt ≃ 0.05 is enough to get within one percent of the correct result (with Δx ≃ 0.8). To achieve the same accuracy with DF, Δt must be chosen well below 0.005, i.e., more than an order of magnitude smaller, which translates linearly to computational expense. I conclude that the pseudo-spectral method should be used by anyone doing SCFT calculations in real space. Considering how valuable computing time is on parallel supercomputers, the results presented in Chapters 4 and 5 of this thesis owe their existence in large part to the use of this method.

A.3 Graphical analysis

In order to analyze configuration snapshots, it is often advantageous to take the $\bar{\phi}_A$ distribution and convert its continuous values (between 0 and 1) to black-and-white bitmaps with white pixels for $0 \leq \bar{\phi}_A(x, y) < 0.5$, and black pixels for $0.5 \leq \bar{\phi}_A(x, y) \leq 1$. From these images, we can then extract the following parameters.
A.3. Graphical analysis

Figure A.3: Local contributions to the circumference. The current (black) pixel from step (3) here is painted blue. All diagrams are modulo $\pi/2$ rotations.

A.3.1 Algorithm to calculate the circumference, $L_c$

(1) Besides the original bitmap, $B_1$, define another bitmap, $B_2$, and paint it white, i.e., set all pixels to 0.

(2) Copy all border pixels from $B_1$ to $B_2$. A border pixel is defined as one which is black (i.e., has a value of 1) and whose four nearest neighbors have at least one white pixel among them.

(3) Scan through $B_2$ from the upper left to the lower right corner and for each pixel check whether it is black. If so, proceed as follows:

   (3a) If any pair of adjacent nearest-neighbor (NN) and next-nearest-neighbor (NNN) pixels of the current pixel are both black, paint the NN pixel white. This is done to prevent ambiguities in the subsequent steps. In practice, however, this case is quite rare and barely changes the end result.

   (3b) Check the distribution of black pixels among the nearest and next-nearest neighbors according to Fig. A.3 and add the specified numbers to the circumference.

(4) Once the entire lattice has been scanned, divide the result by 2 to account for double counting in step (3b).
A.3.2  Algorithm to calculate the curvature diameter, $D_C$

The curvature diameter of the (combined) circumference, $L_c$, of all black areas is defined as

$$D_C := 2 \left[ \frac{1}{L_c} \int ds \left| \frac{dt}{ds} \right|^2 \right]^{-\frac{1}{2}}, \quad (A.20)$$

with $t$ the local (normalized) tangent vector. To calculate the quantity

$$\frac{1}{L_c} \int ds \left| \frac{dt}{ds} \right|^2, \quad (A.21)$$

we pursue the following strategy:

(I.1) Make a border pixel bitmap as above.

(I.2) Scan the bitmap and stop when a black pixel is found.

(I.3) Start a data array, $(x^i, y^i)$, for the coordinates (numbered $j \geq 0$) of a new line, $i \geq 0$.

(I.4) Look for a (nearest or next-nearest) neighbor of the current pixel. In most cases, there will be only one possibility to continue the line. Otherwise, an arbitrary choice is made.

(I.5) Remove the current pixel from the bitmap. Make the neighboring pixel from (4) the current pixel.

(I.6) Repeat (I.4-5) until no neighbor is found for the current pixel. Then terminate that line.

(I.7) Repeat (I.2-6) until the bitmap has been cleared.

(II.1) For each line $i$, $(x^i_j, y^i_j) \equiv (x_j, y_j)$, as determined in (I.1-7) that is longer than 3 points, set $j = 1$.

(II.2) Calculate $|dt/ds|^2$ according to

$$|\frac{dt}{ds}|^2 = \frac{dt_x^2 + dt_y^2}{ds^2}, \quad (A.22)$$

$$dt_x = \frac{x_{j+2} - x_j}{\sqrt{(x_{j+2} - x_j)^2 + (y_{j+2} - y_j)^2}} - \frac{x_{j+1} - x_{j-1}}{\sqrt{(x_{j+1} - x_{j-1})^2 + (y_{j+1} - y_{j-1})^2}} \quad (A.23)$$

$$dt_y = \frac{y_{j+2} - y_j}{\sqrt{(x_{j+2} - x_j)^2 + (y_{j+2} - y_j)^2}} - \frac{y_{j+1} - y_{j-1}}{\sqrt{(x_{j+1} - x_{j-1})^2 + (y_{j+1} - y_{j-1})^2}} \quad (A.24)$$

$$dx = \frac{x_{j+2} - x_{j+1} + x_j - x_{j-1}}{2}, \quad (A.25)$$

$$dy = \frac{y_{j+2} - y_{j+1} + y_j - y_{j-1}}{2}, \quad (A.26)$$

$$ds = \sqrt{dx^2 + dy^2}. \quad (A.27)$$
(II.3) Increase $j$. Repeat (II.2-3) until the end of the line is reached.

(II.4) Repeat (II.1-3) for all lines.

(II.5) Apply

$$L_c = \int ds.$$  \hfill (A.28)

Note that this algorithm produces a slightly different (smaller) value for $L_c$ than the one presented in the last section. This because the length of each line is effectively truncated by three points. $L_c$ here acts only as a normalizing factor and for consistency should be calculated as indicated. If interested in $L_c$ itself, one should use the algorithm of the previous section.

### A.3.3 Algorithm to calculate the direction persistence, $\Lambda$

The direction persistence parameter, $\Lambda$, was used in Chapter 4 to measure the lamellarness of a two-dimensional black-and-white image. The numerical evaluation of this parameter boils down to calculating $\langle \max(l_a) \rangle$ – the maximal length of either black or white sections along a one-dimensional cross section of the image in direction $a$, averaged over all offsets along the $x$ axis. This is done as follows.

1. Copy the image fourfold onto a larger bitmap (to account for periodic boundary conditions).
2. For all offsets in $x$ direction, convert the points on a straight line of direction $a$ in the (enlarged) image into a one-dimensional interval by means of a line-drawing algorithm and store them in an array.
3. Find the longest stretch of black or white pixels in that array.
4. Average over all offsets.
Appendix B

Program Listings

In the following, the most essential programs used in this thesis are listed. m2d.c is given in a Fortran version in Ref. [98]. In writing MFfourier.c, I drew from a program written by T. Geisinger. The other programs are my own developments.

Mean-field

- **MFfourier.c**: Fourier-space implementation of SCFT for a ternary AB+A+B blend of Gaussian chains. In the iteration leading to self-consistency, the sharp density profiles in the starting configurations are first smoothened by the use of a mixing scheme. Then I let Broyden’s multisecant method take over to converge toward the solution.

- **rodcoil.c**: SCFT for wormlike rod-coil block copolymers. The diffusion equation is solved by a FTCS algorithm (see previous appendix), and the solution is converged to with a mixing scheme.

Finite differencing schemes

- **ADI2d.c**: Crank-Nicholson/ADI in two dimensions; serial version.

- **DF2dMPI.c**: DuFort-Frankel in two dimensions; parallel version using Message Passing Interface (MPI).

Monte Carlo simulations

- **Sim2dMPI.c**: Field-Theoretic Monte Carlo simulation of $W_-$ fluctuations in two dimensions; uses the pseudo-spectral method to solve the (real) diffusion equation; parallel version using MPI.

- **FullSim2d.c**: Field-Theoretic Monte Carlo simulation of both $W_+$ and $W_-$ fluctuations in two dimensions; uses the real pseudo-spectral method to solve the complex diffusion equation (via two coupled real equations for the real and imaginary parts); single processor version.
Graphical analysis

- **U2d.c**: Calculates the circumference of microdomain boundaries in units of pixels, which is an unnormalized version of $U$ (nevertheless called $U$ in the program). In the given version, it is evaluated as the circumference around the black areas. However, this is arbitrary and might be changed to white areas. Due to the finite resolution of a lattice, the two will produce (slightly) different results, which is why it is best to take the mean of these two numbers.

- **L2d.c**: Calculates the direction persistence, $\Lambda$, as explained in Appendix A.

- **F2.c**: Calculates $\bar{F}_2$ (cf. definition (4.51)).

- **t2d.c**: Calculates the quantity $\langle 1/L_c \rangle \int ds|\frac{dt}{ds}|^2$, which is essential in evaluating the curvature diameter, $D_C$.

- **m2d.c**: Calculates the Minkowski functionals area, circumference, and Euler characteristic. Note, however, that the circumference algorithm contained herein only counts lattice segments, which was argued in Appendix A to be a bad choice. Therefore, I used **U2d.c** for the circumference.

- **sfac2d.c**: Calculates the structure factor in a simulation done with **Sim2dMPI.c** from averages of the $W_A(\equiv W_+ + W_-)$ field.
MFfourier.c

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define ANSI
#define float double
#include "dnrutil.c"
/* double precision versions of */
#include "dnrutil.h"
/* Numerical Recipes routines */
#include "dtred2.c"
#include "dtqli.c"
#include "dpythag.c"
#include "dludcmp.c"
#include "dlubksb.c"
#define PI M_PI
#define CP 1
/* copolymers : yes (0: no) */
#define f .5
/* t<f: A, t=>f: B */
#define HA 1
/* homopolymers A: yes (0: no) */
#define HB 1
/* homopolymers B: yes (0: no) */
#define nCP (D*(1.−ph))
#define nA ((D*ph)/(2.*alpha))
#define nB nA
#define EPSILON 1e−3
#define h_MIN 1e−7
#define EQU_ACCURACY 1e−20
#define MAX_NO_ITERATIONS 500
#define DW_EFFICIENCY 0.8
#define DW_REDUCTION_FACTOR 0.3
double vlam;
/* for varlam mixing scheme */
double vlam0=−.2;
double vlammin=.2;
double vlammax=.2;
int relaxerr=0.01;
double ph; /* homopolymer volume fraction */
double chiN /* double alpha; */
double fugCP;
double fugHA;
double fugHB;
int GC; /* Grand Canonical ensemble y/h */
int CONF;
int basedim;
int xdim;
double D; /* lamellar periodicity */
double *rphi_a, *rphi_b, **fourier;
double *rwa, *rwb, *O, *R, **OtI, **RtI, **OtR;
double *phi_a, *phi_b;
double *rphi_a, *rphi_b, **fourier;
double *rwa, *rwb, *O, *R, **OtI, **RtI, **OtR;
double *phi_a, *phi_b;

MFfourier.c

double *eta_a, *eta_b, *eta_cp, *eta_ha, *eta_ccp;
double **alpha_ac, **alpha_ha, **alpha_hb, **alpha_ccp;
double *exp_al, *exp_ha, *exp_hb, *exp_ccp;

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define ANSI
#define float double
#include "dnrutil.c"
/* double precision versions of */
#include "dnrutil.h"
/* Numerical Recipes routines */
#include "dtred2.c"
#include "dtqli.c"
#include "dpythag.c"
#include "dludcmp.c"
#include "dlubksb.c"
#define PI M_PI
#define CP 1
/* copolymers : yes (0: no) */
#define f .5
/* t<f: A, t=>f: B */
#define HA 1
/* homopolymers A: yes (0: no) */
#define HB 1
/* homopolymers B: yes (0: no) */
#define nCP (D*(1.−ph))
#define nA ((D*ph)/(2.*alpha))
#define nB nA
#define EPSILON 1e−3
#define h_MIN 1e−7
#define EQU_ACCURACY 1e−20
#define MAX_NO_ITERATIONS 500
#define DW_EFFICIENCY 0.8
#define DW_REDUCTION_FACTOR 0.3
double vlam;
/* for varlam mixing scheme */
double vlam0=−.2;
double vlammin=.2;
double vlammax=.2;
int relaxerr=0.01;
double ph; /* homopolymer volume fraction */
double chiN /* double alpha; */
double fugCP;
double fugHA;
double fugHB;
int GC; /* Grand Canonical ensemble y/h */
int CONF;
int basedim;
int xdim;
double D; /* lamellar periodicity */
double *rphi_a, *rphi_b, **fourier;
double *rwa, *rwb, *O, *R, **OtI, **RtI, **OtR;
double *phi_a, *phi_b;

MFfourier.c

double *eta_a, *eta_b, *eta_cp, *eta_ha, *eta_ccp;
double **alpha_ac, **alpha_ha, **alpha_hb, **alpha_ccp;
double *exp_al, *exp_ha, *exp_hb, *exp_ccp;

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define ANSI
#define float double
#include "dnrutil.c"
/* double precision versions of */
#include "dnrutil.h"
/* Numerical Recipes routines */
#include "dtred2.c"
#include "dtqli.c"
#include "dpythag.c"
#include "dludcmp.c"
#include "dlubksb.c"
#define PI M_PI
#define CP 1
/* copolymers : yes (0: no) */
#define f .5
/* t<f: A, t=>f: B */
#define HA 1
/* homopolymers A: yes (0: no) */
#define HB 1
/* homopolymers B: yes (0: no) */
#define nCP (D*(1.−ph))
#define nA ((D*ph)/(2.*alpha))
#define nB nA
#define EPSILON 1e−3
#define h_MIN 1e−7
#define EQU_ACCURACY 1e−20
#define MAX_NO_ITERATIONS 500
#define DW_EFFICIENCY 0.8
#define DW_REDUCTION_FACTOR 0.3
double vlam;
/* for varlam mixing scheme */
double vlam0=−.2;
double vlammin=.2;
double vlammax=.2;
int relaxerr=0.01;
double ph; /* homopolymer volume fraction */
double chiN /* double alpha; */
double fugCP;
double fugHA;
double fugHB;
int GC; /* Grand Canonical ensemble y/h */
int CONF;
int basedim;
int xdim;
double D; /* lamellar periodicity */
double *rphi_a, *rphi_b, **fourier;
double *rwa, *rwb, *O, *R, **OtI, **RtI, **OtR;
double *phi_a, *phi_b;
/* homogeneous setup */
if (CONF==1) {
    phi_a[1]=a; phi_b[1]=b;
    for (i=2;i<=basedim;i++) {
        phi_a[i]=0.; phi_b[i]=0.;
    }
}
/* disordered setup */
if (CONF==2) {
    phi_a[1]=.5; phi_b[1]=.5;
    for (i=2;i<=basedim;i++) {
        phi_a[i]=0.; phi_b[i]=0.;
    }
}
/* load starting configuration from file */
if (CONF==3) {
    file=fopen("MFload.cnf", "r");
    for (x=1;x<=xdim;x++) {
        fgets(line,80,file);
        sscanf(line,"%lf %lf %lf %lf %lf
", &a, &rphi_a[x], &rphi_b[x], &rwa[x], &rwb[x]);
    }
    for (i=1;i<=basedim;i++) {
        phi_a[i]=0.; phi_b[i]=0.; wa[i]=0.; wb[i]=0.;
        for (x=1;x<=xdim;x++) {
            phi_a[i]+=rphi_a[x]*fourier[i][x];
            phi_b[i]+=rphi_b[x]*fourier[i][x];
            wa[i]+=rwa[x]*fourier[i][x];
            wb[i]+=rwb[x]*fourier[i][x];
        }
    }
}

MFfourier.c

AppENDIX B. PROGRAM LISTINGS
/**************************************************************************/
/*  Initialization.                                                        */
/*                                                                          */

void init() {
    int i,j,k,x;
    double temp;
    gam=f3tensor(1,basedim,1,basedim,1,basedim);
    gam[1][1][1]=1.;
    for (k=2;k<=basedim;k++)
        for (j=2;j<=basedim;j++)
            for (i=2;i<=basedim;i++)
                if (i==k)
                    for (j=1;j<=basedim;j++)
                        if (j==k)
                            for (i=1;i<=basedim;i++)
                                lambda=dvector(1,basedim);
    for (i=1;i<=basedim;i++)
        lambda[i]=4.*PI*PI*((double)((i-1)*(i-1)));
    rphi_a=dvector(1,xdim); rphi_b=dvector(1,xdim);
    fourier=dmatrix(1,basedim,1,xdim);
    for (i=1;i<=basedim;i++)
        for (x=1;x<=xdim;x++)
            fourier[i][x]=cos(2.*PI*((double)(i-1)*(double)(x)/(double)(xdim)));
    rwa=dvector(1,xdim); rwb=dvector(1,xdim);
### MFfourier.c

```c
 OtI=dvector(1,basedim);  RtI=dvector(1,basedim);  OtR=dmatrix(1,basedim,1,basedim);  phi_a=dvector(1,basedim);  ...  xi=dvector(1,basedim);  var=dmatrix(1,3*basedim,0,1);  dvar=dmatrix(1,3*basedim,0,1);  g_new=dvector(1,2*basedim);

/*****************************************************************************/
/*  Free allocated memory.                                      */
/*****************************************************************************/

void close() {
    free_f3tensor(gam,1,basedim,1,basedim,1,basedim);  free_dvector(lambda,1,basedim);  free_dvector(rphi_a,1,xdim);  ...  free_dvector(xi,1,basedim);  free_dmatrix(rho_ha,1,basedim,1,basedim);  free_dmatrix(rho_ca,1,basedim,1,basedim);  free_dmatrix(rho_hb,1,basedim,1,basedim);  free_dmatrix(rho_cb,1,basedim,1,basedim);  free_dvector(zeta_a,1,basedim);  free_dvector(zeta_b,1,basedim);  free_dmatrix(alpha_ha,1,basedim,1,basedim);  free_dmatrix(alpha_hb,1,basedim,1,basedim);  free_dmatrix(alpha_ca,1,basedim,1,basedim);  free_dmatrix(alpha_cb,1,basedim,1,basedim);  free_dvector(exp_ha,1,basedim);  free_dvector(exp_hb,1,basedim);  free_dvector(exp_ca,1,basedim);  free_dvector(exp_cb,1,basedim);  free_dmatrix(A0,1,basedim,1,basedim);  free_dmatrix(TAO,1,basedim,1,basedim);  free_dmatrix(R1,1,basedim,1,basedim);  free_dmatrix(B1,1,basedim,1,basedim);  free_dmatrix(TB1,1,basedim,1,basedim);  free_dvector(EA1,1,basedim);  free_dvector(EA0,1,basedim);  free_dvector(EA,1,basedim);  free_dvector(E0,1,basedim);  free_dvector(E,1,basedim);  free_dvector(offdiag,1,basedim);  free_dvector(wa,1,basedim);  free_dvector(wb,1,basedim);  free_dvector(we,1,basedim);  free_dvector(dwa,1,basedim);  free_dvector(dwb,1,basedim);  free_dvector(dxi,1,basedim);  free_dmatrix(var,1,3*basedim,0,1);  free_dmatrix(dvar,1,3*basedim,0,1);  free_dvector(g_new,1,2*basedim);  }

/*****************************************************************************/
/*  Calculate the densities.                                        */
/*****************************************************************************/

void calc_dens_broyden(int k_w, double h) {
    int i,j,k,l,m,n,p;
    double factHA, factHB, factCP;
    double temp, temp1, temp2, temp3, temp4;
    double phi_ha, phi_hb, phi_ca, phi_cb;
    double exp_ha_l, exp_ca_l, exp_hb_l, exp_cb_l, EA_l, EB_l;
    if (k_w==0) {
        for (i=1;i<=basedim;i++) {
            temp=0.;  for (k=1;k<=basedim;k++)
            temp+=wa[k]*gam[i][i][k];  A0[i][i]=−lambda[i]/(D*D)−temp;  A[i][i]=A0[i][i];
        }
    }

    /* Free allocated memory.                                         */
    /* */
    void close() {
        free_f3tensor(gam,1,basedim,1,basedim,1,basedim);  free_dvector(lambda,1,basedim);  free_dvector(rphi_a,1,xdim);  ...  free_dvector(rwa,1,xdim);  free_dvector(rwb,1,xdim);  free_dmatrix(fourier,1,basedim,1,xdim);  free_dvector(otI,1,basedim);  free_dvector(rti,1,basedim);  free_dmatrix(otR,1,basedim,1,basedim);  free_dvector(phi_a,1,basedim);  free_dvector(phi_b,1,basedim);

    /* Calculate the densities.                                        */
    */

    void calc_dens_broyden(int k_w, double h) {
        int i,j,k,l,m,n,p;
        double factHA, factHB, factCP;
        double temp, temp1, temp2, temp3, temp4;
        double phi_ha, phi_hb, phi_ca, phi_cb;
        double exp_ha_l, exp_ca_l, exp_hb_l, exp_cb_l, EA_l, EB_l;
        if (k_w==0) {
            for (i=1;i<=basedim;i++) {
                temp=0.;  for (k=1;k<=basedim;k++)
                temp+=wa[k]*gam[i][i][k];  A0[i][i]=−lambda[i]/(D*D)−temp;  A[i][i]=A0[i][i];
            }
        }

        /* Free allocated memory.                                         */
        /* */
```
temp2 = exp_ca_l * f;
}
else {
    temp1 = (exp_ha_l - exp_ha[m]) / (EA_l - EA[m]);
    temp2 = (exp_ca_l - exp_ca[m]) / (EA_l - EA[m]);
}
alpha_ha[m][l] = temp1;
alpha_ha[l][m] = temp1;
alpha_ca[m][l] = temp2;
alpha_ca[l][m] = temp2;
if (EB_l == EB[m]) {
    temp1 = exp_hb_l * alpha;
    temp2 = exp_cb_l * f;
}
else {
    temp1 = (exp_hb_l - exp_hb[m]) / (EB_l - EB[m]);
    temp2 = (exp_cb_l - exp_cb[m]) / (EB_l - EB[m]);
}
alpha_hb[m][l] = temp1;
alpha_hb[l][m] = temp1;
alpha_cb[m][l] = temp2;
alpha_cb[l][m] = temp2;
for (p = 1; p <= basedim; p++) {
    temp1 += OtR[l][p] * RtI[p] * exp_cb[p];
    temp2 += OtI[p][l] * OtI[l][p] * exp_ca[p];
}
zeta_a[l] = temp1;
zeta_b[l] = temp2;
for (k = 1; k <= basedim; k++) {
    for (l = 1; l <= basedim; l++) {
        temp1 = 0.;
        temp2 = 0.;
        for (m = 1; m <= basedim; m++) {
            temp1 += O[k][m] * OtI[m][l] * exp_ha[l][m];
            temp2 += R[k][m] * RtI[l][m] * alpha_ha[l][m];
        }
        beta_ha[k][l] = temp1;
        beta_ca[k][l] = temp2;
        for (j = 1; j <= basedim; j++)
        for (k = 1; k <= basedim; k++)
            if (HA) {
                QA = 0.;
                for (i = 1; i <= basedim; i++)
                    QA += OtI[i][i] * exp_ha[i];
                QA *= D;
            }
            if (HB) {
                QB = 0.;
                for (i = 1; i <= basedim; i++)
                    QB += RtI[i][i] * exp_hb[i];
                QB *= D;
            }
            if (CP) {
                QCP = 0.;
                for (i = 1; i <= basedim; i++)
                    QCP += OtI[i][i] * zeta_a[i];
                QCP *= D;
            }
            if (GC) {
                factHA = fugHA;
                factHB = fugHB;
                factCP = fugCP;
            } else {
                if (nA) factHA = nA/QA;
                else factHA = 0.;
                if (nB) factHB = nB/QB;
                else factHB = 0.;
                if (nCP) factCP = nCP/QCP;
                else factCP = 0.;
            }
            for (i = 1; i <= basedim; i++)
                for (j = 1; j <= basedim; j++)
                    phi_ha = 0.;
                    phi_ca = 0.;
                    phi_hb = 0.;
                    phi_cb = 0.;

