UNIVERSIDADE ESTADUAL DE CAMPINAS

Faculdade de Engenharia Química

JOYCE TAVARES LOPES

MODELING NONSPHERICAL PARTICLES: MONTE CARLO SIMULATIONS AND EQUATIONS OF STATE BASED ON PERTURBATION THEORY

MODELAGEM DE PARTÍCULAS NÃO-ESFÉRICAS: SIMULAÇÕES DE MONTE CARLO E EQUAÇÕES DE ESTADO BASEADAS EM TEORIA DE PERTURBAÇÃO

CAMPINAS

2020

JOYCE TAVARES LOPES

MODELING NONSPHERICAL PARTICLES: MONTE CARLO SIMULATIONS AND EQUATIONS OF STATE BASED ON PERTURBATION THEORY

MODELAGEM DE PARTÍCULAS NÃO-ESFÉRICAS: SIMULAÇÕES DE MONTE CARLO E EQUAÇÕES DE ESTADO BASEADAS EM TEORIA DE PERTURBAÇÃO

Tese apresentada à Faculdade de Engenharia Química da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Doutora em Engenharia Química

Thesis presented to the School of Chemical Engineering of the University of Campinas in partial fulfillment of the requirements for the degree of Doctor in Chemical Engineering

Supervisor: Luís Fernando Mercier Franco

ESTE EXEMPLAR CORRESPONDE À VERSÃO FINAL DA TESE DEFENDIDA PELA ALUNA JOYCE TAVARES LOPES, E ORIENTADA PELO PROF. DR. LUÍS FERNANDO MERCIER FRANCO

Campinas

2020

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Rose Meire da Silva - CRB 8/5974

L881m

Lopes, Joyce Tavares, 1991-
Modeling nonspherical particles: Monte Carlo simulations and equations of state based on perturbation theory / Joyce Tavares Lopes. – Campinas, SP : [s.n.], 2020.

Orientador: Luís Fernando Mercier Franco.
Tese (doutorado) – Universidade Estadual de Campinas, Faculdade de Engenharia Química.

1. Simulação molecular. 2. Equações de estado. 3. Termodinâmica estatística. 4. Cristais líquidos. I. Franco, Luís Fernando Mercier, 1988-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Química. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Modelagem de partículas não-esféricas: simulações de Monte Carlo e equações de estado baseadas em teoria de perturbação
Palavras-chave em inglês:
Molecular simulation
Equation of state
Statistical thermodynamics
Liquid crystals
Área de concentração: Engenharia Química
Titulação: Doutora em Engenharia Química
Banca examinadora:
Luís Fernando Mercier Franco [Orientador]
Achille Giacometti
Thomas Lafitte
Marcelo Castier
Jarrel Richard Elliot Jr
Data de defesa: 19-10-2020
Programa de Pós-Graduação: Engenharia Química

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0002-4313-6501
- Currículo Lattes do autor: http://lattes.cnpq.br/8754662340800612

Folha de Aprovação da Defesa de Tese de Doutorado defendida por Joyce Tavares Lopes em 19 de outubro de 2020 pela banca examinadora constituída pelos doutores:

Prof. Dr. Luís Fernando Mercier Franco
Presidente e orientador - FEQ Unicamp

Prof. Dr. Marcelo Castier
Texas A&M University at Qatar

Prof. Dr. Achille Giacometti
Università Ca' Foscari Venezia

Dr. Thomas Lafitte
Process System Enterprise

Prof. Dr. Jarrel Richard Elliot Jr.
University of Akron

A Ata da defesa com as respectivas assinaturas dos membros encontra-se no SIGA/-Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

From Him and through Him and to Him . . .

"For they are not the thing itself; they are only the scent of a flower we have not found, the echo of a tune we have not heard, news from a country we have never yet visited."

Clive Staples Lewis

ACKNOWLEDGMENTS

(in Portuguese)

Ao Deus tão real e tão presente na minha vida. A fonte das coisas perfeitas, agradáveis e que muitas vezes pareciam impossíveis, mas que por tantas vezes vivi. Por me levar além da minha própria capacidade e ser o meu refúgio nos momentos mais difíceis. A Ele agradeço também por ter colocado pessoas espetaculares no meu caminho, o que, para mim, é mais uma prova do Seu amor.

Aos meus pais, Jovina e Jorge, por desde muito cedo estabelecerem a educação como prioridade. Por ultrapassarem o limite do suficiente e proporcionarem a mim e a minha irmã muito mais além do que aquilo que se espera de bons pais. À minha irmã Juliana por todas as longas conversas, apoio, paciência e cumplicidade. Obrigada por participarem de todas as minhas conquistas e pelo suporte e encorajamento nos momentos mais desafiadores. Aos meus avós e familiares pela torcida e orações, e aos meus amigos pelas conversas e presença mesmo nas horas mais difíceis.

Ao Prof. Dr. Luís Fernando Mercier Franco agradeço primeiramente pela orientação, incentivo e pelas ideias e discussões frutíferas sobre o projeto. Sou imensamente grata a Deus por ter sido orientada por alguém com tanto conhecimento, apesar da pouca idade, brilhante e ao mesmo tempo de tão fácil trato. Sobretudo agradeço pela amizade, que espero levar para o resto da vida.

Aos meus amigos do LESC, Rodrigo, Marcelle e Isa, pela amizade, cumplicidade e pelas boas conversas regadas a bons cafés. Quanta sorte tive de encontrá-los, dividir tanto da minha rotina e vida, e tê-los como amigos. Ao meu amigo Eric e à minha primeira amiga em Campinas, Thays, pela amizade, conversas agradáveis e apoio. Guardo com muito carinho todos os momentos vividos nestes últimos cinco anos. Agradeço também a todos os outros amigos e colegas da FEQ e do LESC.

Ao Prof .Dr. Artur Zaghini Francesconi (*in memorian*) que me recebeu como orientanda tão logo cheguei à Unicamp. Queria ter tido uma última oportunidade de passar em sua sala para conversar. Sinto muito por ter nos deixado tão cedo. Ao Prof. Dr. Sávio Souza Venâncio Vianna pela coorientação durante o meu mestrado, ao Prof. Dr. Roger Zemp por ter gentilmente nos cedido espaço em seu laboratório enquanto o nosso não estava pronto, e aos demais professores do DESQ pela convivência tão agradável. Agradeço

também ao Prof. Dr. Pedro de Alcântara Pessoa Filho e ao Prof. Dr. Adalberto Bassi pelos comentários e sugestões valiosos dados nos exames de qualificação do doutorado.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001 (Número dos processos: 88882.329707/2019-01 e 88887.364732/2019-00).

(in English)

I would like to thank Professor Achille Giacometti for having me in his group at Università Ca' Foscari Venezia, for all the support during my research period in Venice and for sharing some of his vast knowledge with me. I would also like to thank Dr. Flavio Romano for the fruitful discussions and my colleagues Dr. Cedrix Dongmo and Alberto.

I would like to thank the members of the commitee, Prof. Dr. Achille Giacometti, Dr. Thomas Lafitte, Prof. Dr. Marcelo Castier, and Prof. Dr. Jarrel Richard Elliot Jr for the comments on the dissertation and the valuable discussion.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001 (file numbers: 88882.329707/2019-01 and 88887.364732/2019-00).

RESUMO

Nesta tese foram desenvolvidos modelos moleculares numéricos e teóricos para partículas anisotrópicas, como uma estratégia para descrever o comportamento termodinâmico de moléculas não-esféricas. Equações de estado moleculares baseadas em teoria de perturbação foram formuladas para partículas elipsoidais, cilíndricas e esferocilíndricas para a modelagem de fluidos de interesse comercial como dióxido de carbono, benzeno, tolueno, alcanos e compostos fluorados. As equações de estado tiveram um bom desempenho na descrição de propriedades termodinâmicas dos fluidos. Simulações moleculares de Monte Carlo foram realizadas para estudo dos impactos das aproximações teóricas no desempenho de equações de estado moleculares, e dos efeitos da escolha do potencial na predição do calor específico a volumer constante. Observou-se que o truncamento da expansão perturbativa no segundo termo impacta negativamente a predição de propriedades a temperaturas mais baixas, ao passo que a escolha de um potencial discreto tem grande influência no cálculo do calor específico a volume constante e possivelmente na estimativa do ponto crítico. Simulações de Monte Carlo para o fluido de cilindros duros foram realizadas para estudo dos limites de formação das fases de cristais líquidos. Fases isotrópica, nemática, esmética e cristalina foram observadas no caso de cilindros, e, além das isotrópicas e nemáticas, fases colunar e cubática foram encontradas no caso de discos cilíndricos. Com base no modelo de cilindros duros, modelos exploratórios foram propostos para o estudo de efeitos comumente encontrados em sistemas biológicos: adesivos para promover fases auto-organizáveis, um colar helicoidal de esferas duras para representar a repulsão entre hélices e adicionar quiralidade e um potencial cilíndrico para levar em conta os efeitos da concentração de sal na formação de fases de cristais líquidos.

ABSTRACT

Theoretical and numerical molecular models were developed for anisotropic particles as a strategy to capture the thermodynamic behavior of nonspherical molecules. Molecular equations of state for ellipsoidal, cylindrical, and spherocylindrical particles were formulated to predict properties of industrial relevant fluids such as carbon dioxide, benzene, toluene, *n*-alkanes, and *n*-perfluoroalkanes. Monte Carlo molecular simulations were carried out to investigate the impacts of theoretical approximations made in the development of equations of state and the effects of the choice of the intermolecular potential on the prediction of the isochoric specific heat. The truncation of the perturbative expansion on the second-order term has a negative impact on the performance of the equations of state at low temperatures, while the application of a discrete potential affects the calculation of the isochoric specific heat and possibly the critical point prediction. Monte Carlo simulations of hard cylinders were carried out to investigate the boundaries of liquid crystalline phases formation. Isotropic, nematic, smectic, and crystalline phases were observed in a system of cylindrical rods, and isotropic, nematic, cubatic, and columnar phases were found in a system of cylindrical disks. Based on the hard cylinder model, exploratory models were proposed to study effects commonly found in biological systems: attractive patches to promote self-assembly phases, a helical array of hard beads to represent the repulsion between helices and to add chirality, and a cylindrical Yukawa potential to take into consideration the effects of salt molality in the formation of liquid crystalline phases.

List of Figures

List of Tables

Summary

"Likewise, the scientist asks not what are the currently most important question, but 'which are at present solvable?' or sometimes merely 'in which can we make some small but genuine advance?"'

— Ludwig Boltzmann

1 Introduction

Admitting the existence of unseen entities seems perhaps more religious than scientific. Without immediate sensory perception, however, Boltzmann and Maxwell recognized the existence of atoms and molecules in the late nineteenth century, and this was crucial to the authors' major contributions to physics. Being at that time a controversial issue, this idea drew huge criticism from well-established scientists such as Ernst Mach. The atomic hypothesis would only begin to be accepted by the scientific community shortly after Boltzmann tragic death, when scientists as Planck and Einstein endorsed the author's methods. It is fascinating that, against the common sense of the intellectuals of the time, the bold belief in the existence of bodies too small to excite our senses led Boltzmann, Maxwell, and Gibbs to lay the foundations of statistical mechanics.

Over the last century, statistical mechanics has been established as one of the pillars of modern physics. The advances in Thermodynamics, "an incomplete expression of the principles of statistical mechanics" in Gibbs' (1902) words, owe much to the development of this new branch of physics. Statistical thermodynamics has allowed the prediction of the behavior of matter from the calculation of the interactions between its atoms and molecules, and hence has become a strong theoretical framework to understand a broad variety of systems: from fluids of industrial interest to biological systems.

Molecular simulation, a family of computational methods to solve complex statistical mechanics calculations, is a modern key tool in thermodynamics. Frequently referred to as "computational experiments", its place has been consolidated in science besides the traditional theoretical and experimental analysis. Molecular simulation results have been increasingly gaining attention from the scientific community, and it is already widely used to guide experimental work. Computational experiments are very useful to predict properties and the behavior of systems at a wide range of conditions that would be arduous, or even impossible, to reproduce experimentally. Monte Carlo and Molecular Dynamics simulations have been extensively used to substantiate theories and experiments; the methods are also powerful tools in providing insights into structural and dynamical behavior of molecules in different systems, to name a few: Nylon 66 crystals (Wendoloski et al., 1990), confined fluids in calcite nanopores (Santos et al., 2018), and biomolecule release in drug delivery (Pakulska et al., 2016). More recently, molecular simulations are also being applied to develop treatments and vaccines for the novel coronavirus (Bzówka et al., 2020; Han and Král, 2020), whose outbreak has caused an unprecedented pandemic in 2020.