for (k=1;k<=basedim;k++) {
    g=gam[i][j][k];
    if (g) {
        phi_ha=phi_ha+factHA*g*rho_ha[j][k];
        phi_ca=phi_ca+factCP*g*rho_ca[j][k];
        phi_hb=phi_hb+factHB*g*rho_hb[j][k];
        phi_cb=phi_cb+factCP*g*rho_cb[j][k];
    }
}
if (i==1) {
    volHA=phi_ha;
    volHB=phi_hb;
    volCP=phi_ca+phi_cb;
    phi_a[i]=phi_ha+phi_ca;
    phi_b[i]=phi_hb+phi_cb;
}
void calc_dens_varlam() {
    int i, j, k, l, m, n, p;
    double g;
    double factHA, factHB, factCP;
    double temp, temp1, temp2, temp3, temp4;
    double phi_ha, phi_hb, phi_ca, phi_cb;
    double exp_ha_l, exp_ca_l, exp_hb_l, exp_cb_l, EA_l, EB_l;
    for (i=1;i<=basedim;i++) {
        temp=0.0;
        for (k=1;k<=basedim;k++)
            temp+=wa[k]*gam[i][i][k];
        A[i][i]=-lambda[i]/(D*D)-temp;
    }
    for (j=1;j<i;j++) {
        temp=0.0;
        for (k=1;k<=basedim;k++)
            temp-=wa[k]*gam[i][j][k];
        A[i][j]=temp;
        A[j][i]=temp;
    }
    dtred2(A,basedim,EA,offdiag);
    dtqli(EA,offdiag,basedim,A);
    O=A;
    for (i=1;i<=basedim;i++) {
        for (j=1;j<=basedim;j++)
            temp=0.0;
        for (k=1;k<=basedim;k++)
            temp+=O[k][i]*R[k][j];
        OtR[i][j]=temp;
        RtI[i][j]=R[i][j];
        exp_ha[i]=exp(EA[i]*alpha);
        exp_hb[i]=exp(EB[i]*alpha);
        exp_ca[i]=exp(EA[i]*fa);
        exp_cb[i]=exp(EB[i]*(1.0-fa));
    }
    for (l=1;l<=basedim;l++) {
        exp_ha_l=exp_ha[l];
        exp_ca_l=exp_ca[l];
        exp_hb_l=exp_hb[l];
        exp_cb_l=exp_cb[l];
        EA_l=EA[l];
        EB_l=EB[l];
        alpha_ha[l][l] = exp_ha_l * alpha;
        alpha_cb[l][l] = exp_cb_l * (1.0-fa);
        if (EB_l==EB[l]) {
            alpha_ha[l][l] = exp_ha_l * alpha;
            alpha_cb[l][l] = exp_cb_l * (1.0-fa);
        } else {
            temp1=(exp_ha_l-exp_ha[l])/EA_l;
            temp2=(exp_ca_l-exp_ca[l])/EA_l;
            alpha_ha[l][l] = temp1;
            alpha_cb[l][l] = temp2;
        }
    }
}
MFfourier.c

for (k=1;k<=basedim;k++)
    temp=wb[k]*gam[i][j][k];
    B[i][j][k]=temp;
}

dtred2(B,basedim,EB,offdiag);
dtqli(EB,offdiag,basedim,B);
R=B;
for (i=1;i<=basedim;i++) {
    for (j=1;j<=basedim;j++)
        temp=0.0;
    for (k=1;k<=basedim;k++)
        temp+=phi[k][i]*R[k][j];
    phi[i][j]=temp;
}
for (l=1;l<=basedim;l++) {
    exp_ha[l]=exp(EB[l]*alpha);
    exp_hb[l]=exp(EB[l]*alpha);
    exp_ca[l]=exp(EB[l]*fa);
    exp_cb[l]=exp(EB[l]*(1.0-fa));
}
temp2 = exp_ch_l * f;
} else {
    temp1 = (exp_hb_l - exp_hb[m]) / (EB_l - EB[m]);
    temp2 = (exp_cb_l - exp_cb[m]) / (EB_l - EB[m]);
}
alpha_hb[m][l] = temp1;
alpha_hb[l][m] = temp1;
alpha_cb[m][l] = temp2;
alpha_cb[l][m] = temp2;
    temp1 = 0.;
    temp2 = 0.;
for (p = 1; p <= basedim; p++) {
    temp1 += OtR_l[p] * RtI[p] * exp_cb[p];
    temp2 += OtR[p] * RtI[p] * exp_ca[p];
} zeta_a[l] = temp1;
zeta_b[l] = temp2;
for (k = 1; k <= basedim; k++) {
    for (l = 1; l <= basedim; l++) {
        temp1 = 0.;
        temp2 = 0.;
        temp3 = 0.;
        temp4 = 0.;
        for (m = 1; m <= basedim; m++) {
            temp1 += O[k][m] * OtI[m] * alpha_ha[l][m];
            temp2 += O[k][m] * zeta_a[m] * alpha_ca[l][m];
            temp3 += R[k][m] * RtI[m] * alpha_hb[l][m];
            temp4 += R[k][m] * zeta_b[m] * alpha_cb[l][m];
        } beta_ha[k][l] = temp1;
beta_ca[k][l] = temp2;
beta_hb[k][l] = temp3;
beta_cb[k][l] = temp4;
for (j = 1; j <= basedim; j++) {
    for (k = 1; k <= basedim; k++) {
        g = gam[i][j][k];
        if (g) {
            phi_ha += factHA * g * rho_ha[i][j][k];
            phi_ca += factCP * g * rho_ca[i][j][k];
            phi_hb += factHB * g * rho_hb[i][j][k];
            phi_cb += factCP * g * rho_cb[i][j][k];
        }
    }
    if (i == 1) {
        volHA = phi_ha;
        volHB = phi_hb;
        volCP = phi_ca + phi_cb;
    }
}
}
double free_energy()
{
    int i;
    double a,b;
    a=0.;
    for (i=1;i<=basedim;i++)
        a+=D*(chin*phi_a[i]*phi_b[i]-wa[i]*phi_a[i]-wb[i]*phi_b[i]);
    if (GC)
        if (fugCP) a-=fugCP*QCP;
        if (fugHA) a-=fugHA*QA;
        if (fugHB) a-=fugHB*QB;
    else
        if (nCP) a-=nCP*log(QCP/nCP);
        if (nA) a-=nA*log(QA/nA);
        if (nB) a-=nB*log(QB/nB);
    return(a/D);
}

void do_nothing(double temp)
{
    return;
}

void calc_g(double *g)
{
    int i,j;
    double h, w_old, *w, wa_old;
    for (k=1;k<dim_eq;k++)
        if (k<=basedim) w=wa;
        else w=wb-basedim;
        w_old=w[k];
        h=EPSILON*w_old;
        if (h<h_MIN) h=h_MIN;
        w[k]+=h;
        do_nothing(w[k]);
        do_nothing(w_old);
        h=w[k]-w_old;
        calc_dens_broyden(k,h);
        calc_g(g_new);
        w[k]=w_old;
    for (j=1;j<dim_eq;j++)
        Jacobian[j][k]=(g_new[j]-g[j])/h;
}

void solve_for_dw(double **jacobian, double *g, double *dw, int dim_eq)
{
    int i,j;
    double h, w_old, *w, wa_old;
    for (k=1;k<dim_eq;k++)
        if (k<=basedim) w=wa;
        else w=wb-basedim;
        w_old=w[k];
        h=EPSILON*w_old;
        if (h<h_MIN) h=h_MIN;
        w[k]+=h;
        do_nothing(w[k]);
        do_nothing(w_old);
        h=w[k]-w_old;
        calc_dens_broyden(k,h);
        calc_g(g_new);
        w[k]=w_old;
    for (j=1;j<dim_eq;j++)
        Jacobian[j][k]=(g_new[j]-g[j])/h;
}
```c
int i, j, k;
double sign_perm;
double **x;
int *perm;
x = dmatrix(1, dim_eq, 1, dim_eq);
perm = ivector(1, dim_eq);
for (i=1; i<=dim_eq; i++)
    for (k=1; k<=dim_eq; k++)
        if (i==k)
            x[i][k] = 1.
        else
            x[i][k] = 0;
dludcmp(jacobian, dim_eq, perm, &sign_perm);
dlubksb(jacobian, dim_eq, perm, x);
}
free_dmatrix(x, 1, dim_eq, 1, dim_eq);
free_ivector(perm, 1, dim_eq);
}

void broyden(int conf) {
    int iter, i, j, k, l, x;
    double *g, *g_old, *g_dum, temp;
    double *dg, *eta, *xi;
    double **jacobian, **M, **M_old, **M_dum;
    M = dmatrix(1, dim_eq, 1, dim_eq);
    M_old = M;
    M_dum = M;
    for (i=1; i<=dim_eq; i++)
        for (k=1; k<=dim_eq; k++)
            jacobian[i][k] = M_old[i][k]*dg[k];
}
```
for [i=1;i<dim_eq;i++ ] {
    dw[0]=0;
    for [k=1;k<dim_eq;k++ ]
        dw[0]−=M[i][k]*g[k];
}

M_dum = M_old; M_old = M; M = M_dum;    g_dum = g_old; g_old = g; g = g_dum; }

while (0==0);

free_dvector(g,1,dim_eq);  free_dvector(g_old,1,dim_eq);  free_dvector(dg,1,dim_eq);  free_dvector(dw,1,dim_eq);  ...
free_dvector(xi,1,dim_eq);  free_dmatrix(jacobian,1,dim_eq,1,dim_eq);  free_dmatrix(M_alloc,1,dim_eq,1,dim_eq);}

void iterate() {
    int iter,i,j,k,l,x;
    double temp1,temp2,temp3;
    double lam, test, oldf, newf;
    double fdwa,fdwb,phi;

    l=1;
    for [i=1;i<basedim;i++ ] {
        xi[i]=0.;
        dwa[i]=chin*phi_b[i];
        wa[i]=dwa[i]+xi[i];
        dwb[i]=chin*phi_a[i];
        ...
        var[l++][0]=dwb[i];    var[l++][0]=xi[i];  }

    iter=1;  calc_dens_varlam();  newf=free_energy();
    do 
    
    l=1;
    for [i=1;i<basedim;i++ ] {
        var[l++][0]=dvar[i][0];
        var[0][0] += vlam*dvar[i][0];
    }

    for [i=1;i<basedim;i++ ] {
        phi_a[i]=phi_b[i];
        fdwa=chin*phi_b[i];
        fdwb=chin*phi_a[i];
        dvar[l++][0]=fdwa−dwa[i];      dvar[l++][0]=fdwb−dwb[i];
        if (i==1) dvar[l++][0]=chin*(phi−1.)/2.;
        else dvar[l++][0]=chin*phi/2.;
    }

    varlam(iter,3*basedim);
    test=0.;  l=1;
    for [i=1;i<3*basedim;i++ ]
    {
        temp2=var[l++][0];
        temp2=var[l++][0];
        temp3=var[l++][0];
    }
\[
test += ((temp1 - dwa[i])^2 + (temp2 - dwb[i])^2 + (temp3 - xi[i])^2) / (temp1 - dwa[i]) / (temp2 - dwb[i]) / (temp3 - xi[i]));
\]

dwa[i] = temp1;
dwb[i] = temp2;
xi[i] = temp3;
wa[i] = dwa[i] + xi[i];
wb[i] = dwb[i] + xi[i];
}
test = sqrt(test);
iter++;
calc_dens_varlam();
oldf = newf;
newf = free_energy();

while ((iter <= relaxnr) && (test > relaxerr));
}

void make_real()
{
    int i, x;
    for (x = 1; x <= xdim; x++)
    {
        rphi_a[x] = 0.;
        rphi_b[x] = 0.;
        rwa[x] = 0.;
        rwb[x] = 0.;
        for (i = 1; i <= basedim; i++)
        {
            rphi_a[x] += phi_a[i] * fourier[i][x];
            rphi_b[x] += phi_b[i] * fourier[i][x];
            rwa[x] += wa[i] * fourier[i][x];
            rwb[x] += wb[i] * fourier[i][x];
        }
    }
}

void printconf()
{
    int x, a;
    printf("\n");
    for (x = 1; x <= xdim; x++)
    {
        a = rint(rphi_a[x] * 10.);
        printf("%d
", a);
    }
    for (x = 1; x <= xdim; x++)
    {
        if (rphi_a[x] < 0.42) printf("0.16%");
        else if (rphi_a[x] < 0.58) printf("0.18%");
        else printf("0.17%");
    }
    printf("free energy/D: %f
volCP: %f  volHA: %f
\n",
free_energy(), volCP, volHA, volHB);
}

double SCFT(int conf, int write)
{
    setconf(conf);
    iterate();
}

MFfourier.c

MFfourier.c

MFfourier.c
broyden(conf);
make_real();
print_conf();
if (write) write_conf();
return free_energy();
}

/***************************************************************************
*  Sweep through lamellar periodicities, D.                                *
*                                                                          */

void scanxsize(int mode, double Dstart, double Dstep, int imax) {
    int i;
    FILE *file;
    double F;
    file=fopen("MFout","w");
    D=Dstart;
    for (i=0;i<imax;i++) {
        F=SCFT(mode,1);
        printf("%f %.10lf
",D,F);
        fprintf(file,"%f %.10lf
",D,F);
        fflush(file);
        D+=Dstep;
    }
    fclose(file);
}

/***************************************************************************
*  Main routine. Insert parameters here and recompile.                     *
*                                                                          */

int main() {
    double z;
    basedim=30;
    xdim=40;
    alpha=.2;
    ini();
    CONF=0;
    chin=12.5;
    fugCP=0.;
    fugHA=0.;
    fugHB=0.;
    GC=0;  ph=.7;
    setup dens=1.;
    scanxsize(0,4.53,.0,1);
    close();
    return(0);
}
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define ANSI
#define float double
#include "nrutil.h" /* Numerical Recipes */
#include "nrutil.c"

double tsize = 1.;
int relaxnr=20000;
double relaxerr=.001;
double vlam;
double vlam00;
double vlammin;
double vlammax;
double G;
int LOAD=0; /* 1/0: load starting configuration yes/no */
char READNAME[]="readfile.dat";
double zsize;
int zmax;
int tg;
int tmax;
double fract;
double xi_rigid;
double xi_flex;
int lmax;
double V, Q, f1;
double dz, dt;
int *zp, *zm;
double **var,**dvar, *dw0;
double **f, **wf, **dwf, **wg, **dwg, **g, *xi;
double ***q, ***qc;
double ***cg, *d, *xi, *xic;
int *wfact, *wfactc;
double **fact1, *fact2, ****fact3, *fact4;
double PI=M_PI;
int iter;

double fact(int x) {
    int i;
    double facts;
    facts=1.;
    if (x==0) return 1.;
    
    for(i=1; i< x; i++)
        facts *= (double)i;
    return facts;
}

double twotothe(int n) {
    int i;
    double a;
    a=2.;
    for (i=2;i<n;i++)
        a*=2.;
    return a;
}

int iabs(int i) {
    if (i<0) return -(−i);
    else return i;
}

void setconf(double aaa) {
    int 1,x,y,z,1,n,a,b;
    FILE *file;
    char fname[20];
    char line[80];
    double d=0.;
    double d1=0.;
    double d2=0.;
    double d3=0.;
    double d4=0.;
    double d5=0.;
    if (LOAD) {
        for (z=1;z<=zmax;z++)
            for (l=0;l<=lmax;l++) {
                f[l][z]=0.;
                g[l][z]=0.;
            }
        file=fopen(READNAME,"r");
        for (z=1;z<=zmax;z++) {
            fgets(line,80,file);
            printf("line %s ",line);
            sscanf(line,"%d %lf %lf %lf %lf %lf %lf ",&a,&d1,&d2,&d3,&d4,&d5,&d6);
            f[0][z]=d1/sqrt(4.*PI);
        }
    }
}

/* Set up starting configuration. */
/* */
/* */

else {
    for(i=1; i< x; i++)
        facts *= (double)i;
    return facts;
} }
```c
double cgc(int j1, int j2, int j3, int m1, int m2, int m3) {
    int a1, a2, a3, a4, a5;
    double a, r, d;
    if (m1+m2==m3) c=0.;
    else {
        c = sqrt(fact(j1+j2-j3))*fact(j2+j3-j1)/fact(j1+j2+j3+1)) * sqrt((2.*j1+1.)/(2.*j2+1.))*sqrt((2.*j3+1.)/(4.*PI));
        b=0.;
        for (i=0;i<j+1;i++) {
            a1=i-m1-i;
            a2=j3-m2-i;
            a3=j2-m2-i;
            a4=j1-m2-i;
            a5=j-j2-j3-i;
            if (a1<0||a2<0||a3<0||a4<0||a5<0) d=0.;
        }
    }
    return c;
}
```
# rodcoil.c

```c
void init() {
    /* Initialize: */
    int l, m1, m2, m3, j1, j2, j3;
    double a;
    f = dmatrix(0, lmax, 1, zmax);  
    g = dmatrix(0, lmax, 1, zmax);
    wf = dmatrix(0, lmax, 1, zmax);
    wfg = dmatrix(0, lmax, 1, zmax);
    xi = dvector(1, zmax);
    d = dvector(0, lmax);
    for (l=0; l <= lmax; l++) {
        d[l] = 0.;
    }
    q = f3tensor(0, lmax, 1, zmax, 0, tmax);  
    qc = f3tensor(0, lmax, 1, zmax, 0, tmax);
    for (z=1; z <= zmax; z++) {
        for (i=1; i <= lmax; i++) {
            q[i][z][0] = 0.;
            qc[i][z][0] = 0.;
        }
        q[0][z][0] = sqrt(4.0*PI);  
        qc[0][z][0] = sqrt(4.0*PI);
    }
    xi = dvector(0, tmax);
    xic = dvector(0, tmax);
    wfact = ivector(0, tmax);
    wfactc = ivector(0, tmax);
    for (i=0; i <= tmax; i++) {
        xi[i] = xi_rigid;
        wfact[i] = 1;
    }
    if (tmax==1) {
        zp[1] = 1;
        zm[1] = 1;
    } else {
        for (z=1; z <= zmax-1; z++)
            zp[z] = z+1;
        zp[zmax] = 1;
        for (z=2; z <= zmax; z++)
            zm[z] = z-1;
    }
    free_dmatrix(f, 0, lmax, 1, zmax);
    free_dmatrix(g, 0, lmax, 1, zmax);
    free_dmatrix(wf, 0, lmax, 1, zmax);
    free_dmatrix(wfg, 0, lmax, 1, zmax);
}
```
Appendix B. Program Listings

```c
void calc_q(int t0, int t1, double ***q, double *xi, double **dw)
{
    int i, x, y, z, l, t, la, lb, lbmin, lbmax, ma, mb, lupper;
    double a, b, br, bi, cr, ci, crx, cix, cry, ciy, dr, di, s, s1, s2;
    double ftemp, gtemp;
    Q=0.;
    for (z=1; z<=zmax; z++) {
        for (l=0; l<=lmax; l++) {
            a=0.;
            for (t=0; t<tmax; t++)
                a+=q[l][z][t]*qc[l][z][tmax-t];
            if ((l%2)!=0) a=-a;
            Q+=a;
        }
        Q*=dz*dt;
    }
}
```

```c
void calc_dens() {
    int i, x, y, z, l, t, la, lb, lbmin, lbmax, ma, mb;
    double a, br, bi, cr, ci, crx, cix, cry, ciy, dr, di, s, s1, s2;
    double ftemp, gtemp;
    Q=0.;
    for (z=1; z<=zmax; z++) {
        for (l=0; l<=lmax; l++) {
            a=0.;
            for (t=0; t<tmax; t++)
                a+=q[l][z][t]*qc[l][z][tmax-t];
            if ((l%2)!=0) a=-a;
            Q+=a;
        }
        Q*=dz*dt;
    }
    Q=0.;
    for (z=1; z<=zmax; z++) {
        for (l=0; l<=lmax; l++) {
            a=0.;
            for (t=0; t<tmax; t++)
                a+=q[l][z][t]*qc[l][z][tmax-t];
            if ((l%2)!=0) a=-a;
            Q+=a;
        }
        Q*=dz*dt;
    }
    for (z=1; z<=zmax; z++) {
        for (l=0; l<=lmax; l++)
            Q+=q[l][z][tmax];
    }
    Q*=dz*dt;
    for (z=1; z<=zmax; z++)
        Q+=qc[l][z][tmax];
    Q*=dz*dt;
}
```
```c
double fact3[10][10][10];

for (la=0; la<lmax; la++) {
    lbmin=la-l;
    if (lbmin<0) lbmin=-lbmin;
    lbmax=la+l;
    if (lbmax>lmax) lbmax=lmax;
    for (lb=lbmin; lb<=lbmax; lb++) {
        dr=0.;
        a=−fact3[lab][la][l];
        if ((la%2)!=0) a=−a;
        for (t=0; t<tg; t++) {
            dr+=qc[lab][z][tmax−t]*q[lb][z][t];
        }
        ftemp+=dr*a;
    }
}

f[l][z]=ftemp*dt*V/Q;
g[l][z]=gtemp*dt*V/Q;
}

void varlam(int iter, int nvar) {
    double sum1, sum2;
    int nv;
    if (iter>1) {
        sum1=0.;
        sum2=0.;
        for (nv=0; nv<nvar; nv++) {
            sum1 += (var[nv][0]−var[nv][1])*(var[nv][0]−var[nv][1]);
            sum2 += (dvar[nv][1]−dvar[nv][0])*(dvar[nv][1]−dvar[nv][0]);
        }
        if (sum2>(1./double(nvar))) {
            vlam = sqrt(sum1/sum2);
            if (vlam<vlammin) vlam=vlammin;
            if (vlam>vlammax) vlam=vlammax;
        } else
            vlam=vlammin;
    } else
        vlam=vlam00;
    for (nv=0; nv<nvar; nv++) {
        var[nv][1] = var[nv][0];
        dvar[nv][1] = dvar[nv][0];
        var[nv][0] += vlam*dvar[nv][0];
    }
}

void printconf() {
    int z,i,j,l,m,t;
    double a,b,c;
    a=sqrt(4.*PI);
    for (z=1; z<=zmax; z++)
        printf("%d %f %f %f %f %f %f %f %f
",z,a*f[0][z],a*g[0][z],f[1][z],g[1][z],f[2][z],g[2][z],w[0][z],w[0][z]);
    a=1.;
    for (l=0; l<1+lmax; l++)
        printf("l: %d f: %f g: %f wf: %f wg: %f
",l,a*f[l][1],a*g[l][1],wf[l][1],wg[l][1]);
}
```
```c
void init_fields()
{
    int x,y,z,l,m;
    for (z=1;z<=zmax;z++)
        for (l=0;l<=lmax;l++)
            if ((l%2)==0) {
                dwf[l][z]=fact4[l]*(f[l][z]+g[l][z]);
                dwg[l][z]=fact4[l]*(f[l][z]+g[l][z]);
            }
            else {
                dwf[l][z]=0.;
                dwg[l][z]=0.;
            }
    wf[l][z]=dwf[l][z];
    wg[l][z]=dwg[l][z];
}

void relax()
{
    int i,x,y,z,l,m;
    double temp1, temp2, temp3, temp4, temp5, test;
    double fmu, fwf, fwg, c;
    i=0;  c=0.;
    for (z=1;z<=zmax;z++)
        for (l=0;l<=lmax;l++)
            var[i++][0]=dwf[l][z];
    iter=1;    propag();  calc_dens();
    do
    { /* Determine new fields */
        i=0;
        temp1=var[i++][0];  temp2=temp1;  temp3=dwf[l][z];  temp4=dwg[l][z];    test += (temp1-temp3)*(temp1-temp3) + (temp2-temp4)*(temp2-temp4);
        dwf[l][z]=temp1;
        dwg[l][z]=temp2;
    } while ((iter<=relaxnr)&&(test>relaxerr));
}

int main()
{
    /* rod section for smectic setup */
    return 0;
}
```

```c
double *e, *r, *b, *u;
double **wm, **wp;
double d0, d1, ***d;
int nmax, ymax;
double axc, *bx, cxc, *rx, *ux;
double ayc, *by, cyc, *ry, *uy;
double **v1, dx, dy, d;

G=29.;
size=1.3;  /* box size in units of contour length */
max=65;    /* box size in number of pixels */
init();
setconf(.3);
init2();
init_fields();
relax();

for (i=2;i<n−1;i++) {
    temp=−a/b[i−1];        b[i]+=temp*c;    r[i]+=temp*r[i−1];    e[i]=temp*e[i−1];  
}  e[n−1]+=c;  temp=−a/b[n−1];  rn=r[n]+temp*r[n−1];  en=b[n]+temp*e[n−1];  
for (i=1;i<n−2;i−−)
    u[i]=(r[i]−e[i]*un−c*u[i+1])/b[i];

/* Invert triadiagonal-plus-opposite-corners matrix */
/* (periodic boundary conditions) */
void periodc(double a, double *b, double c, double *r, double *u, int n)
{
    int i;
    double d,
    d0, d1, ***d;
    int nmax, ymax;
    double axc, *bx, cxc, *rx, *ux;
    double ayc, *by, cyc, *ry, *uy;
    double **v1, dx, dy, d;

    G=29.;
    size=1.3;  /* box size in units of contour length */
    max=65;    /* box size in number of pixels */
    init();
    setconf(.3);
    init2();
    init_fields();
    relax();

    for (i=2;i<n−1;i++) {
        temp=−a/b[i−1];        b[i]+=temp*c;    r[i]+=temp*r[i−1];    e[i]=temp*e[i−1];  
    }  e[n−1]+=c;  temp=−a/b[n−1];  rn=r[n]+temp*r[n−1];  en=b[n]+temp*e[n−1];  
    for (i=1;i<n−2;i−−)
        u[i]=(r[i]−e[i]*un−c*u[i+1])/b[i];
}
```

```c
/* Evaluate diffusion eqn. with CN scheme to find propagator */
/* from t0 to t1−1, using field w, and store in q[x][y][t]. */
int calc_q(double ***q, int w, int t0, int t1) {
    int i,x,y,lt;
    double alphax, alphay;
    for (i=1;i<xmax;i++)
        for (y=1;y<ymax;y++)
```
```c
#include "mpi.h"

double **w, **v0, **d1, **d2, **d3;
double c1, c2, c3, c4, *e, *g;
int xmax, ymax;
int xmnumber; /* left neighbor lattice in geometric decomposition */
int xpnumber; /* right neighbor lattice */
int ymnumber; /* lower neighbor lattice */
int ypnumber; /* upper neighbor lattice */

#include "mpi.h"

double **v, **v2, ***d, ***d2, ***d3;
double c1, c2, c3, c4, *e, *g;
int xmax, ymax;
int xmnumber; /* left neighbor lattice in geometric decomposition */
int xpnumber; /* right neighbor lattice */
int ymnumber; /* lower neighbor lattice */
int ypnumber; /* upper neighbor lattice */

#include "mpi.h"

double **v, **v2, ***d, ***d2, ***d3;
double c1, c2, c3, c4, *e, *g;
int xmax, ymax;
int xmnumber; /* left neighbor lattice in geometric decomposition */
int xpnumber; /* right neighbor lattice */
int ymnumber; /* lower neighbor lattice */
int ypnumber; /* upper neighbor lattice */

#include "mpi.h"

double **v, **v2, ***d, ***d2, ***d3;
double c1, c2, c3, c4, *e, *g;
int xmax, ymax;
int xmnumber; /* left neighbor lattice in geometric decomposition */
int xpnumber; /* right neighbor lattice */
int ymnumber; /* lower neighbor lattice */
int ypnumber; /* upper neighbor lattice */

#include "mpi.h"

double **v, **v2, ***d, ***d2, ***d3;
double c1, c2, c3, c4, *e, *g;
int xmax, ymax;
int xmnumber; /* left neighbor lattice in geometric decomposition */
int xpnumber; /* right neighbor lattice */
int ymnumber; /* lower neighbor lattice */
int ypnumber; /* upper neighbor lattice */
```
j=0;
if (t0==1) {
  prestep=16.;
  for (i=xmax;x<xmax++)
    for (j=ymax;y<ymax++)
      v[x][y]=1.-d[x][y][w]/prestep;
  for (i=0;i<4;i++) {
    c1=2.*dt/(dx*dx*prestep);
    c2=1.-c1;
    c3=1.+c4;
    exchange(j++);
    for (i=xmax;i<xmax++)
      for (j=ymax;j<ymax++)
        v2[x][y]=((c2−d[x][y][w]/prestep)
          +c1*(v[x+1][y]+v[x−1][y]
            +v[x][y+1]+v[x][y−1]))/(c3+d[x][y][w]/prestep);
    vdum=v;v=v2;v2=vdum;
    prestep/=2.;
  }
  for (i=xmax;i<xmax++)
    for (j=ymax;j<ymax++)
      q[x][y][1]=v[x][y];
  t0++;
}
for (it=t0-1;it<t1-1;it++) {
  exchange(j++);
  for (i=xmax;i<xmax++)
    for (j=ymax;j<ymax++)
      a=d2[x][y][w]*q[x][y][it-1] +
        d3[x][y][w]*(v[x+1][y]+v[x-1][y]
          +v[x][y+1]+v[x][y-1]);
      q[x][y][it+1]=a;
      v2[x][y]=a;
    vdum=v;v=v2;v2=vdum;
  return(0);
}
# include<stdio.h>
# include<stdlib.h>
# include<math.h>
# include<rfftw_mpi.h>
# include"mpi.h"
#define ANSI
#define tsize 1.0
/* must be 1. */
/* From Numerical Recipes (double version) */
#include"dnrutil.c"
#include<time.h>
#include<sys/types.h>
time_t start_time; timeval; rfftwnd_mpi_plan plan, iplan;
int numprocs, me;
int local_nx, local_x_start, local_ny_after_transpose;
int local_y_start_after_transpose, total_local_size;
int nx; int ny;
#define nstmax
#define ds dt
double *exp_lapl_factor;
double *exp_wh_factor;
double *data; /* for FFT */
double *work;
double ph; /* vol% homopolymer */
int CP; /* copolymers : yes/no (1/0) */
double ef; /* eff: A, eff: B */
int HA; /* homopolymers A: yes/no (1/0) */
int HB; /* homopolymers B: yes/no (1/0) */
double nCP; /* rel. # of monomers in copolymer -- must be 1.!! */
double nA; /* rel. # of monomers in A homopol. */
double nB; /* rel. # of monomers in B homopol. */
/* Monte Carlo Loop */
int MCiter; int relaxerrFL; /* (max.) # of iterations (relaxation) */
double relaxerrstartFL; /* error margin for relaxations */
double relaxerr; double relxerrFL; double vlam;
/* for varlam mixing scheme */
double vlam0;
double vlammin; double vlammax;
int succmove=0;
int restarts=0;
int newwstep; int savestep;
double accrate; double accgoal;
double accerr; int accnum;
int countacc=0;
double MXstep;
double oldMCstep;
double chi4;
double zCP;
double zA;
double zB;
int GC; /* 0: canonical ensemble, 1: grand canonical ensemble */
double xsize; double ysize;
double C;
int xmax; int xmin;
int ymax; int ymin;
int tmax;
int diffcount;
double diffrate;
double avtest;
double avferr;
int mandim;
int tmax,tmaxmax;
int run;
char NAME[40];
char READNAME[40];
int LOAD;
int basedim;
double V, dx, dy, dt;
double *dwp, *dwm;
double QCP, QA, QB, h1, oldh1;
double volCP, volHA, volHB;
double hinit;
double **var,**dvar;
Sim2dMPI.c

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <math.h>
#include <mpi.h>

/* 'ranmar' random number generator */
double *u, c, cd, cm;
int i97, j97, *im_ran;
double **v, **v2, *e, *g;
MPI_Status status;
MPI_Comm my_comm, mpi_comm_world;
FILE *afile,*ffile,*ifile,*cfile, *bfile;
char aname[40], fname[40], iname[40], cname[40], bname[40];
FILE *sfile;
char sname[40];
int i_start;

/*********************/
/* Initializing routine for the 'ranmar' random number generator. */
/*********************/
void rmarin(int ijk) {
  int i,j,k,l,m;
double s,t;
  int ii,jj,ij,kl;
  ijk = ijk/30082;
  kl = ijk−(ijk*30082);
  i = ((ij/177) % 177)+2;
  j = (ij % 177)+2;
  k = ((kl/169) % 178)+1;
  l = kl % 169;
  for (ii=1;ii<=97;ii++) {
    s = 0.;
    t = 0.5;
    for (jj=1;jj<=24;jj++) {
      m = (((i*j)%179)*k) % 179;
      i = j;
      j = k;
      k = m;
      l = (53*l+1) % 169;
      if (((l*m)%64)>=32) s+=t;
      t/=2;
    }
    u[ii]=s;
  }  
c = 362436. / 16777216.;
d = 7654321. / 16777216.;
return (c)

double rndnr() {
  int i;
  double uni;
  uni = u[i97]−u[j97];
  if (uni<0) uni+=1.;
  u[i97] = uni;
  i97 = im_ran[i97];
  j97 = im_ran[j97];
  c−=cd;
  if (c<0.) c+=cm;
  uni−=c;
  if (uni<0.) uni+=1.;
  return (uni);
}

double randomnr() {
  int i;
  double a,b;
  for (i=0;i<numprocs/2;i++) {
    a=rndnr();
    if ((i==me)||(i==me−numprocs/2)) b=a;
  }
  return (b);
}

/*****************************/
/* Initialize. */
/*****************************/
void init() {
  int i,j,k,k1,k2;
  double da, db;
  da=(xsize/ox);
  db=(ysize/oy);
  dx=(size/cm);
  dy=(size/ny);
  dl=(size/cm);
  da=4*M_PI*M_PI/(ox*oy*dx*dy);ds;
  data = dvector(0,total_local_size−1);
  work = dvector(0,total_local_size−1);
  exp_wh_factor=dvector(0,local_nx*ny−1);
  exp_lapl_factor=dvector(0,2*nx*local_ny_after_transpose+1−1);
a = exp_lapl_factor;

for (j=local_y_start_after_transpose; j<local_y_start_after_transpose+local_ny_after_transpose; j++) {
    for (i=0;i<nx/2;i++) {
        k1 = i;
        k2 = j;
        db = exp(-(k1*k1 + k2*k2)*da);
        *(a++) = db; // Re
        *(a++) = db; // Im
    }
    for (i=nx/2;i<nx;i++) {
        k1 = nx - i;
        k2 = j;
        db = exp(-(k1*k1 + k2*k2)*da);
        *(a++) = db; // Re
        *(a++) = db; // Im
    }
}

phiA = dmatrix(1,xmax,1,ymax);
phiB = dmatrix(1,xmax,1,ymax);
phiAH = dmatrix(1,xmax,1,ymax);
phiBH = dmatrix(1,xmax,1,ymax);

v = dmatrix(0,xmax+1,0,ymax+1);
v2 = dmatrix(0,xmax+1,0,ymax+1);

for (x=1;x<=xmax;x++)
    for (y=1;y<=ymax;y++) {
        Sim2dMPI.c
        q[x][y][0]=1.;
        qc[x][y][0]=1.;
        if (HA) qA[x][y][0]=1.;
        if (HB) qB[x][y][0]=1.;
    }