Although molecular simulations provide an exact solution for a specific intermolecular model, they take considerable computational time to yield the results. Therefore, it is an inadequate tool for predicting properties at the industrial process operation time scale, when a fast response is needed. In this kind of application, theoretical and empirical models arise as a better suited alternative. Fundamental to process design and operation, the development of volumetric models for fluid property prediction had been for a long time confined to the generalization of the model proposed by van der Waals, a class known as cubic Equations of State (EoS). Statistical mechanics theories have, nevertheless, paved the way for a new class of models: the molecular-based equations of state. In comparison to the traditional cubic EoSs, these models have a stronger theoretical basis, and they are often more precise and versatile. On the other hand, in contrast to molecular simulations, approximations are needed for the formulation of these models even for the simplest intermolecular potential.

Even though molecular simulations provide an exact solution for a specific intermolecular model, the potential itself is already an approximation of the reality. For engineering applications, the main concern is not to unravel nature's ultimate reality, but to formulate models that reproduce the essential features of a specific system, neglecting secondary details (Wu and Prausnitz, 2019). To that end, in regard to the development of physical models, a balance between simplicity and accuracy is required. For instance, the geometry of molecules is usually very complex to be described mathematically, but has a great deal of influence on the calculation of the intermolecular interactions. Therefore, to model the shape of the particles, one should make simplifications that are still physically meaningful.

Nonspherical molecules have often been modeled as a set of spherical parts to account for their anisotropy, both in molecular simulations (*e.g.* the multi-site approach), and in theoretical models. In numerical methods, this strategy demands more computational effort, since, to model the interaction between two particles, one has to compute the interactions between each site on a molecule with all the other sites on the other molecule. This inspired Berne and Pechukas (1972) to propose an intermolecular potential to model molecules as a single ellipsoidal site, hence, the molecule anisotropy was accounted for considering only one interaction. Following a similar strategy, in this dissertation, the main goal is to develop both theoretical and numerical anisotropic models that can describe the geometry of the molecules and capture their behavior.

On the next Chapter, we introduce a new approach for developing equations of state for nonspherical molecules, based on a perturbation theory for ellipsoids. The SAFT (Statistical Associating Fluid Theory) approach (Chapman et al., 1989) and a Gaussian model potential (Berne and Pechukas, 1972) underpinned the development of the model. The equation of state was used to calculate thermodynamic properties of carbon dioxide and ethane.

On Chapter 3, a bridge between the theoretical model presented in Chapter 2 and molecular simulations was made. The parameters optimized using the equation of state were applied in Monte Carlo simulations using the same potential, unveiling the effects of the choice of the intermolecular potential on the prediction of the isochoric heat capacity. In addition to that, the validity and limitations of the theoretical approximations were tested.

Chapter 4 is devoted to extending the approach introduced on Chapter 2 to model molecules as cylinders and spherocylinders, besides ellipsoids. Furthermore, all three equations of state are applied to calculate vapor-liquid equilibrium of longer chains and disk-like molecules as well. The models were tested for n-alkanes from methane to octane, carbon dioxide, benzene, toluene, and n-perfluoroalkanes.

Although we have considered an isotropic distribution to formulate the equations of state for industrial relevant fluids, the hard gaussian overlap (HGO), hard spherocylinder (HSC), and hard cylinder (HC) models are often used to study liquid crystalline (LC) phase formation. Since the literature on LC phases of hard cylinders is very limited, on Chapter 5 we provide an investigation of the phase boundaries of the hard cylinder model using Monte Carlo molecular simulations.

The focus is then moved to the development of models to study liquid crystalline phases in biological systems, in Chapter 6. Monte Carlo simulation codes for cylinders with different features and interactions were developed in an attempt to capture some characteristics often present in biological systems such as hydrophobicity, chirality, and dependence on salt molality.

1.1 Fundamentals of Statistical Mechanics

In a thermodynamic system, there is a huge number of possible different ways for particles to be arranged (in terms of position and velocity) while expressing the same macroscopic properties. Each possible configuration defines a different state, or microstate, and together they compose an ensemble. Gibbs has postulated that the average of a property over all possible microstates, that is, the ensemble average, is equal to its corresponding thermodynamic property. The ensemble average of a property ϕ is written as:

$$\langle \phi \rangle_{\text{ensemble}} = \sum_{m}^{X} \phi_m P_m \tag{1.1}$$

where P_m is the probability that the system is found in a microstate m, and X is the total number of possible microstates.

1.1.1 Canonical ensemble

In an N-body system, at a certain temperature T occupying a volume V, the probability that the system is in a certain state m is:

$$P_m = \frac{\exp(-\beta E_m)}{\sum\limits_{j=1}^{X} \exp(-\beta E_j)} \tag{1.2}$$

where $\beta = 1/(k_B T)$, where k_B is the Boltzmann constant, T is the absolute temperature, E_m is the total energy of the microstate m, E_j is the total energy of each microstate j of the ensemble. From Equations 1.1 and 1.2, the average of a property ϕ in the canonical ensemble (NVT) is written as:

$$\langle \phi \rangle_{\mathrm{NVT}} = \frac{\sum\limits_{m=1}^{X} \phi_m \exp(-\beta E_m)}{\sum\limits_{m=1}^{X} \exp(-\beta E_m)} \tag{1.3}$$

The continuous description of the probability P_m is given in Equation 1.4.

$$P_m = \frac{\exp\left(-\beta(U_m + K_m)\right)}{\int\limits_{-\infty}^{+\infty} \cdots \int\limits_{-\infty}^{+\infty} \exp\left(-\beta(U(\vec{q}_1, \cdots, \vec{q}_N) + K(\vec{p}_1, \cdots, \vec{p}_N))\right) \mathrm{d}\vec{q}_1 \cdots \mathrm{d}\vec{q}_N \mathrm{d}\vec{p}_1 \cdots \mathrm{d}\vec{p}_N} \tag{1.4}$$

where $\vec{q}$ and $\vec{p}$ denote the configuration coordinates (position and orientation) and the momenta of each particle, respectively, U is the total potential energy, and K is the total kinetic energy. The denominator is the sum of the probability of all states, that is, the integration over all possible configurations and momenta, and it is related to the partition function $Q(N, V, T)$ defined as:

$$\begin{aligned} Q(N, V, T) =& \frac{1}{N! h^{3N}} \int\limits_{-\infty}^{+\infty} \cdots \int\limits_{-\infty}^{+\infty} \exp(-\beta(U(\vec{q}_1, \cdots, \vec{q}_N) \\ & + K(\vec{p}_1, \cdots, \vec{p}_N))) \mathrm{d}\vec{q}_1 \cdots \mathrm{d}\vec{q}_N \mathrm{d}\vec{p}_1 \cdots \mathrm{d}\vec{p}_N \end{aligned} \tag{1.5}$$

where h is the Planck constant. The Helmholtz free energy of the system is inherently connected to $Q(N, V, T)$ via the expression presented in Equation 1.6:

$$\beta A(N, V, T) = -\ln Q(N, V, T) \tag{1.6}$$

Performing the momentum integration in the partition function, one can write:

$$Q(N, V, T) = \frac{Z_N}{N!\nu^N} \tag{1.7}$$

where ν is the de Broglie volume that incorporates translational and rotational contributions to the kinetic energy. For the matter of this dissertation, since only rigid particles are studied, the vibrational degrees of freedom are neglected. Z_N is the configurational integral defined as:

$$Z_N = \int\limits_{-\infty}^{+\infty} \cdots \int\limits_{-\infty}^{+\infty} \exp\left(-\beta U(\vec{q}_1, \cdots, \vec{q}_N)\right) \mathrm{d}\vec{q}_1 \cdots \mathrm{d}\vec{q}_N \tag{1.8}$$

The probability of finding particle 1 in a configuration $\vec{q}_1$ and a particle 2 in a configuration $\vec{q}_2$ irrespective of the configuration of the other particles is then:

$$P(\vec{q}_1, \vec{q}_2) = \frac{\int\limits_{-\infty}^{+\infty} \cdots \int\limits_{-\infty}^{+\infty} \exp(-\beta U(\vec{q}_1, \cdots, \vec{q}_N)) \mathrm{d}\vec{q}_3 \cdots \mathrm{d}\vec{q}_N}{Z_N} \tag{1.9}$$

The probability that a particle is in $\vec{q}_1$ and any other particle is in $\vec{q}_2$ is then:

$$\rho(\vec{q}_1, \vec{q}_2) = \frac{N!}{(N-2)!} \frac{\int\limits_{-\infty}^{+\infty} \cdots \int\limits_{-\infty}^{+\infty} \exp(-\beta U(\vec{q}_1, \cdots, \vec{q}_N)) \mathrm{d}\vec{q}_3 \cdots \mathrm{d}\vec{q}_N}{Z_N} \tag{1.10}$$

If the molecules were uncorrelated (as for the ideal gas), $\rho(\vec{q}_1, \vec{q}_2)$ would be equal to (N^2/V^2), and this probability would be redefined as $\rho^{(2)}$. The function $g(\vec{q}_1, \vec{q}_2)$ is defined as a correction to $\rho^{(2)}$ due to the presence of an intermolecular potential, that is, when the position of each particle affects the position of the others. Equation 1.10 is rewritten as:

$$\rho^2 g(\vec{q}_1, \vec{q}_2) = N(N-1) \frac{\int\limits_{-\infty}^{+\infty} \cdots \int\limits_{-\infty}^{+\infty} \exp(-\beta U(\vec{q}_1, \cdots, \vec{q}_N)) \mathrm{d}\vec{q}_3 \cdots \mathrm{d}\vec{q}_N}{Z_N} \tag{1.11}$$

1.1.2 Perturbation Theory

Finding an analytical solution for the Helmholtz free energy (Equation 1.6) is far from being an easy task, since such an endeavor involves solving a very complex integral (Equation 1.8). Carnahan and Starling (1969) developed an equation of state for the hard-sphere potential by approximating the virial coefficients by integers, and expressing them as a geometric series. The resulting expression is presented in Equation 1.12.

$$\frac{pV}{Nk_BT} = \frac{1+\eta+\eta^2-\eta^3}{(1-\eta)^3} \tag{1.12}$$

where $\eta = Nv_{\text{particle}}/V$ is the packing fraction, where v_{particle} is the volume of each particle, p is the pressure, and V is the volume of the system.

Since $p = -\left(\frac{\partial A}{\partial V}\right)_T$, integrating Equation 1.12, the expression for A^{HS} is obtained:

$$\beta a^{\text{HS}} = \frac{4\eta - 3\eta^2}{(1-\eta)^2} \tag{1.13}$$

where $a^{\text{HS}} = A^{\text{HS}}/N$ is the molar Helmholtz free energy.

Perturbation theory is a popular strategy for obtaining an approximate solution for the Helmholtz free energy of more complex intermolecular potentials. The cornerstone idea in perturbation theory is that, at least at high densities, the structure of a liquid is mostly determined by the way in which the hard cores pack together (Hansen and McDonald, 2006), while the long-range attraction interactions provide an uniform attractive potential. In this sense, one can treat the repulsive short-range part of an intermolecular potential as a reference system, and the forces of attraction as small perturbations on the forces of repulsion.

Longuet-Higgins (1951) proposed a separation of the intermolecular potential into a reference (superscript 0) and perturbed system (superscript 1) (Equation 1.14). Zwanzig (1954) applied the hard-sphere potential as the reference system and the Lennard-Jones potential as the perturbation and proposed a high-temperature expansion to formulate the equation of state.

$$U = U^{(0)} + U^{(1)} \tag{1.14}$$

Replacing Equation 1.14 into Equation 1.8:

$$Z_N = \int\limits_{-\infty}^{+\infty} \cdots \int\limits_{-\infty}^{+\infty} \exp\left(-\beta U^{(0)}\right) \exp(-\beta U^{(1)}) \mathrm{d}\vec{q}_1 \cdots \mathrm{d}\vec{q}_N \tag{1.15}$$

Multiplying and dividing Equation 1.15 by $Z_N^{(0)} = \int\limits_{-\infty}^{+\infty} \cdots \int\limits_{-\infty}^{+\infty} \exp\left(-\beta U^{(0)}\right) \mathrm{d}\vec{q}_1 \ldots \mathrm{d}\vec{q}_N$, one has:

$$Z_N = Z_N^{(0)} \langle \exp\left(-\beta U^{(1)}\right)\rangle_0 \tag{1.16}$$

where $\langle \exp\left(-\beta U^{(1)}\right)\rangle_0$ is the average of $\exp\left(-\beta U^{(1)}\right)$ over the reference system configurational integral.