/* Free allocated memory. */
void close() {
    free_dvector(data,0,total_local_size−1);
    free_dvector(work,0,total_local_size−1);
    free_dmatrix(v,0,xmax+1,0,ymax+1);
    free_dmatrix(v2,0,xmax+1,0,ymax+1);
    if (CP) {
        free_f3tensor(q,1,xmax,1,ymax,0,tmax);
        free_f3tensor(qc,1,xmax,1,ymax,0,tmax);
        free_f3tensor(qA,1,xmax,1,ymax,0,tamax);
        free_f3tensor(qB,1,xmax,1,ymax,0,tbmax);
    }
    free_f3tensor(d,1,xmax,1,ymax,0,1);
    free_f3tensor(d2,1,xmax,1,ymax,0,1);
    free_f3tensor(d3,1,xmax,1,ymax,0,1);
    free_dvector(e,1,6*maxdim);
    free_dvector(g,1,6*maxdim);

    if (xmax>=ymax) maxdim=xmax;
    else
        maxdim=ymax;
    if (maxdim<6) maxdim=6;
    e=dvector(1,6*maxdim);
    g=dvector(1,6*maxdim);
    u=dvector(1,97);
    im_ran=ivector(1,97);
    rmarin(54217137);
    for (x=1;x<=xmax;x++)
        for (y=1;y<=ymax;y++) { ...:}
free_dvector(u,1,97);
free_ivector(im_ran,1,97);
free_dmatrix(savewp,1,xmax,1,ymax);
free_dmatrix(savewphi,1,xmax,1,ymax);
free_dmatrix(savewphii,1,xmax,1,ymax);
free_dmatrix(savewphi,1,xmax,1,ymax);
free_dvector(dem,1,basedim);
free_dmatrix(dm,1,xmax,1,ymax);
free_dmatrix(xl,1,xmax,1,ymax);
free_dmatrix(dm,1,xmax,1,ymax);
free_dmatrix(wp,1,xmax,1,ymax);
free_dmatrix(var,1,(2*xmax*ymax),0,1);
free_dmatrix(dvar,1,(2*xmax*ymax),0,1);
}

/****************************************************************************/
/*  Evaluate diffusion eqn. with pseudo-spectral method */
/*  in q[x][y][t].                  */
int calc_q(double ***q, int w, int s0, int s1) {
  int i,j,s;
  double *a, *b, *c;
  int nxny=local_nx*ny;
  int nxny2=2*nx*local_ny_after_transpose;
  double ds_half=ds/2.0;
  b=exp_wh_factor;  c=&d[1][1][w];
  for (i=0; i<nxny; i++) {
    *(b++) = exp(− *c);     c+=2;
  }
  for (s=s0−1; s<s1; s++) {
    a=data;    b=exp_wh_factor;    c=&q[1][1][s];
    for (i=0; i<local_nx; i++) {
      for (j=0; j<ny; j++) {
        *(a++) = *(b++) * *(c++) / (double)(nx*ny);
      }
      a+=2;
    }
    rfftwnd_mpi(plan,1,data,work,FFTW_TRANSPOSED_ORDER);    a=data;    b=exp_lapl_factor;
  }
  for (s=s0−1; s<s1; s++) {
    a=data;    b=exp_wh_factor;    c=&q[1][1][s+1];
    for (i=0; i<local_nx; i++) {
      for (j=0; j<ny; j++) {
        *c = *(b++) * *(a++) / (double)(nx*ny);
      }
      a+=2;
    }
    rfftwnd_mpi(iplan,1,data,work,FFTW_TRANSPOSED_ORDER);     a=data;    b=exp_lapl_factor;
  }

/****************************************************************************/
/*  Define d[x][y][w] (used in calc_q()).                                   */
void init_d() {
  int i,j,s;
  double d0,d1;
  for (x=1;x<=local_nx;x++)
    for (y=1;y<=ny;y++) {
      d[x][y][0]=(wp[x][y]+wm[x][y])*dt/2.;  /* WA*dt/2. */
      d[x][y][1]=(wp[x][y]−wm[x][y])*dt/2.;  /* WB*dt/2. */
    }
}

/****************************************************************************/
/*  Calculate the complete propagator ... from t1 to t2−1.                     */
int init_q(double ***q, int wa, int wb, int t0, int t1, int t2) {
  if (calc_q(q,wa,t0,t1))
    return (1);
  if (calc_q(q,wb,t1+1,t2))
    return (1);
  return (0);
}

/****************************************************************************/
/*  Calculate propagators of all polymer species.                            */
int propag() {
  for(i=0; i<nnxy2; i++)
    *(a++) = *(b++);
  rfftwnd_mpi(plan,1,data,work,FFTW_TRANSPOSED_ORDER);
  a=data;
  b=exp_wh_factor;
  for(i=0; i<nnxy2; i++)
    *(a++) = *(b++);
  rfftwnd_mpi(plan,1,data,work,FFTW_TRANSPOSED_ORDER);
  a=data;
  b=exp_wh_factor;
  for(i=0; i<nnxy2; i++)
    *(a++) = *(b++);
  rfftwnd_mpi(plan,1,data,work,FFTW_TRANSPOSED_ORDER);
  a=data;
  b=exp_wh_factor;
  for(i=0; i<nnxy2; i++)
    *(a++) = *(b++);
  rfftwnd_mpi(plan,1,data,work,FFTW_TRANSPOSED_ORDER);
  a=data;
  b=exp_wh_factor;
  for(i=0; i<nnxy2; i++)
    *(a++) = *(b++);
  rfftwnd_mpi(plan,1,data,work,FFTW_TRANSPOSED_ORDER);
  }
```c
int x, it;
double d;

diffcount++;    init_d();

if (CP) {
    if (me<numprocs/2) {
        if (init_q(q,0,1,1,(l-f)*tmax),tmax) return(1); /* (q,w0,w1,...) */
    } else if (init_q(qc,1,0,1,(1-f)*tmax),tmax) return(1); /* (qc,w1,w0,...)*/
}

if (me<numprocs/2) {
    if ((HA)&&(calc_q(qA,0,1,tamax))) return(1);
}

if (me<numprocs/2) {
    if ((HA)&&(calc_q(qA,0,1,tamax))) return(1);
}

if (me<numprocs/2) {
    if (init_q(q,0,1,1,((1−f)*tmax),tmax)) return(1);
}

if (me<numprocs/2) {
    if (init_q(qc,1,0,1,(l-f)*tmax),tmax) return(1);
}

/** gather q and qc to both communicators **/
if (me<numprocs/2) {
    MPI_Send(&q[1][1][0],xmax*ymax*(tmax+1),MPI_DOUBLE,me+numprocs/2,me,mpi_comm_world);
} else {
    MPI_Recv(&q[1][1][0],xmax*ymax*(tmax+1),MPI_DOUBLE,me−numprocs/2,me,mpi_comm_world,&status);  MPI_Barrier(mpi_comm_world);
}

if (me>=numprocs/2) {
    MPI_Send(&qc[1][1][0],xmax*ymax*(tmax+1),MPI_DOUBLE,me−numprocs/2,me,mpi_comm_world);
} else {
    MPI_Recv(&qc[1][1][0],xmax*ymax*(tmax+1),MPI_DOUBLE,me+numprocs/2,me+numprocs/2,mpi_comm_world,&status);  MPI_Barrier(mpi_comm_world);
}    
return(0);
}

void calc_dens() {
    double Q,temp,temp2;
    int it,x,y, z;

    QCP=0.; QA=0.; QB=0.;

    for (x=1;x<=xmax;x++)
        for (y=1;y<=ymax;y++) {
            /* calculate phiA */
            temp=0.;
            temp2=0.;
            if (CP) {
                for (it=1;it<((f*tmax)−1);it++)
                    temp+=q[x][y][it]*qc[x][y][tmax−it];
                temp+=q[x][y][0]*qc[x][y][tmax]/2.;
                temp+=(q[x][y][(int)((f*tmax)]*qc[x][y][(int)((f*tmax)])/2.;
            }
            if (GC) temp*=zCP*dt;
            else temp*=nCP*dt/QCP;
            volCP+=temp;
            if (HA) {
                for (it=0;it<((f*tmax)−1);it++)
                    temp2+=qA[x][y][it]*qA[x][y][tmmax−it];
                temp2*=zA*dt;
                volHA+=temp2;
            } else {
                for (it=0;it<((f*tmax)−1);it++)
                    temp2+=qB[x][y][it]*qB[x][y][tbmax−it];
                temp2*=zB*dt;
                volHB+=temp2;
            }
        }
    Q*=dx*dy*dt/NA;
    QA*=dx*dy*dt/NA;
    QB*=dx*dy*dt/NB;
    QCP+=e[1]; QA+=e[2]; QB+=e[3];
    MPI_Allreduce(e[1],e[2],e[3],MPI_DOUBLE,MPI_SUM,my_comm);
    QCP=QCP/6.; QA=QA/6.; QB=QB/6.;
    volCP=0.; volHA=0.; volHB=0.;
}
```
```c
Sim2dMPI.c

volHA+=temp2;
phiA[x][y]=(temp+temp2);
phiAH[x][y]=temp2;

/* calculate phiB */
temp=0.;
temp2=0.;
if (CP) {
  for (it=(f*tmax)+1;it<tmax;it++)
    temp+=q[x][y][it]*qc[x][y][tmax−it];
  temp+=(q[x][y][(int)(f*tmax)]*qc[x][y][(int)(f*tmax)])/2.420;
  if (GC) temp*=zCP*dt;
  else temp*=nCP*dt/QCP;
}
if (HB) {
  for (it=0;it<tbmax;it++)
    temp2+=qB[x][y][it]*qB[x][y][tbmax−it];
  if (GC) temp2*=zB*dt;
  else temp2*=nB*dt/(QB);
}
volCP+=temp;
volHB+=temp2;
phiB[x][y]=(temp+temp2);
phiBH[x][y]=temp2;
}

Sim2dMPI.c

#include <mpi.h>
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <time.h>

/* Pick dwm[] and add to wp. */
void pick_add_dwm() {
  int i,x,y;
  i=1;
  for (x=1;x<=xmax;x++)
    for (y=1;y<=ymax;y++)
      wm[x][y]+=dwm[i++];
}

/* Subtract dwm[] from wp. */
void sub_dwm() {
  int i,x,y;
  i=1;
  for (x=1;x<=xmax;x++)
    for (y=1;y<=ymax;y++)
      wp[x][y]−=dwm[i++];
}

/* Calculate the Hamiltonian. */
double calc_h1() {
  int x,y,z;
  double a,b,c,xi;
  a=0.; c=0.;
  for (x=1;x<=xmax;x++)
    for (y=1;y<=ymax;y++)
      wp[x][y]+=dwp[i++];
  a+=c; a*=dx*dy; e[1]=a; MPI_Allreduce(&e[1],&e[2],1,MPI_DOUBLE,MPI_SUM,my_comm); a=e[2];
}
```
a+V*chiN/4.;
if (GC) {
if (CP) a-=zCP*QCP;
if (HA) a-=zA*QA;
if (HB) a-=zB*QB;
}
elbow (CP) a-=zCP*QCP;
if (HA) a-=zA*log(QA);
if (HB) a-=zB*log(QB);
return (a*C);
}
****************************************************************************/
/*  'Variable lambda' mixing for iteration of W+, xi.                       */
void varlam(int iter, int nvar) {
double sum1,sum2;
int nv;
if (iter>1) {
sum1=0.; sum2=0.;
for (nv=1;nv<=nvar;nv++) {
sum1 += (var[nv][0]-var[nv][1])^2;
sum2 += ...}
e[2]=sum2; MPI_Allreduce(&e[1],&e[3],2,MPI_DOUBLE,MPI_SUM,my_comm); sum1=e[3]; sum2=e[4];
if (sum2>(1./(double)nvar)) {
vlam = sqrt(sum1/sum2);
if (vlam<vlammin) vlam=vlammin;
if (vlam>vlammax) vlam=vlammax;
else vlam=vlammin;
}
else vlam=vlam00;
for (nv=1;nv<=nvar;nv++) {
var[nv][1] = var[nv][0];
dvar[nv][1] = dvar[nv][0];
var[nv][0] += vlam*dvar[nv][0];
}
} /* iteratively find self-consistent W+. */
} /* */
for (i=1;i<=xmax+nx;)
    for (j=1;j<=nymax+ny;)
        temp5=var[i++][0];
        temp3=var[i++][0];
        test = (temp3-dwpFL[x][y])*(temp3-dwpFL[x][y]);
        test += (temp5-dwpFL[x][y])*(temp5-dwpFL[x][y]);
        xi[x][y]=temp3;
        dwpFL[x][y]=temp5;
        wp[x][y]=temp3+temp5;
    e[1]=test;
    MPI_Allreduce(&e[1],&e[2],1,MPI_DOUBLE,MPI_SUM,my_comm);
    test=e[2];
    test=sqrt(test);
    iter++;
    i=propag();
    calc_dens();
    oldf=newf;
    newf=calc_h1();
while((iter<=relaxnrFL)&&(test>relaxerrFL));
avtest+=test;
avferr+=fabs(newf-oldf);
return(0);

Sim2dMPI.c

Sim2dMPI.c
else {
  for (x=1;x<=xmax;x++)
  for (y=1;y<=ymax;y++) {
    phiA[x][y]=.5;
    phiB[x][y]=.5;
    wm[x][y]=0.;
    wp[x][y]=chiN/2.;
  }
}

/******************************************************************************
/* Simulation of W- fluctuations (Monte Carlo). */

void sim() {
  double r0,r1;
  int idummy;
  int i,j,k,x,y,Jflag,Rflag,update,OK;
  double temp,temp2,test,deltah,getrnd;
  i=i_start;
  relaxerrFL=relaxerrstartFL;
  if (LOAD==0) {
    idummy=propag();
    calc_dens();
  if (me==0) {
    if (!run) fprintf(ifile, "-0.");
    fprintf(ifile, "%f %f %f %f %f %f
", volCP,volHA,volHB,QCP,QA,QB);
    fflush(ifile);
  }  
  if (LOAD==2) j=relaxFL(1);
  idummy=propag();
  calc_dens();
  h1=calc_h1();
  if (i_start==0) hinit=h1;
  oldh1=h1;
  for (x=1;x<=xmax;x++)
  for (y=1;y<=ymax;y++) {
    savephiA[x][y]=phiA[x][y];
    savephiB[x][y]=phiB[x][y];
    savephiAH[x][y]=phiAH[x][y];
    savephiBH[x][y]=phiBH[x][y];
  }
  succmove=0;
  diffcount=0;
  avtest=0.0;
  avferr=0.0;
}

while (i<=MCiter) {
  fflush(stdout);
  OK=0;
  i++;
  pick_add_dwm();
  for (x=1;x<=xmax;x++)
  for (y=1;y<=ymax;y++) {
    savevp[x][y]=wp[x][y];
  }
  if ((me==0)&&(i%100)==0) {
    if (run) fprintf(file,"-%s-");
    fprintf(file,"%d.");
  }
  if ((me==0)&&(i%100)==0) {
    fprintf(file,"%f %f %f %f %f %f
", volCP,volHA,volHB,QCP,QA,QB);
    fflush(file);
    h1=calc_h1();
    deltah=(h1−oldh1);
    getrnd=rndnr();
  if (me==0) {
    if (deltah<0.) OK=1;
    else {
      if (getrnd<(exp(−deltah))) OK=1;
    }
    MPI_Bcast(&OK,1,MPI_INT,0,mpi_comm_world);
  if ((me==0)&&(i%100)==0) {
    if (run) fprintf(file,"-%s-");
    fprintf(file,"%d. %f %f 
", i,oldh1−hinit,oldh1);
    fflush(file);
  }
  if (OK==0) {
    sub_dwm();
    for (x=1;x<=xmax;x++)
    for (y=1;y<=ymax;y++) {
      wp[x][y]=savevp[x][y];
    }
    else {
      succmove++;      oldh1=h1;
      for (x=1;x<=xmax;x++)
      for (y=1;y<=ymax;y++) {
        savephiA[x][y]=phiA[x][y];
        savephiB[x][y]=phiB[x][y];
        savephiAH[x][y]=phiAH[x][y];
        savephiBH[x][y]=phiBH[x][y];
      }
      succmove=0;
      diffcount=0;
      avtest=0.0;
      avferr=0.0;
    }

if ((i%newmcstep)==0) {
    accrate=(((double)succmove)/(double)newmcstep);
diffrate=(((double)diffcount)/(double)newmcstep);
avtest/=((double)newmcstep);
avferr/=((double)newmcstep);
}

if (me==0) {
    if (!run) fprintf(afile,"-%");
    fprintf(afile,"%d. %f %f %f %f %f
",i,accrate,MCstep,diffrate,avtest,avferr);
    fflush(afile);
}

sfile=fopen(sname,"w");
if (!run) fprintf(sfile,"-%");
fprintf(sfile,"%d
",i);
fprintf(sfile,"%f
",MCstep);
fprintf(sfile,"%f
",hinit);
fclose(sfile);

if (run==0) {
    if (abs(accrate-acgoal)<accerr) {
        countacc++;
        if (countacc==accnum) {
            run=1;
            i=0;
            if (!run) fprintf(afile,"-%");
            fprintf(afile,"%d
",i);
            fflush(afile);
        }
    } else {
        oldMCstep=MCstep;
        MCstep*=1.+accgoal*(accurate-acgoal);
        if (!run) fprintf(afile,"-%");
        fprintf(afile,"%d
",i);
        fflush(afile);
    }
}

if ((i%savestep)==0) write_conf(i);

read_in();
fgets(line,80,file); fgets(line,80,file); sscanf(line,"%d",&relaxnrFL);
fgets(line,80,file); sscanf(line,"%lf",&relaxerrstartFL);
fgets(line,80,file); sscanf(line,"%lf",&vlam00);
fgets(line,80,file); sscanf(line,"%lf",&vlammin);
fgets(line,80,file); sscanf(line,"%lf",&vlammax);
fgets(line,80,file); sscanf(line,"%d",&newmcstep);
fgets(line,80,file); sscanf(line,"%d",&savestep);
fgets(line,80,file); sscanf(line,"%lf",&accgoal);
fgets(line,80,file); sscanf(line,"%lf",&accerr);
fgets(line,80,file); sscanf(line,"%d",&accnum);

fclose(file); nx=xmax; ny=ymax; xmax/=(numprocs/2); 

****************************************************************************/*
 main()                                                                  */
/*****************************************************************************/
int main(int argc, char *argv[]) {
    int my_x,ME;
    int my_y;
    int job,color;
    FILE * file;
    char type[20],line[80],name1[20],name2[20],name3[20],name4[20],name5[20],name6[20],name7[20],name8[20];
    time(&start_time);
    MPI_Init(&argc, &argv);
    sprintf(NAME,"%s",argv[1]);
    if(argc==1) {
        if(me==0) printf("No filenames given! Exit.
");
        return 0;
    }  

    file = fopen(argv[1],"r");
    fgets(line,80,file);
    sscanf(line,"%s %s %s %s %s %s %s %s",name1,name2,name3,name4,name5,name6,name7,name8);
    fclose(file);

    MPI_Comm_split(MPI_COMM_WORLD,job,0,&mpi_comm_world);  
    MPI_Comm_rank(mpi_comm_world,&me);  
    numprocs=16;
    if(me<numprocs/2) color=0;
     else color=1;
    MPI_Comm_split(mpi_comm_world,color,0,&my_comm);  
    sprintf(type,"a");
    if(me==0) {
        sprintf(aname,"%s_a.dat",NAME);
        afile = fopen(aname,type);    
        sprintf(fname,"%s_f.dat",NAME);
        ffile = fopen(fname,type);    
        sprintf(iname,"%s_i.dat",NAME);
        ifile = fopen(iname,type);    
        sprintf(cname,"%s.cnf",NAME);
        cfile = fopen(cname,type);  

        read_in();   
        i_start=0;  
        if(me==0) {
            sprintf(sname,"%s.stat",NAME);
            if((sfile=fopen(sname,"r"))!=NULL) {
                fgets(line,80,sfile);      
                sscanf(line,"%d",&i_start);
            }  
        }  

        read_in();
        i_start=0;
    }  
    else {
        read_in();
        i_start=0;
    }