Applying the Taylor series expansion:

$$\langle \exp\left(-\beta U^{(1)}\right)\rangle_0 = 1 - \beta\langle U^{(1)}\rangle_0 + \frac{\beta^2\langle U^{(1)2}\rangle_0}{2} - \frac{\beta^3\langle U^{(1)3}\rangle_0}{6} + \cdots \tag{1.17}$$

The Helmholtz free energy is then written, considering only the translational and rotational degrees of freedom of the kinetic energy, as:

$$-\beta A = \ln Q = \ln\left(Z_N\right) - \ln(N!\nu^{3N}) = \ln\left(\frac{Z_N^{(0)}}{N!\nu^{3N}}\right) + \ln\left(\langle \exp\left(-\beta U^{(1)}\right)\rangle_0\right) \tag{1.18}$$

$$-\beta A = -\beta A^{(0)} - \beta A^{(1)} \tag{1.19}$$

$$-\beta A^{(1)} = \ln\left(\langle \exp(-\beta U^{(1)})\rangle_0\right) \tag{1.20}$$

$$\exp\left(-\beta A^{(1)}\right) = \langle \exp(-\beta U^{(1)})\rangle_0 \tag{1.21}$$

Writing $A^{(1)}$ as a power series of β:

$$A^{(1)} = \sum_{n=1}^{+\infty} \frac{w_n \beta^{n-1}}{n!} \tag{1.22}$$

Replacing Equations 1.22 and 1.17 into Equation 1.21, applying a Taylor series expansion:

$$1 + \sum_{i=1}^{+\infty} \frac{\left(\sum\limits_{n=1}^{+\infty} \frac{-w_n\beta^n}{n!}\right)^i}{i!} = 1 - \beta\langle U^{(1)}\rangle_0 + \frac{\beta^2\langle U^{(1)2}\rangle_0}{2} - \frac{\beta^3\langle U^{(1)3}\rangle_0}{6} + \cdots \tag{1.23}$$

$$\begin{aligned}
&1+(-w_1\beta-\frac{w_2\beta^2}{2}-\frac{w_3\beta^3}{6}-\cdots)+\frac{(-w_1\beta-\frac{w_2\beta^2}{2}-\frac{w_3\beta^3}{6}-\cdots)^2}{2}\\
&+\frac{(-w_1\beta-\frac{w_2\beta^2}{2}-\frac{w_3\beta^3}{6}-\cdots)^3}{6}+\mathcal{O}(\beta^4)\\
&=1-\beta\langle U^{(1)}\rangle_0+\frac{\beta^2\langle U^{(1)2}\rangle_0}{2}-\frac{\beta^3\langle U^{(1)3}\rangle_0}{6}+\mathcal{O}(\beta^4)
\end{aligned} \tag{1.24}$$

Truncating $A^{(1)}$ expansion at the second term, and substituting Equation 1.22 into Equation 1.19:

$$-\beta A=-\beta A^{(0)}-w_1\beta-\frac{w_2\beta^2}{2}+\mathcal{O}(\beta^3) \tag{1.25}$$

Taking like powers of β in Equation 1.24, one can compute expressions for w_1 and w_2:

$$\begin{aligned}
w_1&=\langle U^{(1)}\rangle_0\\
w_2&=\langle U^{(1)}\rangle_0^2-\langle\left(U^{(1)}\right)^2\rangle_0
\end{aligned} \tag{1.26}$$

First-order term

To calculate w_1, one needs to find an expression for $\langle U^{(1)}\rangle_0$:

$$w_1=\langle U^{(1)}\rangle_0=\frac{1}{Z_N^{(0)}}\int\limits_{-\infty}^{+\infty}\cdots\int\limits_{-\infty}^{+\infty}U^{(1)}\exp\left(-\beta U^{(0)}\right)\mathrm{d}\vec{q}_1\cdots\mathrm{d}\vec{q}_N \tag{1.27}$$

Assuming pairwise-additive interactions:

$$U=\sum_{i=1}^{N-1}\sum_{j>i}^{N}u(\vec{q}_i,\vec{q}_j)=\frac{N(N-1)}{2}u(\vec{q}_1,\vec{q}_2) \tag{1.28}$$

Replacing Equation 1.28 in Equation 1.27 :

$$w_1=\frac{N(N-1)}{2}\frac{1}{Z_N^{(0)}}\int\limits_{-\infty}^{+\infty}\int\limits_{-\infty}^{+\infty}u(\vec{q}_1,\vec{q}_2)\int\limits_{-\infty}^{+\infty}\cdots\int\limits_{-\infty}^{+\infty}\exp(-\beta U^{(0)})\mathrm{d}\vec{q}_3\cdots\mathrm{d}\vec{q}_N\mathrm{d}\vec{q}_1\mathrm{d}\vec{q}_2 \tag{1.29}$$

From Equation 1.11, one may rewrite Equation 1.29 in terms of $\rho g^{(0)}(\vec{q}_1,\vec{q}_2)$:

$$w_1=\frac{\rho^2}{2}\int\limits_{-\infty}^{+\infty}\int\limits_{-\infty}^{+\infty}u^{(1)}(\vec{q}_1,\vec{q}_2)g^{(0)}(\vec{q}_1,\vec{q}_2)\mathrm{d}\vec{q}_1\mathrm{d}\vec{q}_2 \tag{1.30}$$

where $g^{(0)}(\vec{q}_1,\vec{q}_2)$ is the pair correlation function of the unperturbed system. An exact solution for the second order term w_2 is complex, since in the calculation of $\langle \left(U^{(1)}\right)^2 \rangle$ terms as $u(\vec{q}_i, \vec{q}_j)u(\vec{q}_k, \vec{q}_l)$ appear, that is, three and four body interactions. Nevertheless, Barker and Henderson (1967) formulated an expression for w_2 based on approximations discussed on the next section. Franco et al. (2017a) formulated a two-body perturbation theory for the Helmholtz free energy and demonstrated that the first order theory is a truncated approximation of the former.

Second-order term - Barker-Henderson approach

Barker and Henderson (1967) proposed two slightly different approximations to consider the second-order term in the temperature expansion. First, to solve Equation 1.16, the strategy was to imagine the existing intermolecular distances that contribute to the potential arranged into groups of small intervals, such as r_0 and r_1, $\cdots$, r_{j-1}, and r_j. Hence, let τ_1 be the number of intermolecular distances lying between r_0 and r_1, τ_j between r_{j-1} and r_j, and so on. By taking small enough intervals, one can assume a constant potential in each one of them and write the total intermolecular potential as:

$$U^{(1)} = \sum_j \tau_j u_j^{(1)} \tag{1.31}$$

Replacing Equation 1.31 into Equation 1.16:

$$Z_N = Z_N^{(0)} \left\langle \exp\left(-\beta \sum_j \tau_j u_j^{(1)}\right) \right\rangle_0 \tag{1.32}$$

The normalized probability that a combination of exactly τ_1, $\cdots$, τ_j molecular pairs lie in each interval in the reference system is written as $P(\tau_1, \cdots, \tau_j) = P(\{\tau_j\})$. Thus, the configurational integral is rewritten as:

$$Z_N = Z_N^{(0)} \sum_{\{\tau_j\}} P(\{\tau_j\}) \exp\left(-\beta \sum_j \tau_j u_j^{(1)}\right) \tag{1.33}$$

Expanding $\exp\left(-\beta \sum_j \tau_j u_j^{(1)}\right)$:

$$Z_N = Z_N^{(0)} \sum_{\{\tau_j\}} P(\{\tau_j\}) \left[1 + \sum_{x=1} \frac{\left(-\beta \sum_j \tau_j u_j^{(1)}\right)^x}{x!} \right] \tag{1.34}$$

Since $P(\{\tau_j\})$ is the normalized probability, $\sum_{\{\tau_j\}} P(\{\tau_j\}) = 1$, then:

$$\frac{Z_N}{Z_N^{(0)}} = 1 - \beta \sum_j u_j^{(1)} \sum_{\{\tau_j\}} P(\{\tau_j\})\tau_j + \frac{\beta^2}{2} \sum_j \sum_k u_j u_k \sum_{\{\tau_j\}} P(\{\tau_j\})\tau_j \tau_k + \mathcal{O}(\beta^3) \tag{1.35}$$

Equation 1.35 can be rewritten as:

$$\frac{Z_N}{Z_N^{(0)}} = 1 - \beta \sum_j u_j^{(1)} \langle \tau_j \rangle_0 + \frac{\beta^2}{2} \sum_j \sum_k u_j^{(1)} u_k^{(1)} \langle \tau_j \tau_k \rangle_0 + \mathcal{O}(\beta^3) \tag{1.36}$$

By expanding the natural logarithm of Equation 1.36 using Taylor series expansion, one finds the expression for the free Helmholtz energy:

$$\begin{aligned} -\beta A + \beta A^{(0)} = & -\beta \sum_j u_j^{(1)} \langle \tau_j \rangle_0 + \frac{\beta^2}{2} \sum_j \sum_k u_j^{(1)} u_k^{(1)} \langle \tau_j \tau_k \rangle_0 \\ & - \frac{1}{2} \left(\beta^2 \sum_j \sum_k u_j^{(1)} u_k^{(1)} \langle \tau_j \rangle_0 \langle \tau_k \rangle_0 \right) + \mathcal{O}(\beta^3) \end{aligned} \tag{1.37}$$

Rearranging Equation 1.37:

$$\begin{aligned} -\beta A + \beta A^{(0)} = & -\beta \sum_j u_j^{(1)} \langle \tau_j \rangle_0 \\ & + \frac{\beta^2}{2} \left(\sum_j \sum_{k \neq j} u_j^{(1)} u_k^{(1)} \langle \tau_j \tau_k \rangle_0 - \sum_j \sum_{k \neq j} u_j^{(1)} u_k^{(1)} \langle \tau_j \rangle_0 \langle \tau_k \rangle_0 \right) \\ & + \frac{\beta^2}{2} \left(\sum_j u_j^{(1)2} \langle \tau_j^2 \rangle_0 - \sum_j u_j^{(1)2} \langle \tau_j \rangle_0^2 \right) + \mathcal{O}(\beta^3) \end{aligned} \tag{1.38}$$

The first moment $\langle \tau_j \rangle_0$ is the average number of intermolecular distances lying in an interval between r_{j-1} and r_j in the unperturbed system, and can be readily calculated from the radial distribution function $g(r)$. The probability of finding a molecule at a distance r from a central particle is $\rho g(r)$. Hence, the number of particles at a distance r from each other is $N\rho g(r)$. Therefore, the number of molecules at a distance between r_{j-1} and r_j apart from each other in the reference system is:

$$N_{r_{j-1},r_j} = N\rho \int\limits_{r_{j-1}}^{r_j} g^{(0)}(r) 4\pi r^2 \mathrm{d}r \tag{1.39}$$

The average number of molecules pairs (or intermolecular distances) in the given interval is half N_{r_{j-1},r_j}:

$$\langle \tau_j \rangle_0 = 2\pi\rho N \int\limits_{r_{j-1}}^{r_j} g^{(0)}(r) r^2 \mathrm{d}r = 2\pi\rho N g(r) r^2 (r_j - r_{j-1}) \tag{1.40}$$

The exact numerical calculation of the second moment is a hard task, thus Barker and Henderson resorted to a physical approximation. τ_j might be understood as the number of molecules in spherical shells surrounding other central molecules. Barker and Henderson (1967) assumed that the shells could be treated as large macroscopic volumes, and for this reason one could consider that:

1. The number of molecules in different shells would be uncorrelated:

$$\langle \tau_j \tau_k \rangle - \langle \tau_j \rangle \langle \tau_k \rangle = 0, \quad \text{if} \quad j \neq k \tag{1.41}$$

2. The fluctuation of the number in a given shell is:

$$\langle \tau_j^2 \rangle - \langle \tau_j \rangle^2 = \langle \tau_j \rangle k_B T \frac{\partial \rho}{\partial p} \tag{1.42}$$

This approach is known as the Macroscopic Compressibility Approximation (MCA). Substituting Equation 1.40 into Equation 1.38, and considering the continuum description:

$$-\beta A + \beta A^{(0)} = -\beta 2N\pi\rho \int u^{(1)}(r) g^{(0)}(r) r^2 \mathrm{d}r + \beta^2 N\pi\rho \int u^{(1)2}(r) g^{(0)}(r) r^2 k_B T \left(\frac{\partial \rho}{\partial p}\right)_0 \mathrm{d}r \tag{1.43}$$

Since the shells are in fact microscopic, a more plausible alternative to Equation 1.42 is to replace ρ by the local density $\rho g^{(0)}(r)$. This strategy is known as the *local compresibility* *approximation* (LCA). Equation 1.43 then becomes:

$$\beta A = \beta A^{(0)} + a_1\beta + a_2\beta^2 \tag{1.44a}$$

where

$$a_1 = 2N\pi\rho \int u^{(1)}(r) g^{(0)}(r) r^2 \mathrm{d}r \tag{1.44b}$$

$$a_2 = -N\pi\rho \int u^{(1)2}(r) r^2 k_B T \left(\frac{\partial \rho g^{(0)}(r)}{\partial p}\right)_0 \mathrm{d}r \tag{1.44c}$$

Zhang's correction to the MCA

Zhang (1999) improved the Barker-Henderson MCA by assuming that the number of molecules in two neighbor shells would be correlated, and that the correlation coefficient would be directly proportional to the square of the packing fraction. After this correction, the resulting expression for the second-order term is simply $a_2^{\mathrm{ZH}} = (1 + 8.23\eta^2)a_2^{\mathrm{BH}}$, where a_2^{BH} is the Barker-Henderson formulation. One may write a_2 in a generalized form as:

$$a_2 = -(1+\Upsilon)N\pi\rho \int {u^{(1)}}^2(r) r^2 k_B T \left(\frac{\partial \rho g^{(0)}(r)}{\partial p}\right)_0 \mathrm{d}r \tag{1.45}$$

where $\Upsilon = 0$ for Barker-Henderson expression and $\Upsilon = 8.23\eta^2$ for Zhang's formulation.

1.2 Nonspherical Systems

Modeling nonspherical particles is very common in the study of liquid crystalline mesophases, which entail a state of matter that has liquid and crystals properties at the same time, combining mobility and order. Besides the translational, orientational degrees of freedom are added to the problem when it comes to nonspherical bodies configurations in a microstate. To define the orientations, a coordinate axes system w_i, v_i, u_i is fixed on each particle (where i is the index of the particles). The orientation of a particle is set as an unit vector $\hat{\Omega}$ parallel to the u-axis, in a reference coordinate system x, y, z fixed in time (Figure 1.2).

A system of elongated nonspherical bodies can be categorized by its degree of order (Figure 1.2). An isotropic phase is completely disorded, while a nematic phase is characterized by a positional disorder coupled with orientational order. A smectic phase, on

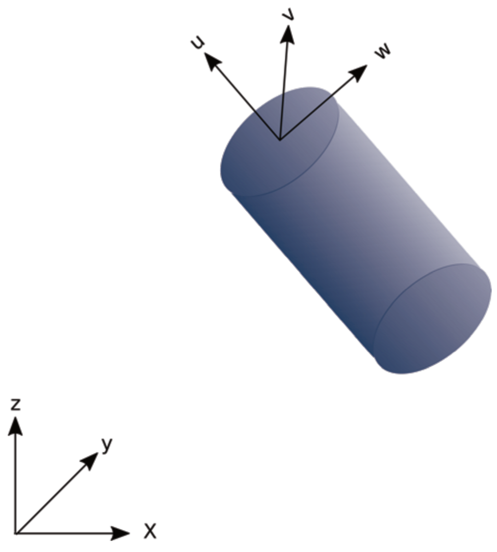


Figure 1.1: Illustration of an axes coordinate system on a particle.

the order hand, exhibits both orientational and positional order, where the molecules are arranged in layers. The phase director n is an unit vector along the particles preferred orientation.

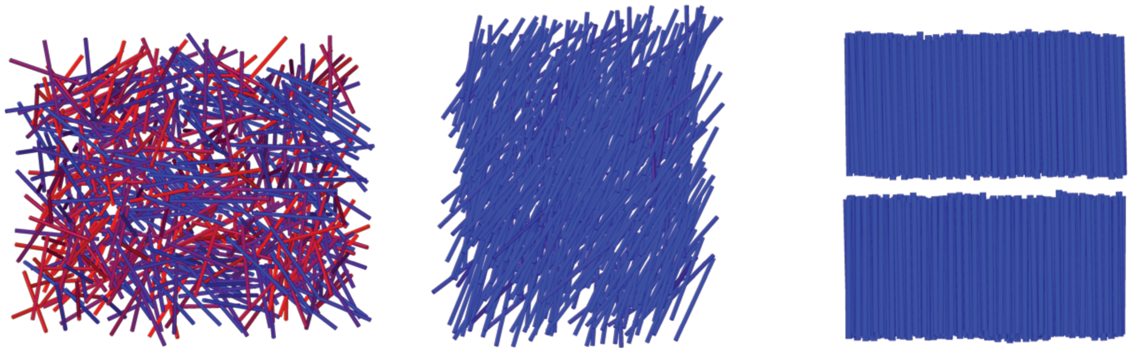


Figure 1.2: Liquid crystalline phases.

1.2.1 Nematic order parameter

The nematic order parameter s is defined as the average over orientations of the second Legendre polynomial P_2 of the angle between the phase director n and the orientation $\hat{\Omega}$ of each particle (Equation 1.46).

$$s = \langle P_2(\cos\theta)\rangle_{\hat{\Omega}} = \left\langle \frac{3}{2}\cos^2\theta - \frac{1}{2}\right\rangle_{\hat{\Omega}} = \left\langle \frac{3}{2}(n\cdot\hat{\Omega})^2 - \frac{1}{2}\right\rangle \tag{1.46}$$

Considering cylindrical symmetric particles, the orientation is solely a function of the polar angle θ. Given the normalized orientational distribution function $f(\theta)$, the nematic order parameter s can be calculated as:

$$s = \int_0^{2\pi}\int_0^{\pi} \left(\frac{3}{2}\cos^2\theta - \frac{1}{2}\right) f(\theta)\mathrm{d}\theta\mathrm{d}\phi \tag{1.47}$$

The isotropic phase is characterized by a uniform distribution of the orientations, *i.e.*, there is no preferable direction. The isotropic orientational distribution function $f(\theta)$ is calculated as shown in Equation 1.48.

$$f^{\mathrm{iso}}(\theta) = \frac{1}{\int_0^{2\pi}\int_0^{\pi} \sin(\theta)\mathrm{d}\theta\mathrm{d}\phi} = \frac{1}{4\pi} \tag{1.48}$$

The nematic order parameter for an isotropic phase is obtained:

$$\begin{aligned} s^{\mathrm{iso}} &= \int_0^{2\pi}\int_0^{\pi} \left(\frac{3}{2}\cos^2\theta - \frac{1}{2}\right) f^{\mathrm{iso}}(\theta)\mathrm{d}\theta\mathrm{d}\phi \\ &= \frac{1}{2}\int_1^{-1} \left(\frac{1}{2} - \frac{3}{2}x^2\right)\mathrm{d}x \\ &= 0 \end{aligned} \tag{1.49}$$

For a phase with perfect orientational organization, the orientational distribution function is zero everywhere with exception of $\theta = 0$ and $\theta = \pi$. Therefore, the distribution function for a perfect nematic phase is treated as a Dirac-δ function:

$$f^{\text{perfect nematic}}(\theta) = \frac{1}{4\pi}\left(\frac{3}{2}\cos(\delta(\theta-0)) - \frac{1}{2} + \frac{3}{2}\cos(\delta(\theta-\pi)) - \frac{1}{2}\right) \tag{1.50}$$

$$\begin{aligned} s^{\text{perfect nematic}} &= \int_0^{2\pi}\int_0^{\pi} \left(\frac{3}{2}\cos^2\theta - \frac{1}{2}\right) f^{\text{perfect nematic}}(\theta)\mathrm{d}\theta\mathrm{d}\phi \\ &= \frac{1}{2}\left(\frac{3}{2} - \frac{1}{2} + \frac{3}{2} - \frac{1}{2}\right) \\ &= 1 \end{aligned} \tag{1.51}$$

Accordingly, $s = 1$ in a perfect nematic phase, and $s = 0$ in a isotropic phase. Fluctuations, however, might appear due to finite-size effects.

1.2.2 Smectic order parameter

The smectic order parameter τ is defined as follows:

$$\tau = \frac{1}{N}\left|\sum_{i=1}^{N} \exp\left(i2\pi\frac{|r_i^{||}|}{d}\right)\right| \tag{1.52}$$

where $r_i^{||}$ is the projection of the vector of the center of mass of particle i along the direction parallel to the phase director n, N is the total number of particles, and d is the periodicity of the smectic layers. Hence, if the phase is organized in layers, $|r_i^{||}|/d$ tends to be an integer and $\tau \to 1$. Since the periodicity is previously unknown, one should apply different values and find such d that maximizes τ.

1.2.3 Hexatic order parameter

The hexatic order parameters ψ_6 is a measure of the two-dimensional in-plane hexagonal order present in columnar and crystalline phases. ψ_6 is defined as:

$$\psi_6 = \frac{1}{N}\sum_k \left|\frac{1}{n_{(k)}}\sum_{\langle ij\rangle} e^{6\mathrm{i}\theta_{ij}}\right| \tag{1.53}$$

where $n_{(k)}$ is the number of all possible pairs of nearest neighbors of particle k within a single layer, θ_{ij} is the angle between the projection of the intermolecular vectors $\mathbf{r_{ki}}$ and $\mathbf{r_{kj}}$ onto the plane perpendicular to the phase director ($\mathbf{r}_{\mathbf{ki}}^{\perp}$ and $\mathbf{r}_{\mathbf{kj}}^{\perp}$) (Figure 1.3), and the sum $\sum\limits_{<ij>}$ is over all possible pairs of the nearest neighbors of particle k within the first coordination shell. In this way, ψ_6 tends to one for phases with in-plane hexagonal order (six nearest neighbors within the same layer), and it approaches zero otherwise.

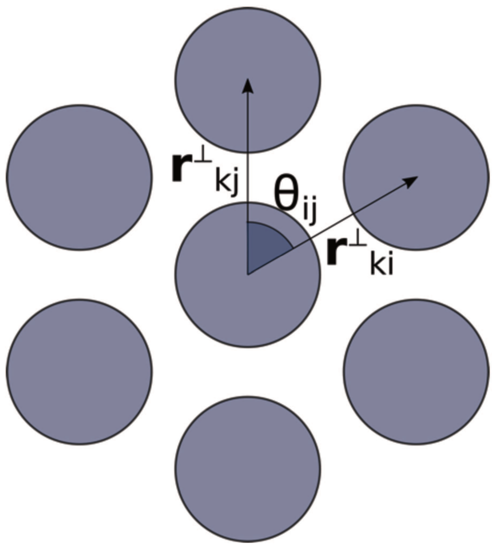


Figure 1.3: Illustration of a transversal section of cylindrical particles with in-plane hexagonal order.

"Thoroughly conscious ignorance is a prelude to every real advance in knowledge."

— James Clerk Maxwell

2 A new approach for the development of equations of state for nonspherical particles

The content of this chapter was reprinted (adapted) with permission from (Lopes, J. T. and Franco, L. F. New Thermodynamic Approach for Nonspherical Molecules Based on a Perturbation Theory for Ellipsoids. Ind. Eng. Chem. Res. 58, 6850–6859 (2019).). Copyright (2020) American Chemical Society. https://doi.org/10.1021/acs.iecr.9b00766 Lopes and Franco (2019)

2.1 Introduction

Volumetric equations of state play a pivotal role in a wide range of scientific and industrial processes. Over the past decades, molecular-based equations of state (EoS) have risen as powerful alternatives to the so called cubic equations of state, since the latter are sensitive to the experimental data used to adjust their parameters and can be inadequate to extrapolated conditions.