    MPI_Comm_split(MPI_COMM_WORLD,color,0,&my_comm);  
    sprintf(type,"a");
    if(me==0) {
        sprintf(aname,"%s_a.dat",NAME);
        afile = fopen(aname,type);    
        sprintf(fname,"%s_f.dat",NAME);
        ffile = fopen(fname,type);    
        sprintf(iname,"%s_i.dat",NAME);
        ifile = fopen(iname,type);    
        sprintf(cname,"%s.cnf",NAME);
        cfile = fopen(cname,type);  

        read_in();
        i_start=0;
    }  
    else {
        read_in();
        i_start=0;
    }

    MPI_Comm_split(MPI_COMM_WORLD,color,0,&my_comm);  
    sprintf(type,"a");
    if(me==0) {
        sprintf(aname,"%s_a.dat",NAME);
        afile = fopen(aname,type);    
        sprintf(fname,"%s_f.dat",NAME);
        ffile = fopen(fname,type);    
        sprintf(iname,"%s_i.dat",NAME);
        ifile = fopen(iname,type);    
        sprintf(cname,"%s.cnf",NAME);
        cfile = fopen(cname,type);  

        read_in();
        i_start=0;
    }  
    else {
        read_in();
        i_start=0;
    }

    MPI_Comm_split(MPI_COMM_WORLD,color,0,&my_comm);  
    sprintf(type,"a");
    if(me==0) {
        sprintf(aname,"%s_a.dat",NAME);
        afile = fopen(aname,type);    
        sprintf(fname,"%s_f.dat",NAME);
        ffile = fopen(fname,type);    
        sprintf(iname,"%s_i.dat",NAME);
        ifile = fopen(iname,type);    
        sprintf(cname,"%s.cnf",NAME);
        cfile = fopen(cname,type);  

        read_in();
        i_start=0;
    }  
    else {
        read_in();
        i_start=0;
    }

    MPI_Comm_split(MPI_COMM_WORLD,color,0,&my_comm);  
    sprintf(type,"a");
    if(me==0) {
        sprintf(aname,"%s_a.dat",NAME);
        afile = fopen(aname,type);    
        sprintf(fname,"%s_f.dat",NAME);
        ffile = fopen(fname,type);    
        sprintf(iname,"%s_i.dat",NAME);
        ifile = fopen(iname,type);    
        sprintf(cname,"%s.cnf",NAME);
        cfile = fopen(cname,type);  

        read_in();
        i_start=0;
    }  
    else {
        read_in();
        i_start=0;
    }

    MPI_Comm_split(MPI_COMM_WORLD,color,0,&my_comm);  
    sprintf(type,"a");
    if(me==0) {
        sprintf(aname,"%s_a.dat",NAME);
        afile = fopen(aname,type);    
        sprintf(fname,"%s_f.dat",NAME);
        ffile = fopen(fname,type);    
        sprintf(iname,"%s_i.dat",NAME);
        ifile = fopen(iname,type);    
        sprintf(cname,"%s.cnf",NAME);
        cfile = fopen(cname,type);  

        read_in();
        i_start=0;
    }  
    else {
        read_in();
        i_start=0;
    }
fgets(line, 80, sfile);  sscanf(line, "%lf", &MCstep);
fgets(line, 80, sfile);  sscanf(line, "%lf", &hinit);
fclose(sfile);  

if (i_start<0) {  
i_start= -i_start;    run=0;  }  
else if (i_start>0) run=1;
if (i_start) {  
sprintf(READNAME, "%s", NAME);  
LOAD=1;  }
if (LOAD) {  
plan = rfftw2d_mpi_create_plan(my_comm, 
    nx, ny, FFTW_REAL_TO_COMPLEX, FFTW_MEASURE);
iplan = rfftw2d_mpi_create_plan(my_comm, 
    nx, ny, FFTW_COMPLEX_TO_REAL, FFTW_MEASURE);

plan = rfftwnd_mpi_create_plan(plan, 
    nlocal_nx, local_nx_start, 
    nlocal_ny_after_transpose, local_y_start_after_transpose, 
    total_local_size);

iplan = rfftwnd_mpi_create_plan(iplan, 
    nlocal_nx, local_nx_start, 
    nlocal_ny_after_transpose, local_y_start_after_transpose, 
    total_local_size);

init();
read_conf();
sim();
close();
if (me==0) {  
    fclose(afile);    fclose(ffile);    fclose(ifile);    fclose(cfile);  }
    rfftwnd_mpi_destroy_plan(plan);  
    rfftwnd_mpi_destroy_plan(iplan);  
    MPI_Barrier(MPI_COMM_WORLD);  
    MPI_Finalize();
    return(0);
}
/* for varlam mixing scheme */

double vlam;
double vlam00;
double vlammin;
double vlammax;

int succmove=0;
int succmovep=0;
int restarts=0;
int savestep;
int wptstep;
int wptnum;
int wptstep;

double accrate;
double accgoal;
double accerr;
double accratep;
double accgoalp;
double accerrp;

int countacc=0;
int countw, countp;

double MCstep;
double MCpstep;

double oldMCstep, oldMCpstep;

double chiH;
double zCP;
double zA;
double zB;

int GC;
/* 0: canonical ensemble, 1: grand canonical ensemble */

double xsize;
double ysize;
double C;

int xmax;
int ymax;
int tmax;

int runw, runp, run;

char NAME[40];
char READNAME[40];

int LOAD;

int maxdim;

int tanax, tmax;

int basedim;

double V, dx, dy, dt;

double **dwpFL, **xi;
double **var,**dvar;

/* 'ranmar' random number generator */

double r_QCP, r_QA, r_QB, th_QCP, th_QA, th_QB;
double voICP, voIA, voIB;
double finit;

double **var,**dvar;

/* 'ranmar' random number generator */

void rmarin(int ijk1) {

int i,j,k,m;

double s,t;

int ii,jj,ij,kl;

ij = ijk1/30082;
kl = ijk1−(ij*30082);  i  = ((ij/177) % 177)+2;  j  = (ij % 177)+2;  k  = ((kl/169) % 178)+1;  l  = kl % 169;

for(ii=1;ii<=97;ii++) {
s = 0.;    t = 0.5;

for(jj=1;jj<=24;jj++) {
m = (((i*j)%179)*k) % 179;      i = j;      j = k;      k = m;      l = (53*l+1) % 169;

if(((l*m)%64)>=32) s+=t;

}

u[ii]=s;
}

c = 362436. / 16777216.;  cd = 7654321. / 16777216.;  cm = 16777213. / 16777216.;  i97 = 97;

for(ii=2;ii<=97;ii++) im_ran[ii]=ii−1;  im_ran[1] = 97;

for(ii=1;ii<=97;ii++) {

if(ii==97) ii+=97;  ii++;

a = 0.1;

t = 0.5;

for(jj=1;jj<=24;jj++) {

m = (((i*j)%179)*k) % 179;

j = k;

k = m;

h = (53*l+1) % 169;

if(((l*m)%64)>=32) s+=t;

}

u[ii]=s;
}

c = 362436. / 16777216.;  cd = 7654321. / 16777216.;  cm = 16777213. / 16777216.;  i97 = 97;

j97 = 33;

for(ii=2;ii<=97;ii++) im_ran[ii]=ii−1;

im_ran[1] = 97;
}
double randomnr() {
    int i;
    double uni;
    uni = u[i97]−u[j97];
    if (uni<0) uni+=1.;
    u[i97] = uni;
    j97 = im_ran[j97];
    c−=cd;
    if (c<0.) c+=cm;
    uni−=c;
    if (uni<0.) uni+=1.;
    return (uni);
}

void init() { /* Initialize */
    /* Initialize: */
    int x,y;
    double ft;
    int i,j,k1,k2;
    double *a;
    double d2;
    dt=1./(double)tmax;
    dxsize/(double)xmax;
    dyysize/(double)ymax;
    p_real = rfftw2d_create_plan(nx,ny,FFTW_REAL_TO_COMPLEX,
        FFTW_ESTIMATE | FFTW_IN_PLACE);
    pinv_real = rfftw2d_create_plan(nx,ny,FFTW_COMPLEX_TO_REAL,
        FFTW_ESTIMATE | FFTW_IN_PLACE);
    d2=4*M_PI*M_PI/(nx*ny*dx*dy)*ds;  data_real = dmatrix(0,nx−1,0,ny+1);  exp_wh_factor=dvector(0,nx*ny−1);    exp_lapl_factor_real=dvector(0,nx*(ny+2)−1);  a = exp_lapl_factor_real;
    for(i=0; i<nx/2; i++) {
        for(j=0; j<ny; j++) {
            k1 = i;
            k2 = j/2;
            *(a++) = exp(−(k1*k1 + k2*k2)*d2);    }
        for(i=nx/2; i<nx; i++) {
            for(j=0; j<ny; j++) {
                k1 = nx − i;
                k2 = j/2;
                *(a++) = exp(−(k1*k1 + k2*k2)*d2);    }
        }
    p_complex = fftw2d_create_plan(nx,ny,FFTW_FORWARD,
        FFTW_ESTIMATE | FFTW_IN_PLACE);
    pinv_complex = fftw2d_create_plan(nx,ny,FFTW_BACKWARD,
        FFTW_ESTIMATE | FFTW_IN_PLACE);
    two_over_nxny=2./(nx*ny);
    ds_half=ds/2.;
    data_complex = dmatrix(0,nx−1,0,2*ny−1);  real_factor=dvector(0,nx*ny−1);  imag_factor=dvector(0,nx*ny−1);  a = exp_lapl_factor_complex;
    for(i=0; i<nx/2; i++) {
        for(j=0; j<ny; j++) {
            k1 = i;
            k2 = j/2;
            *(a++) = exp(−(k1*k1 + k2*k2)*d2);    }
    for(i=nx/2; i<nx; i++) {
        for(j=0; j<ny; j++) {
            k1 = nx − i;
            k2 = j/2;
            *(a++) = exp(−(k1*k1 + k2*k2)*d2);    }
    }
    phiA = dmatrix(1,xmax,1,ymax);  phiB = ... = dmatrix(1,xmax,1,ymax);  phiAi = dmatrix(1,xmax,1,ymax);  phiBi = dmatrix(1,xmax,1,ymax);  basedim=xmax*ymax;   tamax=(int)(tmax*NA);
    tbmax=(int)(tmax*NB);
    var = dmatrix(1,(2*xmax*ymax),0,1);  dvar = dmatrix(1,(2*xmax*ymax),0,1);
    if (CP) {
        q = f3tensor(1,xmax,1,ymax,0,tmax);    qc = f3tensor(1,xmax,1,ymax,0,tmax);    qr = f3tensor(1,xmax,1,ymax,0,tmax);    qcr = f3tensor(1,xmax,1,ymax,0,tmax);    qi = f3tensor(1,xmax,1,ymax,0,tmax);    qci = f3tensor(1,xmax,1,ymax,0,tmax);
    if (HA) {
        qA=q;      qAr=qr;      qAi=qi;    }
    if (HB) {
        qB=qc;      qBr=qcr;      qBi=qci;
    }
    if (HC) { qH=q; qHr=qr; qHi=qi; }
    if (HH) { qH=q; qHr=qr; qHi=qi; }
}
```c
else
  if (HA) {
    qA = f3tensor(1,xmax,1,ymax,0,tamax);
    qAr = f3tensor(1,xmax,1,ymax,0,tamax);
    qAi = f3tensor(1,xmax,1,ymax,0,tamax);
  }
  if (HB) {
    qB = f3tensor(1,xmax,1,ymax,0,tbmax);
    qBr = f3tensor(1,xmax,1,ymax,0,tbmax);
    qBi = f3tensor(1,xmax,1,ymax,0,tbmax);
  }
  d = f3tensor(1,xmax,1,ymax,0,1);
  u=dvector(1,97);
  im_ran=ivector(1,97);
  rmarin(54217137);
  for (x=1;x<=xmax;x++)
    for (y=1;y<=ymax;y++) {
      if (CP) {
        q[x][y][0]=1.;
        qc[x][y][0]=1.;
        qr[x][y][0]=1.;
        qcr[x][y][0]=1.;
        qi[x][y][0]=0.;
        qci[x][y][0]=0.;
      }
      if (HA) {
        qA[x][y][0]=1.;
        qAr[x][y][0]=1.;
        qAi[x][y][0]=0.;
      }
      if (HB) {
        qB[x][y][0]=1.;
        qBr[x][y][0]=1.;
        qBi[x][y][0]=0.;
      }
    }
  V=xsize*ysize;
  nCP=(V*(1.-ph));
  nA=((V*ph)/(2.*NA));
  nB=nA;
  dx=(xsize/xmax);
  dy=(ysize/ymax);
  dt=(tsize/tmax);
  ...  
  dwp = dvector(1,basedim);
  dwm = dvector(1,basedim);
  dwpt = dvector(1,basedim);
  savewp = dmatrix(1,xmax,1,ymax);

savephiA = dmatrix(1,xmax,1,ymax);
savephiB = dmatrix(1,xmax,1,ymax);
savephiAr = dmatrix(1,xmax,1,ymax);
savephiBr = dmatrix(1,xmax,1,ymax);
savephiAi = dmatrix(1,xmax,1,ymax);
savephiBi = dmatrix(1,xmax,1,ymax);
}

void close() {
  free_dvector(exp_lapl_factor_real,0,nx*(ny+2)-1);
  free_dvector(exp_wh_factor,0,nx*ny-1);
  free_dmatrix(data_real,0,na,0,ny);
  rfftwd_destroy_plan(p_real);
  rfftwd_destroy_plan(pinv_real);
  free_dmatrix(data_complex,0,na-1,0,2*ny-1);
  free_dvector(real_factor,0,na*ny-1);
  free_dvector(imag_factor,0,na*ny-1);
  free_dvector(exp_lapl_factor_complex,0,na*ny-1);
  free_dvector(exp_wh_factor_complex,0,na*ny-1);
  free_dmatrix(phia,1,xmax,1,ymax);
  free_dmatrix(phiaR,1,xmax,1,ymax);
  free_dmatrix(phiaI,1,xmax,1,ymax);
  free_dmatrix(phiaRi,1,xmax,1,ymax);
  free_dmatrix(phiaBi,1,xmax,1,ymax);

if (CP) {
  free_f3tensor(q,1,xmax,1,ymax,0,tmax);
  free_f3tensor(qc,1,xmax,1,ymax,0,tmax);
  free_f3tensor(qr,1,xmax,1,ymax,0,tmax);
  free_f3tensor(qcr,1,xmax,1,ymax,0,tmax);
  free_f3tensor(qi,1,xmax,1,ymax,0,tmax);
  free_f3tensor(qci,1,xmax,1,ymax,0,tmax);
}
else {
  if (HA) {
    free_f3tensor(qA,1,xmax,1,ymax,0,tamax);
    free_f3tensor(qAr,1,xmax,1,ymax,0,tamax);
    free_f3tensor(qAi,1,xmax,1,ymax,0,tamax);
  }
  if (HB) {
    free_f3tensor(qB,1,xmax,1,ymax,0,tbmax);
    free_f3tensor(qBr,1,xmax,1,ymax,0,tbmax);
    free_f3tensor(qBi,1,xmax,1,ymax,0,tbmax);
  }
}
  free_dmatrix(dwp,1,xmax,1,ymax,0,1);
  free_dvector(u,1,97);
  free_ivector(im_ran,1,97);
  free_dmatrix(savewp,1,xmax,1,ymax);
```
Evaluate diffusion eqn. with DF scheme to find propagator from s0 to s1, using field w, and store in q[x][y][z][t].

```c
int calc_q(double ***q, int w, int s0, int s1) {
    int i, j, s;
    double *a, *b, *c;
    int nxny = xmax*ymax;
    int nxny2 = xmax*(ymax+2);
    b = exp_wh_factor;  
    c = &d[1][1][w];
    for (i=0; i<nxny; i++) {
        *(b++) = exp(-(*c)*ds_half);    c+=2;  
    }
    for (s=s0-1; s<s1; s++) {
        a = data_real[0];    b = exp_wh_factor;    c = &q[1][1][s];
        for (i=0; i<nx; i++) {
            for (j=0; j<ny; j++) {
                *(a++) = *(b++) * *(c);c+=ns+1;
            }      a+=2;    }
        rfftwnd_one_real_to_complex(p_real, &data_real[0][0], NULL);
        a = data_real[0];    b = exp_lapl_factor_real;
        for (i=0; i<nxny2; i++) {
            *(a++) = *(b++);    rfftwnd_one_complex_to_real(pinv_real, (fft_real *)&data_real[0][0], NULL);
        }
    }
    return 0;
}
```

Define d[x][y][w] (used in calc_q()).

```c
void init_d() {
    int x, y, z;
    double d0, d1;
    for (x=1;x<=xmax;x++)
        for (y=1;y<=ymax;y++) {
            d[x][y][0]=(wp[x][y]+wm[x][y]); /* WA */
            d[x][y][1]=(wp[x][y]-wm[x][y]); /* WB */
        }
}
```

Calculate the complete propagator (applies to copolymers) using field wa from t0 to t1-1, wb from t1 to t2-1.

```c
int init_q(double ***q, int wa, int wb, int t0, int t1, int t2) {
    if (calc_q(q,wa,t0,t1)) return (1);
    if (calc_q(q,wb,t1+1,t2)) return (1);
    return (0);
}
```