Chapman et al. (1989) proposed the Statistical Association Fluid Theory (SAFT), a successful approach that laid ground for other relevant models. Differing mainly with respect to the intermolecular potential applied in the reference and perturbed terms, many variations of SAFT have been developed over the last decades, *e.g.*, SAFT-VR SW (Gil-Villegas et al., 1997), soft-SAFT (Blas and Vega, 1997), PC-SAFT (Gross and Sadowski, 2001), and SAFT-VR Mie (Lafitte et al., 2013). These equations have been successfully applied to different fluid mixtures (Economou, 2002) for both phase equilibrium as well as derivative properties (Nikolaidis et al., 2018), and have also been extended to electrolytes (Cameretti et al., 2005; Das et al., 2015; Eriksen et al., 2015; Selam et al., 2018), and confined fluids (Franco et al., 2017b; Araújo and Franco, 2019; Aslyamov et al., 2019). PC-SAFT (Gross and Sadowski, 2001) and SAFT-VR Mie (Lafitte et al., 2013) are present in many commercial chemical process simulators.

Based on the perturbation theory developed by Wertheim (Wertheim, 1984b,a, 1986a,b,c, 1987) for highly anisotropic fluid interactions, in SAFT original approach, Chapman et al. (1989) consider that the Helmholtz free energy of a system is a sum of different contributions (Equation 2.1).

$$A = A^{\text{ideal}} + A^{\text{segment}} + A^{\text{chain}} + A^{\text{association}} \tag{2.1}$$

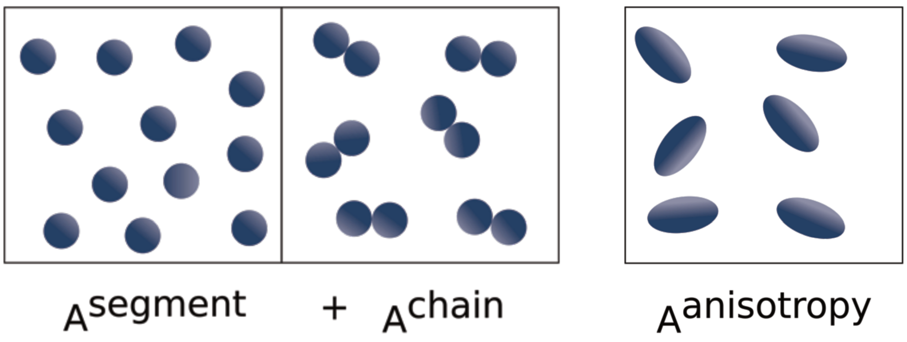
where A^{ideal} is the ideal gas contribution, A^{segment} is the contribution of the interaction between spherical segments, A^{chain} is the contribution due to covalent bond formation between segments, and $A^{\text{association}}$ is the contribution of the hydrogen bonds.

In this approach, the anisotropic shape of the molecule is built by associating isotropic shaped particles, that is, spherical segments. To retain physical meaning, the number of segments m, one of the parameters of the model, should be an integer. Nevertheless, for the sake of better correlation with experimental data, m has been often fitted to a non-integer in most of SAFT EoS applications. Some molecules, however, when considered to be spherical, are modeled as a single segment, *e.g.* methane (Gil-Villegas et al., 1997; Gross and Sadowski, 2001; Lafitte et al., 2013), water (Dufal et al., 2015), and hydrogen (Nikolaidis et al., 2018). A non-integer number of segments fitted for nonspherical molecules weakens its physical meaning and precludes a more predictive use of it. Gil-Villegas *et al.* (Gil-Villegas et al., 1997) have shown that, for chains of square-well monomers, as the number of segments increases, the overprediction of the vapor-liquid coexistence curve obtained with SAFT-VR SW when compared to results obtained with Monte Carlo simulations is magnified. These observations might lead to the conclusion that something is rather missing in the theory, or that some approximations used in the chain formation contribution are inadequate.

For nonspherical molecules, one might suppose that, instead of having spherical segments in a chain, such molecules could be represented as single ellipsoids. In this perspective, we propose a modification of the SAFT original approach in which small chains and linear molecules are taken as one single ellipsoidal segment rather than a set of spherical segments. We account for A^{segment} and A^{chain} in only one new term, called $A^{\text{anisotropy}}$. For non-associative molecules, the fluid free energy is given by Equation 2.2.

$$A = A^{\text{ideal}} + A^{\text{anisotropy}} \tag{2.2}$$

Figure 2.1 illustrates how the residual free energy is composed in the SAFT original approach and in our proposed model.



(a) SAFT original approach (b) Proposed model

Figure 2.1: Diagram to illustrate the residual free energy in SAFT model and in our proposed equation of state

Wu et al. (2014) have developed a theoretical equation of state for hard sphero-cyllinders (HSC) with an anisotropic square-potential to study the LC behavior. Williamson and Del Rio (1998) have presented two different theories to describe the isotropic and nematic phases in a fluid of HSC with a spherocylindrical square-well potential, which García-Sánchez et al. (2002) extended to second-order perturbation theory. With a strategy similar to the one we propose in this work, Williamson and Guevara (1999) applied the earlier developed EoS(Williamson and Del Rio, 1998) for spherocylinders to examine the deviation from corresponding states as a function of molecular shape anisotropy. They also fitted the theoretical results to experimental data to use the model to calculate saturated densities of n-alkanes as a function of temperature. Here, we optimize the parameters of our proposed equation of state to correlate saturated properties and we also use the EoS to assess the predictive power of the model by calculating supercritical and derivative properties.

2.2 Formulation of the Equation of State

An exact solution to A by means of classical statistical mechanics is complex even for the simplest intermolecular potential (Equations 1.6, 1.7, 1.8), since it involves, among other complex calculations, the still unsolved many-body problem. Theoretical approximations with reasonable physical arguments must be made to formulate $A^{\text{anisotropy}}$. Perturbation theory is one of the most popular routes to find an approximate solution. To calculate $A^{\text{anisotropy}}$, we apply the Hard Gaussian Overlap (HGO) as the reference system in the Barker-Henderson second-order perturbation theory, and the attractive part of the spherical square-well potential as the perturbation (Equation 1.44).

$$a^{\text{anisotropy}} = a^{\text{HGO}} + a_1^{\text{SW}}\beta + a_2^{\text{SW}}\beta^2 \tag{2.3}$$

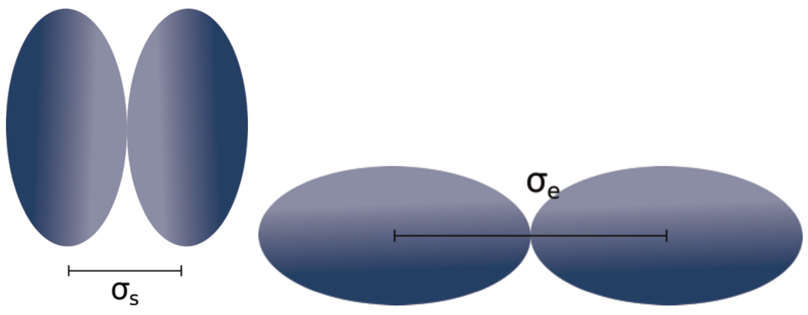
where $a^{\text{anisotropy}} = A^{\text{anisotropy}}/N$ and $a^{\text{HGO}} = A^{\text{HGO}}/N$.

Although the application of the spherical square-well attractive potential is a strong approximation, it is based on the assumption that, at larger distances, the orientations become less significant for the interaction. On the next sections, the derivations of A^{HGO}, a_1^{SW}, and a_2^{SW} are presented.

2.2.1 Hard Gaussian Overlap (HGO)

The Gaussian Model Potentials were proposed by Berne and Pechukas (1972) as a strategy to reduce computational demands in simulating polyatomic and nonspherical molecules. The molecules are represented as ellipsoids (or a rigid union of them) and the potential between the particles is associated with the mathematical overlap of two Gaussian distributions. Hence, the number of interactions between two particles considerably decreases if compared to the multi-site approach.

The mathematical structure of the model is similar to the Lennard-Jones potential, but the main difference is that the contact distance depends on the orientations of the particles. The ellipsoids are characterized by the parameters σ_s and σ_e, which correspond to the distances where the potential becomes zero when the particles are on a side-by-side or end-by-end configuration, respectively. See Figure 2.2.



(a) Side-by-side (b) End-by-end

Figure 2.2: Ellipsoids configurations.

From σ_s and σ_e, the elongation, κ, and the shape anisotropy parameter, χ, can be defined as:

$$\kappa = \frac{\sigma_e}{\sigma_s} \quad \text{and} \quad \chi = \frac{\kappa^2 - 1}{\kappa^2 + 1} \tag{2.4}$$

For spherical particles, $\kappa \to 1$ and $\chi \to 0$, for long rods $\kappa \to \infty$ and $\chi \to 1$, and for very thin disks $\kappa \to 0$ and $\chi \to -1$.

The Hard Gaussian Overlap is based on the Gaussian Model Potentials, and it is equivalent to the hard sphere model in the sense that the potential is infinite if the particles are in contact with each other, and zero otherwise. Nevertheless, the HGO contact distance, σ_{HGO}, varies with the orientation of the particles and also with the vector linking the centers of mass, as illustrated in Figure 2.3. For the HGO potential, σ_e and σ_s are simply the diameters of the ellipsoids of revolution.

The HGO contact distance, $\sigma_{\text{HGO}}(\vec{r}, \hat{\Omega}_1, \hat{\Omega}_2)$, is constrained between the values of σ_s and σ_e depending on the orientation of the particles, as presented in Equation 2.5:

$$\sigma_{\text{HGO}} = \sigma_s \left[1 - \frac{\chi}{2} \left(\frac{(\hat{\Omega}_1 \cdot \hat{r} + \hat{\Omega}_2 \cdot \hat{r})^2}{1 + \chi(\hat{\Omega}_1 \cdot \hat{\Omega}_2)} + \frac{(\hat{\Omega}_1 \cdot \hat{r} - \hat{\Omega}_2 \cdot \hat{r})^2}{1 - \chi(\hat{\Omega}_1 \cdot \hat{\Omega}_2)} \right) \right]^{-1/2} \tag{2.5}$$

where $\hat{r}$ is the unit vector along the vector connecting two ellipsoids centers of mass, and $\hat{\Omega}_1$ and $\hat{\Omega}_2$ are the unit vectors along the axis of the ellipsoids, which represent the

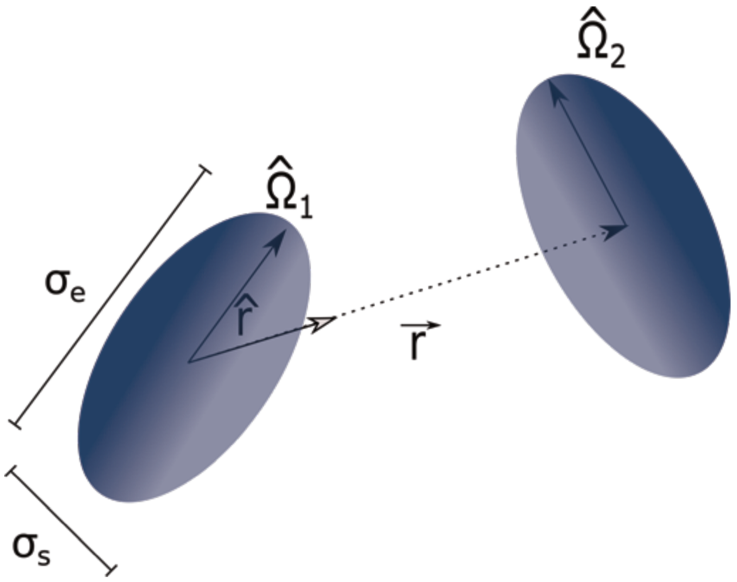


Figure 2.3: Two ellipsoids with different orientations at a certain distance $|\vec{r}|$.

molecules orientations.

HGO overestimates the contact distance of two perpendicular ellipsoids, leading to differences when compared to the Hard Ellipsoid of Revolution (HER). HER contact distance, however, is much more difficult to be obtained (Vieillard-Baron, 1972). Perera (2008) has examined the differences between fluids of hard and Gaussian ellipsoids, showing that, although the latter leads to small differences in Mayer function, quite similar values for the integral of these functions are obtained.

Derivation of $A^{\mathbf{HGO}}$

Onsager (1949) proposed an expression for the configurational-integral of fluids with orientational degrees of freedom by treating particles of different orientations as particles of different kinds. Neglecting the terms which depend on second-order or higher cluster integrals in the configurational expansion, the configurational-integral expression for cylindrical symmetric particles proposed by the author is shown in Equation 2.6.

$$\frac{\ln Z_N}{N} = 1 - \ln \rho - \sigma(f) + \frac{\rho}{2} \int \int B_1(\hat{\Omega}_1, \hat{\Omega}_2) f(\hat{\Omega}_1) f(\hat{\Omega}_2) \mathrm{d}\hat{\Omega}_1 \mathrm{d}\hat{\Omega}_2 \tag{2.6}$$

where Z_N is the configuration integral, ρ is the number density N/V, B_1 is excluded volume, $f(\hat{\Omega})$ is the normalized orientational distribution function and $\sigma(\hat{\Omega})$ is the orientational entropy given by Equation 2.7.

$$\sigma(f) = \int f(\hat{\Omega}) \ln(4\pi f(\hat{\Omega}))\mathrm{d}\hat{\Omega} \tag{2.7}$$

For the purposes of this chapter, an isotropic distribution will be assumed, thus $f(\hat{\Omega})$ is constant (Equation 1.48). In this way, $\sigma(\hat{\Omega}) = 0$, and the Helmholtz free energy can be written as:

$$\frac{\beta A}{N} = \ln\left(\frac{\rho\nu}{4\pi}\right) - 1 - \frac{\rho}{2}\int\int B_1(\hat{\Omega}_1, \hat{\Omega}_2) f(\hat{\Omega}_1) f(\hat{\Omega}_2)\mathrm{d}\hat{\Omega}_1\mathrm{d}\hat{\Omega}_2 \tag{2.8}$$

where $\beta = 1/(k_B T)$, k_B is Boltzmann constant, T is the absolute temperature, ν is the de Broglie volume incorporating rotational and translational degrees of freedom.

In the case of hard potentials, that is, when particles cannot overlap, $-B_1$ equals the inaccessible volume for a particle due to the presence of a second particle. In the case of hard-spheres, $B_1 = -8v_{\text{sphere}}$, as shown in Figure 2.4. The light grey area represents the volume denied to the center of mass of the sphere, $-B_1 = v_{\text{exc}} = 4\pi d^3/3$.

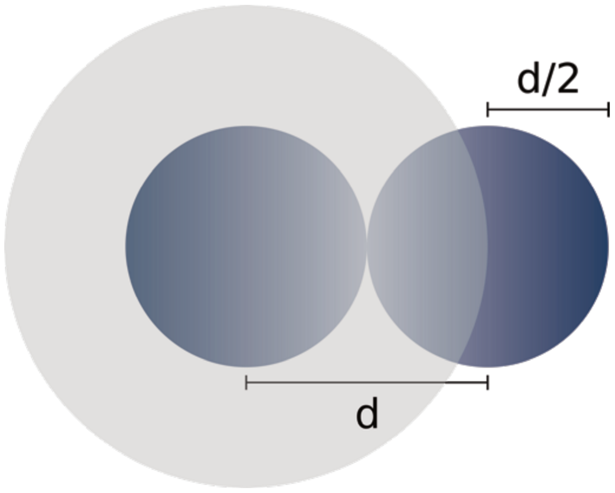


Figure 2.4: van der Waals excluded volume.

Based on the equation of state formulated by Carnahan and Starling (1969) for the hard-sphere model, Lee (1987) proposed a generalized expression for the Helmholtz free energy of nonspherical particles, shown in Equation 2.9.

$$\beta\frac{A^{\text{nonspherical}}}{N} = \ln\left(\frac{\rho\nu}{4\pi}\right) - 1 - \sigma(f) + \frac{1}{8}\frac{4\eta - 3\eta^2}{(1-\eta)^2}\left\langle\frac{v_{\text{exc}}}{v_0}\right\rangle_{\hat{\Omega}} \tag{2.9}$$

where v_{exc} is the excluded volume, v_0 is the volume of the particle and $\langle\rangle_{\hat{\Omega}}$ is the average over the orientations. The factor 1/8 was chosen in such a way that the expression is equal to the Carnahan and Starling (1969) original expression for hard spheres (Equation 1.12).

Parsons (1979) derived the expression for the excluded volume of the HGO potential, shown in Equation 2.10.

$$\frac{v_{\text{exc}}}{v_0} = \frac{8}{\sqrt{1-\chi^2}}\sqrt{(1-\chi^2\cos^2\theta_{12})} \tag{2.10}$$

where θ is the angle between the particles. Assuming an isotropic distribution and taking the average over orientations of Equation 2.10, A^{HGO} is derived from Equation 2.9:

$$\beta a^{\text{HGO}} = \ln\left(\frac{\rho\nu}{4\pi}\right) - 1 + \frac{(4-3\eta)\eta}{2(1-\eta)^2}\left(1 + \frac{\arcsin\chi}{\chi\sqrt{1-\chi^2}}\right) \tag{2.11}$$

2.2.2 Square-Well Potential - Gil-Villegas et al. (1997)

As previously mentioned , for the perturbed potential in Equation 2.3, we have chosen the spherical square-well potential:

$$u^{\text{SW}} = \begin{cases} +\infty, & \text{if} \quad r \leq \sigma \\ -\varepsilon, & \text{if} \quad \sigma < r \leq \lambda\sigma \\ 0, & \text{if} \quad r > \lambda\sigma \end{cases} \tag{2.12}$$

where ε is the well depth, λ is the attractive range, and $\sigma = \sqrt[3]{{\sigma_s}^2\sigma_e}$, which is formulated by equating the hard core volumes (or the packing fraction) of a sphere and of an ellipsoid of revolutions:

$$\frac{\pi\sigma^3}{6} = \frac{\pi\sigma_s^2\sigma_e}{6} \tag{2.13}$$

Applying the square-well potential in Equations (1.44b) and (1.44c):

$$a_1 = -2N\pi\rho\epsilon\sigma^3 \int_1^\lambda g^{(0)}(x)x^2\mathrm{d}x \tag{2.14a}$$

$$a_2 = -N\pi\rho\epsilon^2\sigma^3 k_B T \int_1^\lambda \left(\frac{\partial\rho g^{(0)}(x)}{\partial p}\right)_0 x^2\mathrm{d}x \tag{2.14b}$$

where $x = r/\sigma$. Rearranging Equation (2.14b):

$$\begin{aligned} a_2 &= -N\pi\rho\epsilon^2\sigma^3 k_B T \int_1^\lambda \left(g^{(0)}(x)\frac{\partial \rho}{\partial p} + \rho \frac{\partial g^{(0)}(x)}{\partial p} \right) x^2 \mathrm{d}x \\ &= -N\pi\rho\epsilon^2\sigma^3 k_B T \int_1^\lambda \left(g^{(0)}(x)\frac{\partial \rho}{\partial p} + \rho \frac{\partial g^{(0)}(x)}{\partial \rho}\frac{\partial \rho}{\partial p} \right) x^2 \mathrm{d}x \\ &= -N\pi\rho\epsilon^2\sigma^3 k_B T \frac{\partial \rho}{\partial p}\frac{\partial}{\partial \rho} \int_1^\lambda \rho g^{(0)}(x) x^2 \mathrm{d}x \end{aligned} \tag{2.15}$$

Introducing Equation (2.14a) into Equation 2.15:

$$a_2 = \epsilon\rho\frac{1}{2}k_B T \frac{\partial \rho}{\partial p}\frac{\partial a_1}{\partial \rho} = \frac{1}{2}\epsilon K_T \eta \frac{\partial a_1}{\partial \eta} \tag{2.16}$$

where K_T is the isothermal compressibility, which can be calculated with the Percus-Yevick expression for hard-sphere (Equation 2.17). Nonetheless, the molecular volume of an ellipsoid of revolution is used to calculate the packing fraction ($\eta = \rho\pi\sigma_s^2\sigma_e/6$).

$$K_T = \frac{(1-\eta)^4}{1+4\eta+4\eta^2} \tag{2.17}$$

To solve Equation (2.14a), we adopt the same strategy as Gil-Villegas et al. (1997), *i. e.*, the mean-value theorem (MVT) is applied to evaluate the integral. In the original work, the full function $g^{\mathrm{HS}}(\xi;\eta)$ is represented, which comes from the application of MVT, by its contact value evaluated at an effective packing fraction η_{eff}. So that $g^{\mathrm{HS}}(\xi;\eta)$ becomes $g^{\mathrm{HS}}(1;\eta_{\mathrm{eff}})$

$$a_1 = -2N\pi\varepsilon\sigma^3 g^{\mathrm{HS}}(1;\eta_{\mathrm{eff}}) \int_1^\lambda x^2 \mathrm{d}x = -4\eta\varepsilon(\lambda^3 - 1) g^{\mathrm{HS}}(1;\eta_{\mathrm{eff}}) \tag{2.18}$$

The Carnahan and Starling expression for the radial distribution function at the contact value is:

$$g^{\mathrm{HS}}(1;\eta_{\mathrm{eff}}) = \frac{1-\eta_{\mathrm{eff}}/2}{(1-\eta_{\mathrm{eff}})^3} \tag{2.19}$$

The parameters obtained by Gil-Villegas et al. (1997) are used to calculate η_{eff} (Equation 2.20).

$$\eta_{\text{eff}} = c_1\eta + c_2\eta^2 + c_3\eta^3 \tag{2.20a}$$

where the coefficients c_n are given by:

$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 2.25855 & -1.50349 & 0.249434 \\ -0.669270 & 1.40049 & -0.827739 \\ 10.1576 & -15.0427 & 5.30827 \end{bmatrix} \begin{bmatrix} 1 \\ \lambda \\ \lambda^2 \end{bmatrix} \tag{2.20b}$$

Although the form of a_1 and a_2 terms are the same as is in SAFT-VR SW (Gil-Villegas et al. (1997)), strictly, since the repulsive core is different in the two models and the terms are integrated from the contact distance, they should be different. The radial distribution function of the HGO, however, is a complicated function of orientations and the vector joining the particles centers of mass. To have a closed and more tractable formulation, we apply the decoupling approximation (Parsons, 1979) to map the HGO radial distribution as the hard-sphere one at the same packing fraction: $g^{\text{HGO}}(\vec{r}, \hat{\Omega}_1, \hat{\Omega}_2) = g^{\text{HS}}(r/\sigma, \eta)$. Such an assumption is exact at lower densities. In addition to that, since we have applied a spherical square-well potential as the repulsive part of the potential, the terms are the same as SAFT-VR SW.

With Equations 2.11, 2.18, and 2.16, Equation 2.3 can finally be evaluated. With an expression for the Helmholtz free energy, A, the fluid properties are derived through classical thermodynamics relations.

2.3 Results and Discussion

The proposed equation of state was applied to ethane and carbon dioxide, both small nonspherical molecules. For spherical molecules like methane, the proposed EoS and SAFT-VR SW are completely equivalent. The model parameters were optimized to fit vapor pressure and saturated liquid density data obtained in NIST (Linstrom and Mallard). Although taking into account supercritical derivative properties such as heat capacity and speed of sound improves the applicability of the fitted parameters (Lafitte et al., 2013); we have maintained the same properties used in the original optimization of SAFT-VR SW EoS (Gil-Villegas et al., 1997) for the sake of a fair comparison. The optimized parameters are shown in Table 2.1.

Table 2.1: Optimized parameters for ethane and carbon dioxide.