Calculate propagators of all polymer species.

```c
int propag() {
    int x, it;
    double d;
    init_d();
    if (CP) {
```
# Appendix B. Program Listings

```c
if (init_q(q,0,1,1,*(tmax,tmax))) return(1); /* (q,w0,w,...) */
if (init_q(qc,1,0,1,((1-f)*tmax),tmax)) return(1);
/* (qc,w1,w0,...) */
else {
    if ((HA)&&(calc_q(qA,0,1,tamax))) return(1);
    if ((HB)&&(calc_q(qB,1,1,tbmax))) return(1);
}
return(0);
/******************** COMPLEX VERSION***************************/
int calc_q_complex(double **qr, double **qi, int w, int t0, int t1) {
    /* solves the diffusion equation, assumes the laplacian operator */
    /* in k-space is ((2*pi*k/n)/dx)^2. complex version */
    int i,j,s;
    double *a, *b, *c, *g, *e;
    int nxy=mx*ny;
    int nxy2=mx*(ny+2);
    b=exp_wh_factor;  c=&d[1][1][w];
    for (i=0; i<nxny; i++) {
        *(b++) = exp(- *c * ds_half);     c+=2;
    }
    for (s=t0-1; s<t1; s++) {
        /*** step in qr ***/
        a=data_real[0];    b=exp_wh_factor;    c=&qr[1][1][s];
        for (i=0; i<nx; i++) {
            for (j=0; j<ny; j++) {
                *(a++) = *(b++) * (*c);
                c+=ns+1;
            }
            a+=2;
        }
        rfftwnd_one_real_to_complex(p_real,&data_real[0][0],NULL);
        a=data_real[0];
        b=exp_lapl_factor_real;
        for (i=0; i<nxny2; i++)
            *(a++) *= *(b++);
        rfftwnd_one_complex_to_real(pinv_real,(fftw_complex *)&data_real[0][0],NULL);
        a=data_real[0];    b=exp_wh_factor;    c=&qr[1][1][s+1];
        g=&wpt[1][1];    e=&qi[1][1][s];
        for (i=0; i<nx; i++) {
            for (j=0; j<ny; j++) {
                *c = *(b++) * *(a++) / (double)nxny - *(g++) * *e * ds;
                c+=ns+1;
                e+=ns+1;
            }
            a+=2;
        }
    }
    return(0);
}
/******************** */
int init_q_complex(double **qr, double **qi, int w, int t0, int t1, int t2) {
    if (calc_q_complex(qr,qi,wa,t0,t1)) return(1);
    if (calc_q_complex(qr,qi,wb,t1,t2)) return(1);
} FullSim2d.c
```
return(0);
}
/******************************/
int propag_complex()
{
    int x, it;
    init_d();
    if (CP) {
        if (init_q_complex(qr, qi, 0, 1, 1, f*tmax, tmax)) return(1);
        /* (q,w0,w1,..,w0,w1) */
        if (init_q_complex(qcr, qci, 1, 0, 1, ((1-f)*tmax), tmax)) return(1);
        /* (qc,w1,w0,..,w0,w1) */
    } else {
        if ((HA)&&(calc_q_complex(qAr, qAi, 0, 1, tamax))) return(1);
        if ((HB)&&(calc_q_complex(qBr, qBi, 1, 1, tbmax))) return(1);
    }
    return(0);
}/**********END COMPLEX VERSION************************************/
void calc_Q_complex()
{
    int it, x, y, z;
    QCPr=0.;  QAr=0.;  QBr=0.;  QCPi=0.;  QAi=0.;  QBi=0.;
    for (x=1;x<=xmax;x++)
        for (y=1;y<=ymax;y++) {
            if (CP) {
                for (it=1;it<tmax;it++) {
                    QCPr+=qr[x][y][it]*qcr[x][y][tmax-it]; QCPr+=qr[x][y][0]*qcr[x][y][tmax]/2.;
                    QCPi+=q[cr[x][y][it]*qci[x][y][tmax-it]; QCPi+=q[cr[x][y][0]*qci[x][y][tmax]/2.;
                }
            }
            if (HA) {
                for (it=0;it<tamax;it++) {
                    QAr+=qAr[x][y][it]*qAr[x][y][tamax-it];
                    QAr+=qAr[x][y][0]*qAr[x][y][tamax]/2.;
                    QAi+=qAr[x][y][it]*qAr[x][y][tamax-it];
                    QAi+=qAr[x][y][0]*qAr[x][y][tamax]/2.;
                }
            }
        }
    QCPr*=dx*dy*dt;
    QAr*=dx*dy*dt/NA;
    QBr*=dx*dy*dt/NB;
    QCPi*=dx*dy*dt;
    QAi*=dx*dy*dt/NA;
    if (CP) r_QCP=sqrt(QCPr*QCPr+QCPi*QCPi);
    if (HA) r_QA=sqrt(QAr*QAr+QAi*QAi);
    if (HB) r_QB=sqrt(QBr*QBr+QBi*QBi);
    if (CP) th_QCP=atan(QCPi/QCPr);
    if (HA) th_QA=atan(QAi/QAr);
    if (HB) th_QB=atan(QBi/QBr);
}/****************************************************************************/* Calculate the densities from the propagators. */**************************************************************************/
void calc_dens()
{
    double Q, temp, temp2;
    int it, x, y, z;

    /***** Saddle point densities (real) *****/
    QCP=0.;
    QA=0.;
    QB=0.;
    for (x=1;x<=xmax;x++)
        for (y=1;y<=ymax;y++) {
            if (CP) {
                for (it=1;it<tmax;it++) {
                    QCP+=q[x][y][it]*qcr[x][y][tmax-it];
                    QCP+=q[x][y][0]*qcr[x][y][tmax]/2.;
                    QCP+=q[x][y][it]*qci[x][y][tmax-it];
                    QCP+=q[x][y][0]*qci[x][y][tmax]/2.;
                }
                if (HA) {
                    for (it=0;it<tamax;it++) {
                        QA+=qAr[x][y][it]*qAr[x][y][tamax-it];
                        QA+=qAr[x][y][0]*qAr[x][y][tamax]/2.;
                        QA+=qAr[x][y][it]*qAr[x][y][tamax-it];
                        QA+=qAr[x][y][0]*qAr[x][y][tamax]/2.;
                    }
                }
            }
        }
```c
}    QCP*=dx*dy*dt;  QA*=dx*dy*dt/NA;  QB*=dx*dy*dt/NB;    volCP=0.;  volHA=0.;  volHB=0.;
for (x=1;x<=xmax;x++)
    for (y=1;y<=ymax;y++) {
        /* calculate phiA */
        temp=0.;      temp2=0.;
        if (CP) {
            for (it=1;it<=(f*tmax)-1;it++)
                temp+=q[x][y][it]*qc[x][y][tmax-it];
            temp+=q[x][y][0]*qc[x][y][tmax]/2.;
            temp+=(q[x][y][(int)(f*tmax)]*qc[x][y][(int)(f*tmax)])/2.;
            if (GC) temp*=zCP*dt;
            else temp*=nCP*dt/QCP;
        }
        if (HA) {
            for (it=0;it<tamax;it++)
                temp2+=qA[x][y][it]*qA[x][y][tamax-it];
            if (GC) temp2*=zA*dt;
            else temp2*=nA*dt/(QA);
        }
        volCP+=temp;      volHA+=temp2;      phiA[x][y]=(temp+temp2);
    }
    temp=xmax*ymax;    volCP/=temp;  volHA/=temp;  volHB/=temp;
}    void calc_dens_complex() {
    double Q,tempr,tempi,temp2r,temp2i;
    int x,y,z;
    /***** Full (complex) densities *****/
    for (x=1;x<=xmax;x++)
        for (y=1;y<=ymax;y++) {
            /* calculate phiA */
            tempr=0.;      tempi=0.;      temp2r=0.;      temp2i=0.;
            if (CP) {
                for (it=1;it<=(f*tmax)-1;it++)
                    { tempr+=qr[x][y][it]*qcr[x][y][tmax-it] -
                      qi[x][y][it]*qci[x][y][tmax-it];
                      ... +   qi[x][y][0]*qcr[x][y][tmax]/2.;
                    temp2r+=(qr[x][y][(int)(f*tmax)]*qcr[x][y][(int)(f*tmax)])/2. -
                      (qi[x][y][(int)(f*tmax)]*qci[x][y][(int)(f*tmax)])/2.;
            if (GC) temp2r*=zCP*dt;
            else temp2r*=nCP*dt/QCP;
        }
        if (HA) {
            for (it=0;it<tamax;it++)
                { tempi+=(qr[x][y][it]*qci[x][y][tmax-it] +
                  qi[x][y][it]*qcr[x][y][tmax-it];
                }
            if (GC) temp2i*=zCP*dt;
            else temp2i*=nCP*dt/QCP;
        }
        if (CP) {
            for (it=(f*tmax)+1;it<tmax;it++)
                { temp+=qr[x][y][it]*qcr[x][y][tmax-it] -
                  qi[x][y][it]*qci[x][y][tmax-it];
                ... +   qi[x][y][0]*qcr[x][y][tmax]/2.;
                temp2i+=(qr[x][y][(int)(f*tmax)]*qci[x][y][(int)(f*tmax)])/2. +
                  (qi[x][y][(int)(f*tmax)]*qcr[x][y][(int)(f*tmax)])/2.;
            if (GC) temp2i*=zCP*dt;
            else temp2i*=nCP*dt/QCP;
        }
        if (HB) {
            for (it=0;it<tbmax;it++)
                { temp2r+=qB[x][y][it]*qB[x][y][tbmax-it];
                ... +   qB[x][y][0]*qB[x][y][tbmax]/2.;
                temp2i+=(qB[x][y][(int)(f*tmax)]*qB[x][y][(int)(f*tmax)])/2. +
                  (qB[x][y][(int)(f*tmax)]*qB[x][y][(int)(f*tmax)])/2.;
            if (GC) temp2r*=zB*dt;
            else temp2r*=nB*dt/QB;
        }
```
if (HA) {
  for (it=0; it< tamax; it++) {
    temp2r+=qAr[x][y][it]*qAr[x][y][tamax-it] -
    qAi[x][y][it]*qAi[x][y][tamax-it];
    temp2i+=qAr[x][y][it]*qAi[x][y][tamax-it] +
    qAi[x][y][it]*qAr[x][y][tamax-it];
    /* temp2r+= .../2.;   temp2r+= .../2.; */
  }
  if (GC) {
    temp2r*=zA*dt;  temp2i*=zA*dt;
  } else {
    temp2r*=0.;//nA*dt/(QA);
    temp2i*=0.;//nA*dt/(QA);
  }
}

if (HB) {
  for (it=0; it< tbmax; it++) {
    temp2r+=qBr[x][y][it]*qBr[x][y][tbmax-it] -
    qBi[x][y][it]*qBi[x][y][tbmax-it];
    temp2i+=qBr[x][y][it]*qBi[x][y][tbmax-it] -
    qBi[x][y][it]*qBr[x][y][tbmax-it];
  }
  if (GC) {
    temp2r*=zB*dt;  temp2i*=zB*dt;
  } else {
    temp2r*=0.;//nB*dt/(QB);
    temp2i*=0.;//nB*dt/(QB);
  }
}

phiAr[x][y]=(tempr+temp2r);
phiAi[x][y]=(tempi+temp2i);

/* calculate phiB */

void add_dwp() {
  int i,x,y;
  i=1;
  for (x=1;x<=xmax;x++)
    for (y=1;y<=ymax;y++)
      wp[x][y]+=dwp[i++];
}

void pick_add_dwm() {
  int i,x,y;
  i=1;
  for (x=1;x<=xmax;x++)
    for (y=1;y<=ymax;y++)
      dwm[i]=((randomnr()*2.)−1.)*MCstep;
      wm[x][y]+=dwm[i++];
}

void pick_add_dwpt() {
  int i,x,y;
  i=1;
  for (x=1;x<=xmax;x++)
    for (y=1;y<=ymax;y++)
      dwpt[i]=((randomnr()*2.)−1.)*MCpstep;
      wpt[x][y]+=dwpt[i++];
}
```c
void sub_dwm() {
    int i, x, y;
    i = 1;
    for (x = 1; x <= xmax; x++)
        for (y = 1; y <= ymax; y++)
            wm[x][y] -= dwm[i++];
}

void sub_dwpt() {
    int i, x, y;
    i = 1;
    for (x = 1; x <= xmax; x++)
        for (y = 1; y <= ymax; y++)
            wpt[x][y] -= dwpt[i++];
}

double calc_h1() {
    int idummy, x, y, z;
    double a, b, c, xi;
    a = 0.; c = 0.;
    for (x = 1; x <= xmax; x++)
        for (y = 1; y <= ymax; y++) {
            b = wm[x][y]; c += b * b / chiN;
            a -= wp[x][y];
        }
    a += c; a *= dx * dy;
    a += V * chiN / 4.;
    if (MCpstep) {
        if (GC) {
            if (CP) a = sCP * QCP;
            if (SH) a = zB * QSH;
        }
        else {
            if (CP) a = nCP * log(QCP);
            if (SH) a = nB * log(QSH);
        }
    }
    return (a * C);
}

double calc_h_bl() {
    int idummy, x, y, z;
    double a, b, c, xi;
    a = 0.; c = 0.;
    for (x = 1; x <= xmax; x++)
        for (y = 1; y <= ymax; y++) {
            b = wm[x][y]; c += b * b / chiN;
            a -= wp[x][y];
        }
    a += c; a *= dx * dy;
    a += V * chiN / 4.;
    if (MCpstep) {
        if (GC) {
            if (CP) a = sCP * QCP;
            if (SH) a = zB * QSH;
        }
        else {
            if (CP) a = nCP * log(r_QCP);
            if (SH) a = nB * log(r_QSH);
        }
    }
    else {
        if (GC) {
            if (CP) a = zA * QCP;
            if (SH) a = zB * QSH;
        }
        else {
            if (CP) a = nA * log(QCP);
            if (SH) a = nB * log(QSH);
        }
    }
    return (a * C);
}
```
if (HB) a = nB*log(QB);
}
return(a*C);
}

/***********************************************************/
/* 'Variable lambda' mixing for iteration of W+, xi. */
/***********************************************************/
void varlam(int iter, int nvar) {
  double sum1,sum2;
  int nv;
  if (iter>1) {
    sum1=0.; sum2=0.;
    for (nv=1;nv<=nvar;nv++) {
      sum1 += (var[nv][0]-var[nv][1])*(var[nv][0]-var[nv][1]);
      sum2 += (dvar[nv][1]-dvar[nv][0])*(dvar[nv][1]-dvar[nv][0]);
    }
    if (sum2>(1./(double)nvar)) {
      vlam = sqrt(sum1/sum2);
      if (vlam<vlammin) vlam=vlammin;
      if (vlam>vlammax) vlam=vlammax;
    }
    else vlam=vlammin;
  }
  else vlam=vlam00;
  for (nv=1;nv<=nvar;nv++) {
    var[nv][1] = var[nv][0];
    dvar[nv][1] = dvar[nv][0];
    var[nv][0] += vlam*dvar[nv][0];
  }
}

/***********************************************************/
/* Iteratively find self-consistent W+ */
/***********************************************************/
int relaxFL(int mode) {
  int i,x,y,z,iter;
  int ngflag;
  double phi,temp3,temp5,sum1,sum2;
  double lam, test,test2;
  double fdep;
  double oldf,newf;
  i=1;
  ngflag=0;
  for (y=1;y<ymax;y++)
    for (x=1;x<xmax;x++)
      for (y=1;y<ymax;y++)
        for (x=1;x<xmax;x++)
          temp3=phiA[x][y];
          temp5=phiB[x][y];
          xi[x][y]=wp[x][y]-chiN/2.;
          /* is minus xi */
          temp3=chiN*(temp3-temp5)/2.;
          dwp[x][y]=chiN/2.;
          var[i++][0]=chiN/2.;
          var[i++][0]=xi[x][y];
    iter=1;
    /* do */
    /* Calculate propagators */
    i=propag();
    /* Calculate phi's */
    calc_dens();
    newf=calc_h1();
    do {
      /* Determine new fields */
      i=1;
      test2=0.;
      for (y=1;y<ymax;y++)
        for (x=1;x<xmax;x++)
          phi=phiA[x][y]+phiB[x][y];
          fdep=chiN*phi/2.;
          test2+=phi;
          dvar[i++][0]=fdep-dwp[x][y];
          dvar[i++][0]=chiN*(phi-1.)/2.;
        varlam(iter,2*xmax*ymax);
        test=0.;
        test2=0.;
        i=1;
        for (y=1;y<ymax;y++)
          for (x=1;x<xmax;x++)
            temp5=var[i++][0];
            temp3=var[i++][0];
            test = (temp3-xi[x][y])*(temp3-xi[x][y]);
            test += (temp5-dwp[x][y])*(temp5-dwp[x][y]);
            xi[x][y]=temp3;
            dwp[x][y]=temp5;
          wp[x][y]=temp3+temp5;
    }
```c
while ((iter<=relaxnrFL)&&(test>relaxerrFL));
return (0);
}

void write_conf(int i) {
    int x,y;
    FILE *f1;
    char fname[20];
    sprintf(fname, "%s_c.cnf", NAME);
    f1=fopen(fname, "w");
    fprintf(f1,"%d
",i);
    fprintf(cfile,"%d
",i);
    for (x=1;x<=xmax;x++)
        for (y=1;y<=ymax;y++) {
            fprintf(f1,"%d. %d. %f %f %f %f %f %f %f %f %f
",x,y,savephiA[x][y],savephiB[x][y],savephiAr[x][y],savephiBr[x][y],savephiAi[x][y],savephiBi[x][y],wp[x][y],wm[x][y],wpt[x][y]);
            fprintf(cfile,"%d. %d. %f %f %f %f %f %f %f %f %f
",x,y,savephiA[x][y],savephiB[x][y],savephiAr[x][y],savephiBr[x][y],savephiAi[x][y],savephiBi[x][y],wp[x][y],wm[x][y],wpt[x][y]);
        }
    fflush(f1);
    fclose(f1);
}

void change_endian(unsigned char *array, int j) {
    int i;
    unsigned char c;
    for (i=0;i<j;i++) {
        c=*(array+3);
        array[3]=array[0];
        array[0]=c;
        c=*(array+2);
        array[2]=array[1];
        array[1]=c;
    }
}
```
else if (LOAD==11) {
    sprintf(fname,"%s_c.cnf",READNAME);
    file=fopen(fname,"r");
    fgets(line,160,file);
    for (x=1;x<=xmax;x++)
        for (y=1;y<=ymax;y++) {
            fgets(line,160,file);
            sscanf(line,"%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf ", &d1, &d2, &phiA[x][y], &phiB[x][y], &phiAr[x][y], &phiBr[x][y], &phiAi[x][y], &phiBi[x][y], &wp[x][y], &wm[x][y], &wpt[x][y]);
        }
    fclose(file);
}
else {
    for (x=1;x<=xmax;x++)
        for (y=1;y<=ymax;y++) {
            phiA[x][y]=.5;phiB[x][y]=.5;phiAr[x][y]=.5;phiBr[x][y]=.5;phiAi[x][y]=.5;phiBi[x][y]=.5;
            wp[x][y]=chiN/2.;wm[x][y]=0.;wpt[x][y]=0.;
        }
}
for (x=1;x<=xmax;x++)
    for (y=1;y<=ymax;y++) {
        savephiA[x][y]=phiA[x][y];
        savephiB[x][y]=phiB[x][y];
        savephiAr[x][y]=phiAr[x][y];
        savephiBr[x][y]=phiBr[x][y];
        savephiAi[x][y]=phiAi[x][y];
        savephiBi[x][y]=phiBi[x][y];
    }

void sim() {
    double deltahnowpi;
    for (x=1;x<=xmax;x++)
        for (y=1;y<=ymax;y++) {
            savephiA[x][y]=phiA[x][y];
            savephiB[x][y]=phiB[x][y];
        }

    /* Simulation of W+ and W− fluctuations of Re(H) (Monte Carlo). */
    /*
    */

    double deltahnowpi;
/* Monte Carlo Loop */
while (i<=MCiter) {
    OK=0;
    i++;
    if (me==0) {
        fflush(fout);
        fflush(afile);
        fflush(ffile);
        fflush(ifile);
        fflush(cfile);
        fflush(wfile);
        fflush(qfile);
    }
    fflush(stdout);
    pick_add_dwm();

    for (x=1;x<=xmax;x++)
    for (y=1;y<=ymax;y++)
        savewp[x][y]=wp[x][y];
    j=relaxFL(1);
    if (MCpstep!=0.) {
        idummy=propag_complex();
        calc_Q_complex();
        calc_dens_complex();
    } else {
        r_QCP=QCP;
        r_QA=QA;
        r_QB=QB;
        th_QCP=0.;
        th_QA=0.;
        th_QB=0.;
        QCPi=0.;
        QAi=0.;
        QBi=0.;
        h1=calc_h1();
        deltalhnowpi=calc_h_nowpi();
        if (deltah<0.) OK=1;
        else {
            if (randomnr()<(exp(-deltah))) OK=1;
        }
    }

    if ((i%savestep)==0) {
        if (!run) fprintf(ffile,"- %d.
",i);
        fprintf(ffile,"%f %f %f %f %f %.14f %d.
",oldh1−finit,oldh1nowpi−finit,finit,deltah,deltahnowpi,deltah−deltahnowpi,OK);
    }
    if (((i%wptstep)==0)&&(MCpstep!=0.)) {
        for (ip=0;ip<wptnum;ip++) {
            OKp=0;
            pick_add_dwpt();
            idummy=propag_complex();
            calc_Q_complex();
            calc_dens_complex();
            if (deltah<0.) OKp=1;
            else {
                if (randomnr()<(exp(-deltah))) OKp=1;
            }
            temp=0.;
            temp2=0.;
            for (x=1;x<=xmax;x++)
            for (y=1;y<=ymax;y++)
                temp+=wpt[x][y];
            temp2+=wpt[x][y]*wpt[x][y];
            if (GC) temp2=zCP*QCPi+zA*QAi+zB*QBi;
            else temp2=(nCP*th_QCP+nA*th_QA+nB*th_QB);
            temp*=dx*dy;
            temp2*=dx*dy;
            printf("%f %f
",temp,temp2);
        }
    }
}
I_new=C*(temp+temp2);
if (OKp) I=I_new;
printf("%f %f %f %d
",temp2,temp+temp2,I,OKp);
fflush(stdout);
if ((i%savestep)==0) {
    /********************/
    if (!run) fprintf(ifile,"−");
    fprintf(wfile,"%d.
",i);
    fprintf(wfile,"%f %f
",temp,temp2);
    fprintf(wfile,"%f %f %f
",temp2,temp+temp2,I);
    /********************/
}
if (OKp==0) {sub_dwpt();
    else {succmovep++; oldh1=h1;}
}
if ((i%savestep)==0) {
    if (!run) fprintf(ifile,"−");
    fprintf(ifile,"%d
",i);
    accrate=(double)succmove/(double)savestep;
    accratep=(double)succmovep/(double)(wptnum*savestep);
    if (accrate>(accgoal−accerr)&&accrate<(accgoal+accerr)) {
        countacc++;
        if (countacc==countw) {
            runw=1; countacc=0;
            MCpstep=MCsave; write_conf(i); // from savephi
        }
    } else {
        oldMCstep=MCstep; MCstep=(1.+accgoal*(accrate−accgoal));
    }
}
if (accrate>(accgoalp−accerrp)&&accrate<(accgoalp+accerrp)) {
    countacc++;
    if (countacc==countp) {run=1;i=0;write_conf(i);
        // from savephi
    }
} else {
    oldMCpstep=MCpstep; MCpstep=(1.+accgoalp*(accratep−accgoalp));
    } else {
        oldMCpstep=MCpstep; MCpstep=(1.+accgoalp∗(accratep−accgoalp));
        write_conf(-i); // from savephi
    }
} else {write_conf(i); // from savephi
    }
} /* main() */
/** read_in() */
void read_in() {
    int i;
    double d;
    FILE *file;
    char fname[40];
FullSim2d.c