Compound	EoS	λ	ε/k_B / K	$\sigma_s{}^a$ / Å	σ_e / Å	m^b
ethane	This work	1.597	243.5	2.968	6.864	-
	SAFT-VR SW (Gil-Villegas et al., 1997)	1.448	241.8	3.788	-	1.3
carbon dioxide	This work	1.626	275.0	2.144	7.494	-
	SAFT-VR SW (Galindo and Blas, 2002)	1.516	179.3	2.786	-	2.0

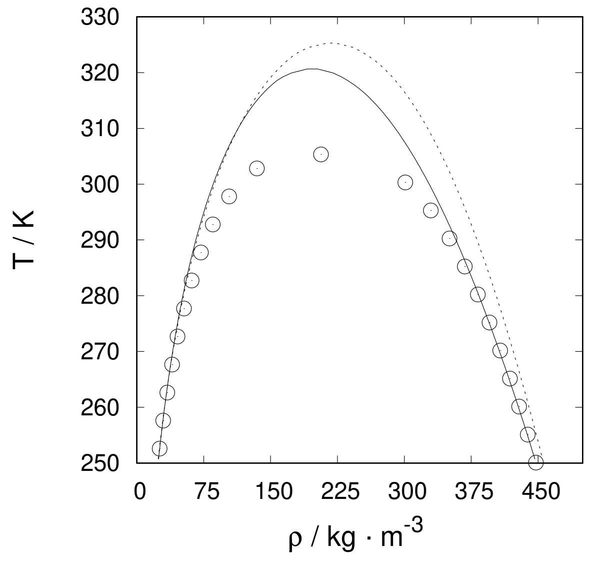
[a] for SAFT-VR SW, $\sigma_s = \sigma$.
[b] m stands for the number of spherical segments in a chain.

Figures 2.5 and 2.6 present the vapor-liquid equilibrium of ethane and carbon dioxide. The proposed equation of state correlates the coexistence curves of both ethane and carbon dioxide better than SAFT-VR SW when compared to NIST data. Both the proposed equation of state and SAFT-VR SW (Gil-Villegas et al., 1997) overpredict the critical point of ethane and carbon dioxide. Nevertheless, our proposed EoS predicts more accurately the values of critical temperature and critical pressure when compared to NIST data (Linstrom and Mallard), as shown in Table 2.2.

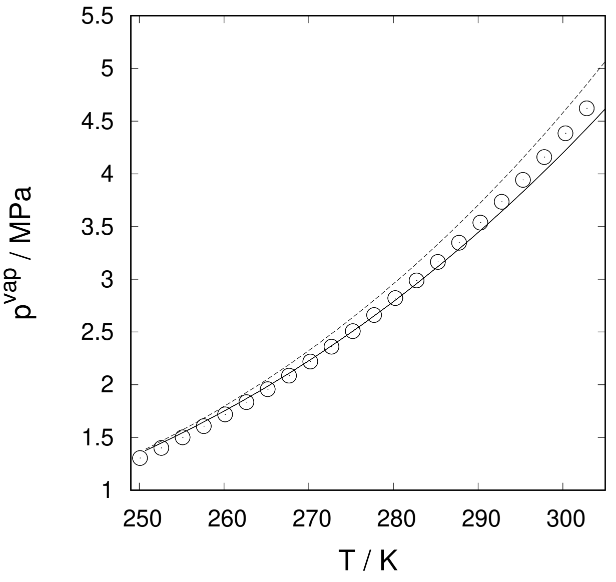
Table 2.2: Critical properties of ethane and carbon dioxide.

Compound		T_c / K	p_c / MPa
ethane	NIST	305.33	4.87
	This work	320.57	6.10
	SAFT-VR SW	325.33	7.43
carbon dioxide	NIST	304.12	7.38
	This work	315.49	8.52
	SAFT-VR SW	322.97	11.18

The reason behind the overprediction of the critical properties (temperature and pressure) for both models is twofold: the choice of a discrete perturbed potential as the square-well potential, and the truncation of the high temperature series expansion on the second term. Lafitte et al. (2013) showed that, with the inclusion of the third term in the expansion with a Mie potential, the prediction of the critical point is much more accurate. Likewise Sastre et al. (2018) observed that the higher the order of the terms introduced the lower the critical point obtained. Moreover, the prediction of the coexistence curve near the critical point is improved by higher order perturbation theory for square-well (Gil-Villegas and Benavides, 1996; Espíndola-Heredia et al., 2009) and Lennard-Jones (van Westen and Gross, 2017) fluids. Another strategy would be to take into account the critical point in the fitting procedure, but the correlation of the saturated liquid density would certainly be deteriorated.



(a)



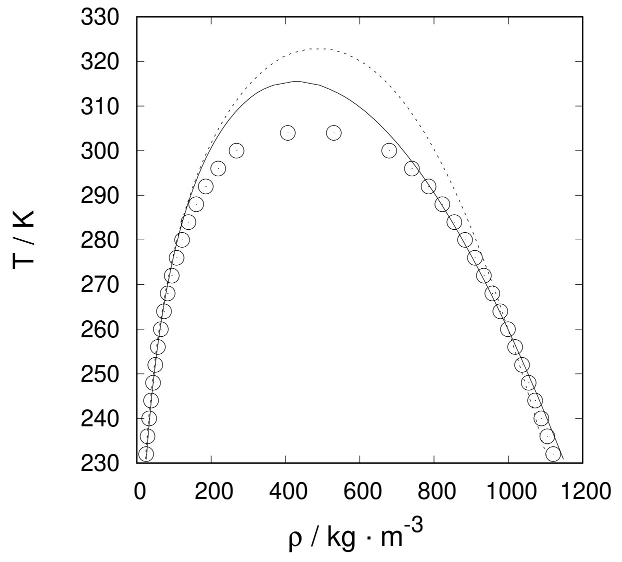
(b)

Figure 2.5: Vapor-liquid equilibrium for pure ethane: (a) coexistence curve, (b) vapor pressure as a function of temperature. Open symbols, NIST data (Linstrom and Mallard). Continuous lines, our proposed equation of state. Dotted lines, SAFT-VR SW (Gil-Villegas et al., 1997).

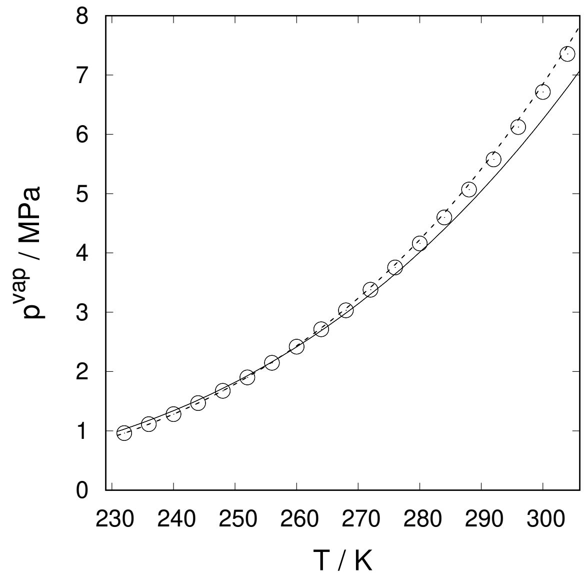
For the sake of a quantitative comparison, the Average Absolute Relative Deviation (AARD) was calculated:

$$\text{AARD}(\%) = \frac{1}{N_p}\sum_{i=1}^{N_p}\left|\frac{\varphi_i^{\text{NIST}} - \varphi_i^{\text{EoS}}}{\varphi_i^{\text{NIST}}}\right| \times 100\% \tag{2.21}$$

where N_p is the number of calculated points, φ^{NIST} is the NIST (Linstrom and Mallard)



(a)



(b)

Figure 2.6: Vapor-liquid equilibrium for pure carbon dioxide: (a) coexistence curve, (b) vapor pressure as a function of temperature. Open symbols, NIST data (Linstrom and Mallard). Continuous lines, our proposed equation of state. Dotted lines, SAFT-VR SW (Gil-Villegas et al., 1997) with optimized parameters from Galindo and Blas (2002).

value for a certain property φ, and φ^{EoS} is the property value calculated by the Equation of State.

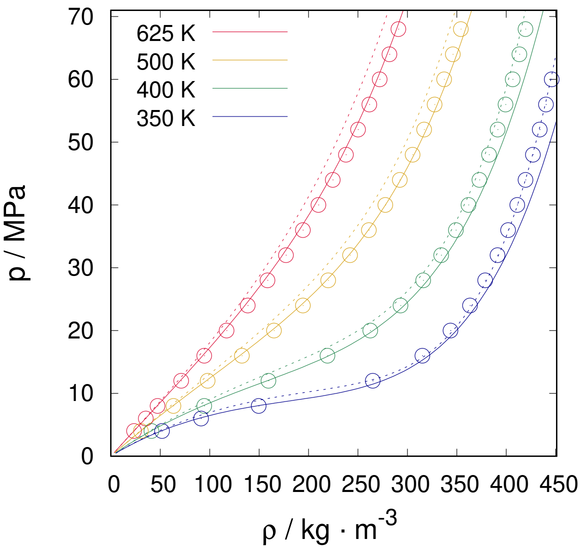
Table 2.3 presents the AARD values for vapor pressure, saturated liquid density, and saturated vapor density of both ethane and carbon dioxide calculated with the proposed EoS and with SAFT-VR SW. The vapor pressure of ethane is better correlated with the proposed equation of state; whereas the opposite is observed to carbon dioxide. Nevertheless, the saturated liquid density for both fluids is better correlated with the proposed EoS.

Table 2.3: Average Absolute Relative Deviation (%) for vapor pressure, saturated liquid density, and saturated vapor density.

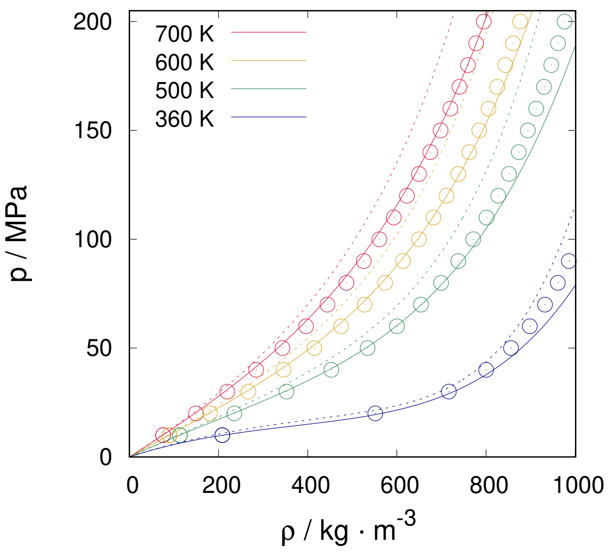
Compound		p^{vap}	ρ_l	ρ_v
ethane	This work	1.96	1.41	8.78
	SAFT-VR SW	5.07	6.08	7.96
carbon dioxide	This work	3.53	0.78	6.41
	SAFT-VR SW	0.76	2.26	7.95

The parameters of the proposed equation of state were adjusted solely to correlate saturated properties. The supercritical properties of ethane and carbon dioxide, however, can be used to assess the predictive power of the proposed model. Figure 2.7 presents the results for supercritical density of ethane and carbon dioxide. Overall the proposed EoS predictions are more accurate than those obtained with SAFT-VR SW, with the exception of ethane at a low temperature (350 K). SAFT-VR SW generally overpredicts pressure at a given density and temperature for both ethane and carbon dioxide.