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char *argv[]) {
  char fname[40];
  sprintf(fname, "%sIN", NAME);
  file=fopen(fname, "r");
  fgets(line, 80, file);
  sscanf(line, "%d", &MCiter);
  fgets(line, 80, file);
  sscanf(line, "%d", &runw);
  fgets(line, 80, file);
  sscanf(line, "%d", &run);
  fgets(line, 80, file);
  sscanf(line, "%s", READNAME);
  fgets(line, 80, file);
  sscanf(line, "%d", &LOAD);
  fgets(line, 80, file);
  fgets(line, 80, file);
  sscanf(line, "%d", &GC);
  fgets(line, 80, file);
  sscanf(line, "%lf", &zCP);
  fgets(line, 80, file);
  sscanf(line, "%lf", &zA);
  fgets(line, 80, file);
  sscanf(line, "%lf", &zB);
  fgets(line, 80, file);
  fgets(line, 80, file);
  sscanf(line, "%lf", &ph);
  fgets(line, 80, file);
  sscanf(line, "%d", &CP);
  fgets(line, 80, file);
  sscanf(line, "%lf", &f);
  fgets(line, 80, file);
  sscanf(line, "%d", &HA);
  fgets(line, 80, file);
  sscanf(line, "%d", &HB);
  fgets(line, 80, file);
  fgets(line, 80, file);
  sscanf(line, "%lf", &NCP);
  fgets(line, 80, file);
  sscanf(line, "%lf", &NA);
  fgets(line, 80, file);
  sscanf(line, "%lf", &NB);
  fgets(line, 80, file);
  fgets(line, 80, file);
  sscanf(line, "%lf", &MCstep);
  fgets(line, 80, file);
  sscanf(line, "%lf", &MCpstep);
  fgets(line, 80, file);
  sscanf(line, "%lf", &chiN);
  fgets(line, 80, file);
  sscanf(line, "%lf", &xsize);
  fgets(line, 80, file);
  sscanf(line, "%lf", &ysize);
  fgets(line, 80, file);
  sscanf(line, "%d", &C);
  fgets(line, 80, file);
  sscanf(line, "%d", &xmax);
  fgets(line, 80, file);
  sscanf(line, "%d", &ymax);
  fgets(line, 80, file);
  sscanf(line, "%d", &tmax);

  // Lutheran
  int me = 0;
  if (argc==1) {
    printf("No filename given! Exit.
    return 0;
  } else {
    if (argc==3) sprintf(type, "%s", argv[2]);
    else sprintf(type, "%s");
    sprintf(fname, "%s.IN", argv[1]);
    file = fopen(fname, type);
    scanf(fname, "%s", &NAME);
    if (argc==3) sprintf(type, "%s", argv[2]);
    else sprintf(type, "%s");
    sprintf(fname, "%s", argv[1]);
  }
  return 0;
}

FullSim2d.c

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char *argv[]) {
  char fname[40];
  sprintf(fname, "%sIN", NAME);
  file=fopen(fname, "r");
  fgets(line, 80, file);
  sscanf(line, "%d", &MCiter);
  fgets(line, 80, file);
  sscanf(line, "%d", &runw);
  fgets(line, 80, file);
  sscanf(line, "%d", &run);
  fgets(line, 80, file);
  sscanf(line, "%s", READNAME);
  fgets(line, 80, file);
  sscanf(line, "%d", &LOAD);
  fgets(line, 80, file);
  fgets(line, 80, file);
  sscanf(line, "%d", &GC);
  fgets(line, 80, file);
  sscanf(line, "%lf", &zCP);
  fgets(line, 80, file);
  sscanf(line, "%lf", &zA);
  fgets(line, 80, file);
  sscanf(line, "%lf", &zB);
  fgets(line, 80, file);
  fgets(line, 80, file);
  sscanf(line, "%lf", &ph);
  fgets(line, 80, file);
  sscanf(line, "%d", &CP);
  fgets(line, 80, file);
  sscanf(line, "%lf", &f);
  fgets(line, 80, file);
  sscanf(line, "%d", &HA);
  fgets(line, 80, file);
  sscanf(line, "%d", &HB);
  fgets(line, 80, file);
  fgets(line, 80, file);
  sscanf(line, "%lf", &NCP);
  fgets(line, 80, file);
  sscanf(line, "%lf", &NA);
  fgets(line, 80, file);
  sscanf(line, "%lf", &NB);
  fgets(line, 80, file);
  fgets(line, 80, file);
  sscanf(line, "%lf", &MCstep);
  fgets(line, 80, file);
  sscanf(line, "%lf", &MCpstep);
  fgets(line, 80, file);
  sscanf(line, "%lf", &chiN);
  fgets(line, 80, file);
  sscanf(line, "%lf", &xsize);
  fgets(line, 80, file);
  sscanf(line, "%lf", &ysize);
  fgets(line, 80, file);
  sscanf(line, "%d", &C);
  fgets(line, 80, file);
  sscanf(line, "%d", &xmax);
  fgets(line, 80, file);
  sscanf(line, "%d", &ymax);
  fgets(line, 80, file);
  sscanf(line, "%d", &tmax);

  // Lutheran
  int me = 0;
  if (argc==1) {
    printf("No filename given! Exit.
    return 0;
  } else {
    if (argc==3) sprintf(type, "%s", argv[2]);
    else sprintf(type, "%s");
    sprintf(fname, "%s.IN", argv[1]);
    file = fopen(fname, type);
    scanf(fname, "%s", &NAME);
    if (argc==3) sprintf(type, "%s", argv[2]);
    else sprintf(type, "%s");
    sprintf(fname, "%s", argv[1]);
  }
  return 0;
}
```c
#define PI M_PI

void U_2D(int nx, int ny, int **lattice1, int **lattice2, double *U) {
  int i,x,y,c;
  int *xp = ivector(1,nx);
  int *xm = ivector(1,nx);
  int *yp = ivector(1,ny);
  int *ym = ivector(1,ny);
  float *l = vector(0,256);
  int *l2 = ivector(0,255);
  /******************************************************************//*  **** Initialize lists ****/  
  for (x=1;x<=nx−1;x++)
    xp[x]=x+1;  xp[nx]=1;  xm[1]=nx;
  for (x=2;x<=nx;x++)
    xm[x]=x−1;
  for (y=1;y<=ny−1;y++)
    yp[y]=y+1;  yp[ny]=1;  ym[1]=ny;
  for (y=2;y<=ny;y++)
    ym[y]=y−1;
  for (i=0;i<256;i++)
    l[i]=0;
  l[12]=sqrt(2.);
  /******************************************************************//*  **** Initialize lists ****/  
  for (x=1;x<nx-1;x++)
    xp[x]=x+1;
  for (x=2;x<nx;x++)
    xp[nx]=1;
  for (y=1;y<ny-1;y++)
    yp[y]=y+1;
  for (y=2;y<ny;y++)
    yp[ny]=1;
  for (i=0;i<256;i++)
    l2[i]=0;
  l2[12]=sqrt(2.);
  /******************************************************************//*  **** Initialize lists ****/  
  int l,x,y,c;
  int *xp = ivector(1,nx);
  int *xm = ivector(1,nx);
  int *yp = ivector(1,ny);
  int *ym = ivector(1,ny);
  float *l = vector(0,255);
  int *l2 = ivector(0,255);
  return (0);
}
```
```c
/* oxo */
l[66]=2.;  l[24]=2.;
/* oxo */
l[5]=2.*sqrt(2.);
l[132]=2.*sqrt(2.);
l[160]=2.*sqrt(2.);
l[33]=2.*sqrt(2.);
/* oxo */
l[18]=sqrt(2.);
l[80]=sqrt(2.);
l[72]=sqrt(2.);
l[10]=sqrt(2.);
/* oxo */
l[48]=sqrt(5.);
l[17]=sqrt(5.);
l[12]=sqrt(5.);
l[136]=sqrt(5.);
l[165]=sqrt(5.);
l[168]=sqrt(5.);
l[134]=sqrt(5.);
l[130]=sqrt(5.);
/* oxo */
l[37]=3.*sqrt(2.);
l[133]=3.*sqrt(2.);
l[164]=3.*sqrt(2.);
l[161]=3.*sqrt(2.);
/* oxo */
l[82]=2.*sqrt(2.);
l[88]=2.*sqrt(2.);
l[74]=2.*sqrt(2.);
l[26]=2.*sqrt(2.);
/* oxo */
l[165]=4.*sqrt(2.);
/* oxo */
l[90]=4.;
/* oxo */
l[76]=2.*sqrt(2.);
l[138]=2.*sqrt(2.);
l[50]=2.*sqrt(2.);
l[81]=2.*sqrt(2.);
/* oxo */
l[69]=1.*2.*sqrt(2.);
l[140]=1.*2.*sqrt(2.);
l[162]=1.*2.*sqrt(2.);
l[49]=1.*2.*sqrt(2.);

/**** Make circumference bitmap ****/
for(x=1;x<=nx;x++)
  for(y=1;y<=ny;y++) {
    if (lattice1[x][y]) {
      if (((!lattice1[xp[x]][y]) ||
            (!lattice1[xm[x]][y]) ||
            (!lattice1[x][yp[y]]) ||
            (!lattice1[x][ym[y]]))
        lattice2[x][y]=1;
      else
        lattice2[x][y]=0;
    }
  }

/**** Measure circumference ****/
*U=0.;
for(i=0;i<255;i++) l2[i]=0;
for(y=1;y<ny;y++)
  for(x=1;x<nx;x++)
    if (lattice2[x][y]) {
      if (((lattice2[xm[x]][ym[y]] +
            2*lattice2[x][ym[y]] +
            4*lattice2[xp[x]][ym[y]] +
            8*lattice2[x][yp[y]] +
            16*lattice2[xm[x]][yp[y]] +
            32*lattice2[2*xm[x]][yp[y]] +
            64*lattice2[x][yp[y]] +
            128*lattice2[x][yp[y]]));
        if ((c[i])==3) c[i]=2;
        if ((c[i])==9) c[i]=8;
        if ((c[i])==40) c[i]=4;
        if ((c[i])==96) c[i]=64;
        if ((c[i])==192) c[i]=64;
        if ((c[i])==192) c[i]=16;
        if ((c[i])==20) c[i]=16;
        if ((c[i])==56) c[i]=2;
```
```c
U^2d.c

#include <stdio.h>
#include <math.h>

#define nx 256
#define ny 256

int n;
int **lattice;
int *phivec;

int lmax(x0, y0, x1, y1) {
    int i, imax, ix, iy, phi, max, l, c;
    float m, mx, my, mx, mx;
    imax=(int)sqrt((float)((x0-x1)*(x0-x1)+(y0-y1)*(y0-y1)));
    if (x1!=x0) {
        m=(float)(y1-y0)/((float)(x1-x0));
        mx=sqrt(1./(1.+m*m));
        my=sqrt(m*m/(1.+m*m));
    } else {
        my=1.;
        mx=0.;
    }
    fx=(float)x0+.5;
    fy=(float)y0+.5;
    for (i=0;i<imax;i++) {
        ix=(int)fx;
        iy=(int)fy;
        phi=lattice[ix][iy];
        phivec[i]=phi;
        phivec[i+imax]=phi;
        mx+=mx;
        fy+=my;
    }
    l=1;
    max=1;
    c=phivec[0];
    for (i=1;i<2*imax;i++) {
        if (phivec[i]==c) l++;
        else {
            c=1-c;
            if (l>max) max=l;
            l=1;
        }
        if (i==2*imax) return max;
    }
}

void L2D(int an, int **alattice, float *L, float *Lmax) {
    int i, x, y, p;
    ...
}
```
float l1, l2, r, rmax;

n=an;  lattice=imatrix(0,2*n-1,0,2*n-1);
for (p=1; p<=n; p++)
  for (y=1; y<=n; y++) {
    lattice[x-1][y-1]=lattice[x][y];
    lattice[x-1+n][y-1]=lattice[x][y];
    lattice[x-1][y-1+n]=lattice[x][y];
    lattice[x-1+n][y-1+n]=lattice[x][y];
  }
phivec=ivector(0,200);
r=0.;  rmax=0.;
for (x=0; x<=n; x++) {
  l1=0.;
  for (p=0; p<n; p++)
    l1+=lmax(p,n,x+p,0);
  l1/=(float)n;
  l2=0.;
  for (p=0; p<n; p++)
    l2+=lmax(0,p,n,x+p);
  l2/=(float)n;
  r+=l1/l2+l2/l1;
  if ((l1/l2)>rmax) rmax=l1/l2;
  if ((l2/l1)>rmax) rmax=l2/l1;
}
for (x=0; x<=n; x++) {
  l1=0.;
  for (p=0; p<n; p++)
    l1+=lmax(0,n+p,n,p);
  l1/=(float)n;
  l2=0.;
  for (p=0; p<n; p++)
    l2+=lmax(p,0,n−x+p,n);
  l2/=(float)n;
  r+=l1/l2+l2/l1;
  if ((l1/l2)>rmax) rmax=l1/l2;
  if ((l2/l1)>rmax) rmax=l2/l1;
}
r/=(float)(4*(n+1));
free_imatrix(lattice,0,2*n−1,0,2*n−1);
free_ivector(phivec,0,200);
*L=r;
*Lmax=rmax;
F2.c

```c
void integral(float *bin, double *I) {
    int i;
    double c;
    double max;
    int maxk;
    double sd, av, av2;
    c = 0.;
    for (i = 1; i < 50; i++) {
        c += bin[i];
    }
    max = 0.;
    maxk = 0;
    c = 0.;
    for (i = 1; i < 50; i++) {
        if (((double)bin[i]) > max) {
            max = (double) bin[i];
            maxk = i;
        }
        c += bin[i];
    }
    sd = 0.;
    av = 0.;
    av2 = 0.;
    for (i = 1; i < 50; i++) {
        av += ((double)bin[i]) / c;
        av2 += ((double)bin[i]) * ((double)bin[i]) / c;
    }
    *I = c / sd;
}
```

F2.c

```c
while (fread(array, sizeof(float), xmax*ymax*6, file) != EOF) {
    change_endian(array, xmax*ymax*6, file);
    i = 0;
    k = 0;
    for (x = 1; x <= xmax; x++) {
        for (y = 1; y <= ymax; y++) {
            data[k++] = (double)array[i];
            i += sizeof(float);
        }
    }
```

F2.c

```c
main(int argc, char **argv) {
    int i, done = 0; /* control variable */
    int k, j, me, my_x, my_y, x, y;
    int **phiA;
    int a, jx, jy;
    float array[24576];
    float *fp;
    unsigned char c[4];
    unsigned long nn[3];
    double data[8192];
    float dx, dy, dlat;
    float bin[100], bini[100], binn[100];
    int count[100];
    double phi;
    float sq2;
    int MCS;
    double t1;
    printf(NAME, "Gm", argv[1]);
    read_in();
    p = fftw2d_create_plan(nx, ny, FFTW_FORWARD,
                           FFTW_ESTIMATE | FFTW_IN_PLACE);
    pinv = fftw2d_create_plan(nx, ny, FFTW_BACKWARD,
                              FFTW_ESTIMATE | FFTW_IN_PLACE);
    fp = p;
    sprintf(fwname, "%s.s2", NAME);
    filew = fopen(fwname, "w");
    MCS = 0;
    while (fread(array, sizeof(float), xmax*ymax*6, file) != EOF) {
        change_endian(array, xmax*ymax*6, file);
        MCS += savestep;
        i = 0;
        for (x = 1; x <= xmax; x++) {
            for (y = 1; y <= ymax; y++) {
                data[k++] = (double)array[i];
                i += sizeof(float);
            }
        }
    }
    return 0;
}
```
F2.c

```c
data[k++] = 0.;
i += 6;
}
fftwnd_one(p, (fftw_complex *) data, NULL);
for (i = 0; i < 100; i++) {
    bin[i] = 0.;
    bini[i] = 0.;
    count[i] = 0;
}
i = 0;
k = 0;
for (x = 1; x <= xmax; x++)
    for (y = 1; y <= ymax; y++) {
        dr = data[k++];
        di = data[k++];
        *fp = sqrt(dr * dr + di * di);
        sq2 = dr * dr + di * di;
        for (j = 0; j < 4; j++) {
            array[j] = c[3 - j];
        }
        jx = x;
        jy = y;
        if (jx > xmax / 2) jx -= xmax;
        if (jy > ymax / 2) jy -= ymax;
        if (jx < 0) jx += xmax;
        if (jy < 0) jy += ymax;
        f = sqrt((jx * jx + jy * jy));
        dist = sqrt(f);
        a = rint(dist);
        if (a > 99) a = 99;
        if (jx)
            phi = atan2((double) jy, (double) jx);
        else if (jy > 0) phi = PI / 2.;
        else if (jy < 0) phi = -PI / 2.;
        if (jx < 0) phi += PI;
        phi *= 2.;
        if (a)
            if (a < 100) {
                bin[a] += sq2 * cos(phi);
                bini[a] += sq2 * sin(phi);
                count[a]++;
            }
    }
for (i = 1; i < 50; i++) {
    if (count[i]) bin[i] /= (float) count[i];
    bin[i] = sqrt(bin[i] * bini[i] + bini[i] * bini[i]);
}
integral(bin, I);
printf("%d %f\n", MCS, I);
} // end while
fclose(file);
fclose(file);
return 0;
}
```

F2.c

```c
if (count[i]) bin[i] /= (float) count[i];
bin[i] = sqrt(bin[i] * bin[i] + bini[i] * bini[i]);
}
integral(bin, I);
printf("%d %f\n", MCS, I);
} // end while
fclose(file);
fclose(file);
return 0;
```
159

Calculates \( \frac{1}{L_c} \int ds |\frac{dt}{ds}|^2 \)

(essential quantity to calculate the curvature diameter)

# define PI 3.1415926535897932

```c
void t_2D(int nx, int ny, int ***lattice1, int ***lattice2, int ***f, double *T) {
    int i, x, y, c, ix, iy, k;
    int *xp = ivector(1, nx);
    int *xm = ivector(1, nx);
    int *yp = ivector(1, ny);
    int *ym = ivector(1, ny);
    float *l = vector(0, nx * ny);
    int *l2 = ivector(0, nx * ny);
    int border_pixels, j, found;
    int x1, y1, x2, y2;
    double u1x, u1y, u2x, u2y, dx, dy, ds, s;
    double a, b;
    int i, j, x, y;
    /**** Initialize lists ****/
    for (x = 1; x <= nx - 1; x++)
        xp[x] = x + 1;  xp[nx] = 1;
    for (x = 2; x <= nx; x++)
        xm[x] = x - 1;
    for (y = 1; y <= ny - 1; y++)
        yp[y] = y + 1;  yp[ny] = 1;
    for (y = 2; y <= ny; y++)
        ym[y] = y - 1;
    /**** Make circumference bitmap ****/
    border_pixels = 0;
    for (x = 1; x <= nx; x++)
        for (y = 1; y <= ny; y++)
            if (lattice1[x][y]) {
                if (!lattice1[xp[x]][y])
                    if (!lattice1[xm[x]][y])
                        if (!lattice1[x][yp[y]])
                            lattice2[x][y] = 1;
                            border_pixels++;
                            if (lattice1[x][ym[y]])
                                border_pixels++;
                            else lattice2[x][y] = 0;
                        }
                        else lattice2[x][y] = 0;
                    }
                    else lattice2[x][y] = 0;
                }
                i = 0;
                j = 0;
            }
            while (border_pixels) {
                if (j == 0) {
                    / * Find new seed point for f[i] */
                    for (i = 1; i <= nx; i++)
                        for (j = 1; j <= ny; j++)
                            if (lattice2[i][j]) {
                                x = i;
                                y = j;
                                if (!lattice1[i][j])
                                    if (!lattice1[xp[i]][j])
                                        if (!lattice1[xm[i]][j])
                                            if (!lattice1[i][yp[j]])
                                                lattice2[i][j] = 0;
                                                border_pixels--;
                                                i = nx + 1;
                                                j = ny + 1;
                                            }
                                            else lattice2[i][j] = 0;
                                        }
                                        else lattice2[i][j] = 0;
                                    }
                                    if (found) {
                                        found = 0;
                                        if (!found && lattice2[xp[i]][yp[y]])
                                            if (lattice1[i][yp[j]] && lattice1[xp[i]][y])
                                                if (found)
                                                    border_pixels++;
                                                    found = 1;
                                                    iy = yp[j];
                                                    iy = yp[j];
                                                }
                                                else if (!found && lattice2[xp[i]][yp[y]])
                                                    if (lattice1[i][yp[j]] && lattice1[xp[i]][ym[y]])
                                                        if (found)
                                                            border_pixels++;
                                                            found = 1;
                                                            iy = yp[j];
                                                            iy = yp[j];
                                                    }
                                                    else if (!found && lattice2[xp[i]][ym[y]])
                                                        if (lattice1[i][ym[y]] && lattice1[xp[i]][ym[y]])
                                                            if (found)
                                                                border_pixels++;
                                                                found = 1;
                                                                iy = yp[j];
                                                                iy = yp[j];
                                                            }
                                                            else if (!found && lattice2[xm[i]][ym[y]])
                                                                if (lattice1[i][ym[y]] && lattice1[xm[i]][ym[y]])
                                                                    if (found)
                                                                        border_pixels++;
                                                                        found = 1;
                                                                        iy = yp[j];
                                                                        iy = yp[j];
                                                                    }
                                                                    else if (!found && lattice2[xm[i]][yp[j]])
                                                                        if (lattice1[i][ym[y]] && lattice1[xm[i]][yp[j]])
                                                                            if (found)
                                                                                border_pixels++;
                                                                                found = 1;
                                                                                iy = yp[j];
                                                                                iy = yp[j];
                                                                            }
                                                                            else if (!found && lattice2[xm[i]][yp[j]])
                                                                                if (lattice1[i][ym[y]] && lattice1[xm[i]][yp[j]])
                                                                                    if (found)
                                                                                        border_pixels++;
                                                                                        found = 1;
                                                                                        iy = yp[j];
                                                                                        iy = yp[j];
                                                                                }
```
```c
if (!found && (lattice2[x][yp[y]])) {
    if (!((lattice1[xm[x]][y]) && (lattice1[x][yp[y]]))) {
        found=1;
        ix=xm[x];
        iy=yp[y];
    }
}
if (!found && (lattice2[x][yp[y]])) {
    if (!((lattice1[xm[x]][yp[y]]) && (lattice1[xp[x]][yp[y]]))) {
        found=1;
        ix=x;
        iy=yp[y];
    }
}
if (found) {
    lattice2[x][y]=0;
    border_pixels--;
    y=iy;
    f[1][1][0]=x;
    f[1][1][1]=y;
    j+2;
} else {
    i[i]=j;
    j=0;
    i++;}
}

l[i]=0;
l=0;
s=0.;
*T=0.;
while (l[i]) {
    if (l[i]>3) {
        for (j=1;j<l[i]-2;j++) {
            u1x=(double)(f[i][j+1][0]-f[i][j-1][0]);
            u1y=(double)(f[i][j+1][1]-f[i][j-1][1]);
            u2x=(double)(f[i][j+2][0]-f[i][j+1][0]);
            u2y=(double)(f[i][j+2][1]-f[i][j+1][1]);
            a=sqrt(u1x*u1x+u1y*u1y);
            u1x/=a;
            u1y/=a;
            a=sqrt(u2x*u2x+u2y*u2y);
            u2x/=a;
            u2y/=a;
            a=u2x-u1x;
            b=u2y-u1y;
            dx=(double)(f[i][j+2][0]-f[i][j+1][0]-f[i][j][0]-f[i][j-1][0])/2.8;
            dy=(double)(f[i][j+2][1]-f[i][j+1][1]-f[i][j][1]-f[i][j-1][1])/2.8;
            ds=sqrt(dx*dx+dy*dy);
            t2d.c
            if (ds) {
                s += ds;  *T += (a*a+b*b)/ds;
            }
            i++;
            *T/=s;
        }
        free_vector(xp,1,nx);
        free_vector(xm,1,nx);
        free_vector(yp,1,ny);
        free_vector(ym,1,ny);
        free_vector(l,0,255);
        free_vector(l2,0,255);
    }
    i++;}
*T/=s;
free_vector(xp,1,nx);
free_vector(xm,1,nx);
free_vector(yp,1,ny);
free_vector(ym,1,ny);
free_vector(l,0,255);
free_vector(l2,0,255);
```
```c
#define PI (3.1415926535897932)
#define WISH_GRID_SIZE (0.1)
#define SURFACE_FACE (1)
#define PERIMETER_FACE (−4)
#define EULER2D_FACE (1)
#define EULER2D_EDGE (−1)
#define EULER2D_VERTEX (1)
#define STEP2D_AREA (0.)
#define STEP2D_PERI (0.)

void Minkowski_2D(int nx, int ny, int **lattice, double *area, double *perimeter, int *euler, int *step) {
    int i, j;
    int nedges, nvert, natags;
    int jx, jy, jxi, jyi, k4, kcl, a, b;
    int eu1, stp;
    double sur, cur;
    int **temp;

    sur = 0;
    cur = 0;
    eu1 = 0;
    stp = 0;

    temp = imatrix(0, nx+1, 0, ny+1);
    for (i = 0; i <= nx+1; i++)
        for (j = 0; j <= ny+1; j++)
            temp[i][j] = 0;

    for (jx = 1; jx <= nx; jx++)
        for (jy = 1; jy <= ny; jy++) {
            if (lattice[jx][jy]) {
                nedges = 0;
                nvert = 0;
                natags = 0;

                for (i = −1; i <= 3; i++)
                    for (j = −1; j <= 3; j++) {
                        j4 = jy + j;
                        jyi = jy + i;
                        kcl = 1−temp[jxi][jyi];
                        nedges += kcl+1−temp[jx][jy+1];

                        for (j4 = jy + j;
                             j4 <= ny;
                             j4++)
                            if (lattice[jx][j4]) {
                                nvert = 1;
                                natags = 1;
                            }

                        if (jxi == 1 && jyi == 1) {  // Check boundary conditions
                            temp[nx+1][jy] = 1;
                            if (jx == nx) temp[0][jy] = 1;
                            if (jy == ny) temp[jx][0] = 1;
                            if (jx == nx && jy == ny) temp[0][0] = 1;
                            if (jx == 1 && jy == ny) temp[nx+1][0] = 1;
                            if (jx == nx && jy == 1) temp[0][ny+1] = 1;
                        }
                    }

                m2d.c
```

```c
#define PI (3.1415926535897932)
#define WISH_GRID_SIZE (0.1)
#define SURFACE_FACE (1)
#define PERIMETER_FACE (−4)
#define EULER2D_FACE (1)
#define EULER2D_EDGE (−1)
#define EULER2D_VERTEX (1)
#define STEP2D_AREA (0.)
#define STEP2D_PERI (0.)

void Minkowski_2D(int nx, int ny, int **lattice, double *area, double *perimeter, int *euler, int *step) {
    int i, j;
    int nedges, nvert, natags;
    int jx, jy, jxi, jyi, k4, kcl, a, b;
    int eu1, stp;
    double sur, cur;
    int **temp;

    sur = 0;
    cur = 0;
    eu1 = 0;
    stp = 0;

    temp = imatrix(0, nx+1, 0, ny+1);
    for (i = 0; i <= nx+1; i++)
        for (j = 0; j <= ny+1; j++)
            temp[i][j] = 0;

    for (jx = 1; jx <= nx; jx++)
        for (jy = 1; jy <= ny; jy++) {
            if (lattice[jx][jy]) {
                nedges = 0;
                nvert = 0;
                natags = 0;

                for (i = −1; i <= 3; i++)
                    for (j = −1; j <= 3; j++) {
                        j4 = jy + j;
                        jyi = jy + i;
                        kcl = 1−temp[jxi][jyi];
                        nedges += kcl+1−temp[jx][jy+1];

                        for (j4 = jy + j;
                             j4 <= ny;
                             j4++)
                            if (lattice[jx][j4]) {
                                nvert = 1;
                                natags = 1;
                            }

                        if (jxi == 1 && jyi == 1) {  // Check boundary conditions
                            temp[nx+1][jy] = 1;
                            if (jx == nx) temp[0][jy] = 1;
                            if (jy == ny) temp[jx][0] = 1;
                            if (jx == nx && jy == ny) temp[0][0] = 1;
                            if (jx == 1 && jy == ny) temp[nx+1][0] = 1;
                            if (jx == nx && jy == 1) temp[0][ny+1] = 1;
                        }
                    }

                m2d.c
```
```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "nrutil.c"
#include <rfftw.h>
#include <fftw.h>
#define nx xmax
#define ny ymax
#define PI M_PI
fftwnd_plan p, pinv;
double xsize, ysize;
int xmax, ymax;
char frname[40], fwname[40], line[160], NAME[40];
float ***w, **w_av;
float *array;
float **corr;
FILE *filer;
FILE *filew;
float bin[100];
unsigned long nn[3];
double *data;
float dr, di, dist;
double chlN;
int from, to;

void read_in() {

    int i, j;
    double d;
    FILE* file;
    char ffname[40], line[80];
    int xnum, ynum;
    sprintf(ffname, "%s.in", NAME);
    file=fopen(ffname, "r");
    for (i=0; i<22; i++)
        fgets(line, 80, file);
    sscanf(line, "%d", &xmax);
    fgets(line, 80, file);
    sscanf(line, "%d", &xnum);
    fgets(line, 80, file);
    sscanf(line, "%d", &ymax);
    fgets(line, 80, file);
    sscanf(line, "%d", &ynum);
    for (i=0; i<9; i++)
        fgets(line, 80, file);
    fclose(file);
xmax=xnum;
ynum=ynum;
}

void change_endian(unsigned char *array, int j) {
    int i;
    unsigned char c;
    for (i=0; i<j; i++) {
        c=array[i];    array[i]=array[i+3];    array[i+3]=c;    array+=4;
    }
}

void init() {
    read_in();
    array=vector(0, xmax*ymax*5-1);
    w=f3tensor(1-xmax, 2*xmax, 1-ymax, 2*ymax, 1, to);
    w_av=matrix(1-xmax, 2*xmax, 1-ymax, 2*ymax);
    corr=dmatrix(1, xmax, 1, ymax);
    data=dvector(0, xmax*ymax*5-1);
}

void close() {
    free_vector(array, 0, xmax*ymax*6-1);
    free_f3tensor(w, 1-xmax, 2*xmax, 1-ymax, 2*ymax, 1, to);
    free_matrix(w_av, 1-xmax, 2*xmax, 1-ymax, 2*ymax);
    free_dmatrix(corr, 1, xmax, 1, ymax);
    free_dvector(data, 0, xmax*ymax*5-1);
}

int main(int argc, char **argv) {
```
## sfac2d.c

```c
int i, j, imax, jm, jy, ix, jy, q, d1, d2; double d2x, d2y, ia;
float a, p;
float x, f, dist, binmax;
double C, fd;
sprintf(NAME, "%s", argv[1]);
from = atof(argv[2]); to = atof(argv[3]); init();
p = fftw2d_create_plan(nx, ny, FFTW_FORWARD, FFTW_ESTIMATE | FFTW_IN_PLACE);
pinv = fftw2d_create_plan(nx, ny, FFTW_BACKWARD, FFTW_ESTIMATE | FFTW_IN_PLACE);
i = 0;
sprintf(fname, "%s.cnf", NAME);
filer = fopen(fname, "r");
while (i < to - 1) {
    fread(array, sizeof(float), xmax * ymax * 6, filer);
    printf("reading %d\n", i);
    fflush(stdout);
    i++;
    change_endian(array, xmax * ymax * 6);
    j = 0;
    for (x = 1; x <= xmax; x++)
        for (y = 1; y <= ymax; y++) {
            a = array[j + 2] + array[j + 3];
            w[x][y][i] = a;
            w[x - xmax][y][i] = a;
            w[x + xmax][y][i] = a;
            w[x][y - ymax][i] = a;
            w[x - xmax][y - ymax][i] = a;
            w[x][y + ymax][i] = a;
            w[x - xmax][y + ymax][i] = a;
            // printf("%.16f \n", a);
            fflush(stdout);
        }
    j += 6;
}
for (i = 1; i < xmax; x++)
    for (j = 1; j < ymax; y++)
        a = 0.0;
    for (i = from; i < to; i++)
        a /= (double) (to - from);
    w_av[x][y] = a;
    w_av[x][ymax] = a;
    w_av[xmax][y] = a;
    w_av[x][ymax] = a;
    w_av[x][ym] = a;
    w_av[xmax][ym] = a;
    w_av[xmax][ym] = a;
    w_av[x][ym] = a;
    w_av[x][ym] = a;
    w_av[x][ym] = a;
    w_av[x][ym] = a;
    w_av[x][ym] = a;
    w_av[x][ym] = a;
    w_av[x][ym] = a;
    w_av[x][ym] = a;
    w_av[x][ym] = a;
    w_av[x][ym] = a;
    w_av[x][ym] = a;
    w_av[x][ym] = a;
    w_av[x][ym] = a;
    w_av[x][ym] = a;
    w_av[x][ym] = a;
    w_av[x][ym] = a;
    w_av[x][ym] = a;
    w_av[x][ym] = a;
    w_av[x][ym] = a;
    w_av[x][ym] = a;
```
```c
jx=x+xmax/2;  
jy=y+ymax/2;
if (jx>xmax)  jx-=xmax;
if (jy>ymax)  jy-=ymax;
corr[jx][jy]=fd;
}
sprintf(fwname,"%s.sf",NAME);
filew = fopen(fwname,"w");
for (x=1;x<=xmax;x++)
  for (y=1;y<=ymax;y++)
    fprintf(filew,"%.d %.d %.f",x,y,corr[x][y]);
fclose(filew);
close();
return 0;
```
Bibliography


Acknowledgments/Danksagung

I wish to thank: / Ich bedanke mich bei:

An erster Stelle möchte ich mich ganz herzlich bei Prof. Dr. Friederike Schmid bedanken: für die interessante Themenstellung, für die stets hervorragende Betreuung trotz schwerer Krankheit und für das angenehme Arbeitsklima. Und nicht zuletzt für die Unterstützung meines Wunsches, einen Teil der Promotionszeit im Ausland zu verbringen.

Prof. Dr. Don E. Sullivan, whom I had the pleasure visiting at the U of Guelph, Ontario, during a seven-month period in 2001 and who helped make my stay there a worthwhile experience, both academically and privately. I am also grateful for a stipend to help cover my living expenses from the U of Guelph and for the efforts in connection with our paper on rod-coil polymers.

Prof. Dr. Glenn H. Fredrickson at Materials Research Lab/UC Santa Barbara and Prof. Dr. Venkat Ganesan at UT Austin, with whom I have enjoyed a very fruitful collaboration on the fluctuation problem (they used a complex Langevin method complementary to my Monte Carlo method). Thank you for hosting me during visits in your groups (including the financial support), for the many fruitful discussions, and for telling me about the pseudo-spectral method which has so helped to bring this work to a successful conclusion. I also wish to thank Dr. Scott W. Sides, Dr. André Moreira, Deena Patel, and Alfredo Katz in Santa Barbara for interesting exchanges.

I am grateful to Prof. Dr. Mark W. Matsen (U of Reading, England) for an exchange on the revision of the mean-field phase diagram for ternary blends.


Die Gewährung von Rechenzeit am Forschungszentrum Jülich hat, wie die Förderung durch die Deutsche Forschungsgemeinschaft und die an der Universität Bielefeld zur Verfügung stehenden Computer, entscheidend zum Erfolg dieser Arbeit beigetragen.

Schließlich danke ich meinen Eltern für die Unterstützung während des gesamten Studiums.
Lebenslauf

Persönliche Daten:
Name: Dominik Düchs
Geburtsdatum: 15. April 1975
Geburtsort: Frankfurt am Main

Bildungsweg:
1981-1984 Grundschule in Neu-Isenburg und Mainz-Drais
1984-1993 Bischöfliches Willigis-Gymnasium Mainz
18.06.1993 Abitur
07/1993-06/1994 Grundwehrdienst
10/1994-12/1999 Studium der Physik an der Johannes Gutenberg-Universität Mainz
03.09.1996 Vordiplom
1997/98 Auslandsjahr an der University of Washington, Seattle (USA)
03-12/1999 Diplomarbeit bei PD Dr. F. Schmid am Max-Planck-Institut für Polymerforschung (MPIP) Mainz: “The Influence of Head Group Potentials on the Phase Behavior of Langmuir Monolayers”
28.12.1999 Diplom
03/2000-04/2000 Wissenschaftlicher Mitarbeiter am MPIP Mainz
seit 01.05.2000 Wissenschaftlicher Mitarbeiter und Doktorand an der Fakultät für Physik, Universität Bielefeld
04-11/2001 Gastwissenschaftler bei Prof. Dr. D. E. Sullivan, Dept. of Physics, University of Guelph (Ontario/Kanada)
seit 2001 Kollaboration mit Arbeitsbesuchen bei Prof. Dr. G. H. Fredrickson, Materials Research Lab/University of California Santa Barbara (USA) und Prof. Dr. V. Ganesan, Chemical Engineering Dept., University of Texas at Austin (USA)
Hiermit versichere ich, daß ich die vorliegende Arbeit selbständig erstellt und außer den angegebenen Quellen keine weiteren Hilfsmittel verwendet habe.

Dominik Düchs

Bielefeld, den 27.02.2003

Gedruckt auf alterungsbeständigem Papier © ISO 9706.