Supercritical derivative properties of ethane and carbon dioxide, such as isochoric and isobaric heat capacities, speed of sound, Joule-Thomson coefficient, isothermal compressibility, and thermal expansion coefficient, were also investigated. Figures 2.8, 2.9, 2.10, 2.11, 2.12, and 2.13 show the results for the proposed EoS and SAFT-VR SW (Gil-Villegas et al., 1997), compared to NIST data (Linstrom and Mallard). The ideal gas isobaric heat capacity was calculated according to the empirical expression proposed by Passut and Danner (1972). The proposed EoS captures the trends observed for all thermodynamic derivative properties, with the exception of the isochoric heat capacity (Figure 2.8). Insights regarding the inadequacy of the models to quantitatively describe the c_v are given in Chapter 3 of this dissertation. The original SAFT-VR SW, however, describes qualitatively well only the speed of sound and the isothermal compressibility (Figures 2.10 and 2.12). Lafitte et al. (2007) underline that the SAFT VR Mie approach enhances the derivative properties prediction and, in another work, Lafitte et al. (2006) imply that a possible origin of the failure in describing these properties might be the use of the square-well potential. A deeper discussion with regard to that is also present in Chapter 3. Nonetheless, it is interesting to note that our approach was capable of significantly improving the calculation of derivative properties even while using the SW



(a)

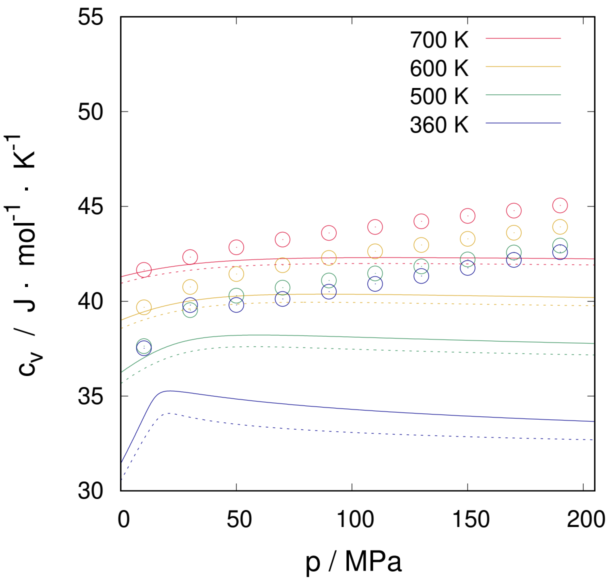


(b)

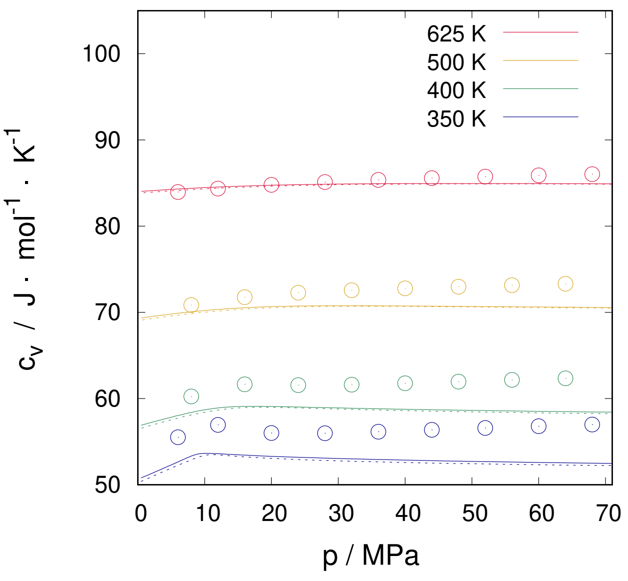
Figure 2.7: Pressure *versus* density at constant temperature for: (a) ethane, and (b) carbon dioxide. Open symbols, NIST data (Linstrom and Mallard). Continuous lines, our proposed equation of state. Dotted lines, SAFT-VR SW (Gil-Villegas et al., 1997).

potential as the perturbed potential.

Larger deviations are observed for the proposed EoS at high pressures and low temperatures. At these thermodynamic conditions, a high dense fluid is found, and the approximation made in the formulation of the equation of state in which the reference and the perturbed potential are treated with different molecular geometries is challenged. The replacement of the square-well potential by an anisotropic intermolecular potential in



(a)

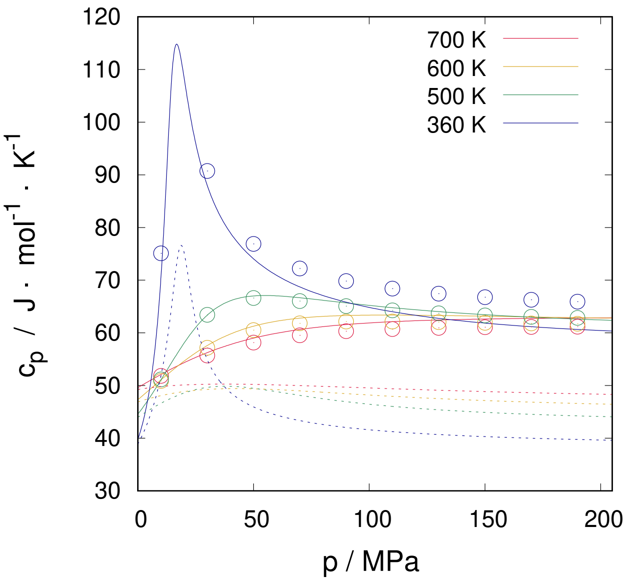


(b)

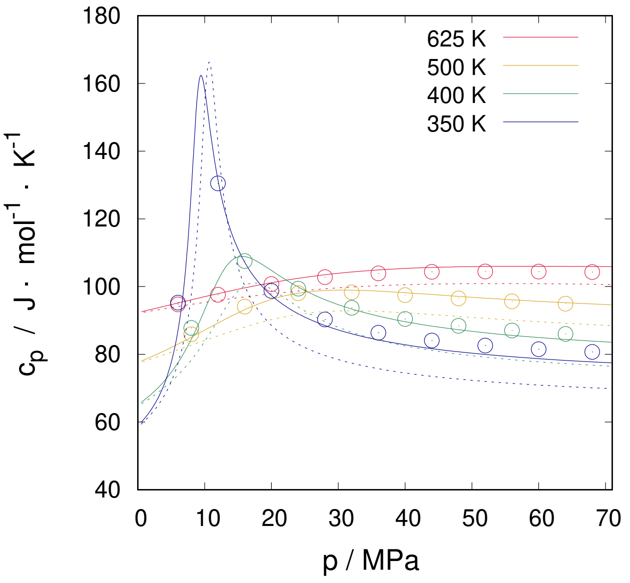
Figure 2.8: Isochoric heat capacity: (a) carbon dioxide (b) ethane Open symbols, NIST data (Linstrom and Mallard). Continuous lines, our proposed equation of state. Dotted lines, SAFT-VR SW (Gil-Villegas et al., 1997).

the attractive part of the perturbation theory might improve the prediction of derivative properties at these specific conditions. The decoupling approximation applied to formulate the HGO Helmholtz free energy also has a significant impact on the prediction of properties at higher densities. Since the DA takes $g(\vec{r}, \hat{\Omega}_1, \hat{\Omega}_2)$ as $g(r/\sigma_{\text{HGO}})$, it is exact at low densities where $g \simeq \exp(-\beta u_{\text{HGO}})$, but not at high ones.

The AARD values for the calculation of supercritical density and derivative properties



(a)

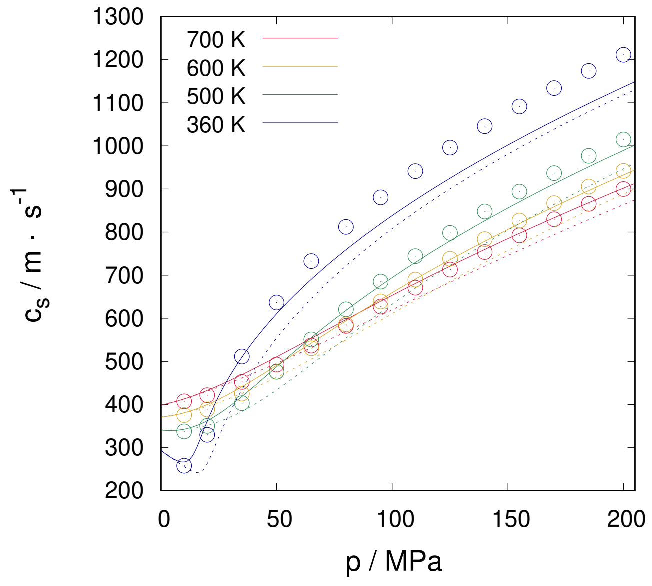


(b)

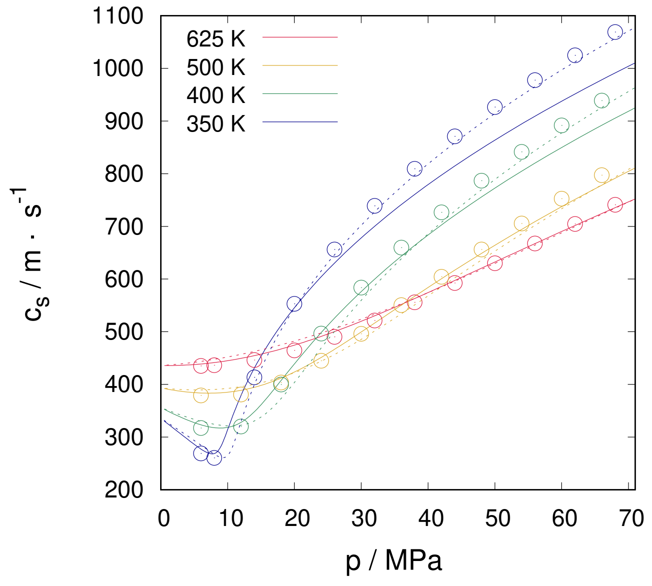
Figure 2.9: Isobaric heat capacity: (a) carbon dioxide (b) ethane Open symbols, NIST data (Linstrom and Mallard). Continuous lines, our proposed equation of state. Dotted lines, SAFT-VR SW (Gil-Villegas et al., 1997).

for ethane and carbon dioxide are shown in Table 2.4. With the exception of the carbon dioxide isothermal compressibility, the proposed EoS gives better predictions than SAFT-VR SW for all supercritical properties. The highest deviations are observed for the Joule-Thomson coefficient predictions. A thorough analysis shows that such high deviations are exclusively related to the deviation in the prediction of the inversion point.

The results obtained here by the proposed equation of state shows that somehow



(a)

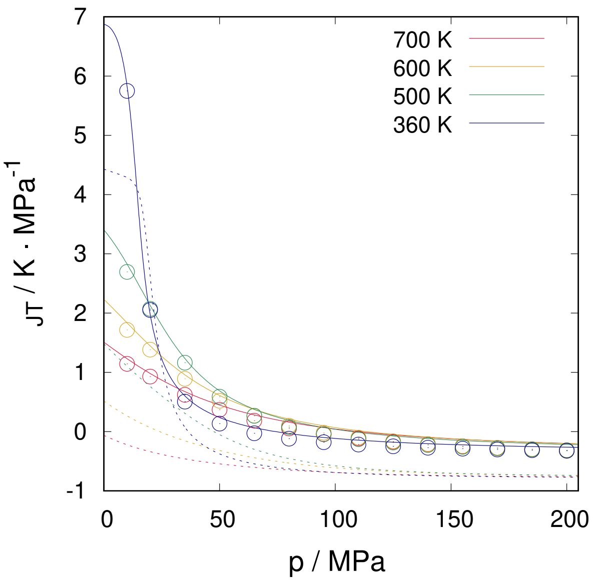


(b)

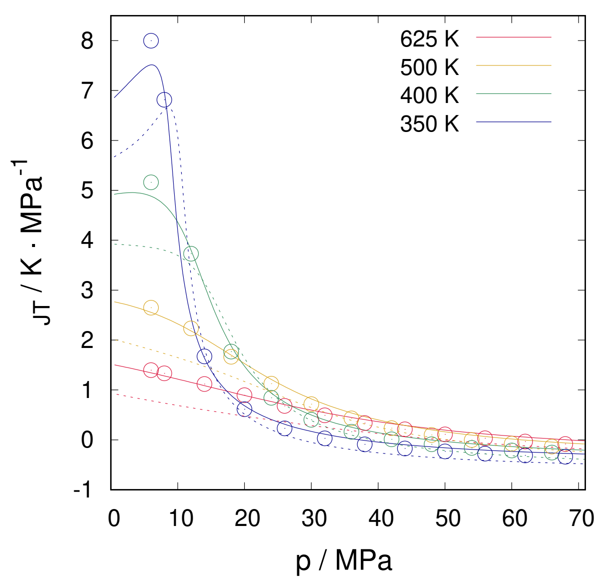
Figure 2.10: Speed of sound: (a) carbon dioxide (b) ethane Open symbols, NIST data (Linstrom and Mallard). Continuous lines, our proposed equation of state. Dotted lines, SAFT-VR SW (Gil-Villegas et al., 1997).

Table 2.4: Average Absolute Relative Deviation (%) for supercritical density, isochoric heat capacity, isobaric heat capacity, Joule-Thomson coefficient, speed of sound, thermal expansion coefficient, and isothermal compressibility.

Compound		ρ	c_v	c_p	μ_{JT}	c_s	α	k_T
ethane	This work	1.69	3.39	1.90	114.13	2.00	5.55	4.74
	SAFT-VR SW	3.86	3.63	7.10	157.34	2.34	10.25	5.28
carbon dioxide	This work	2.43	7.75	2.79	131.45	2.44	11.05	8.26
	SAFT-VR SW	6.88	9.29	24.91	724.19	6.43	36.32	3.64



(a)

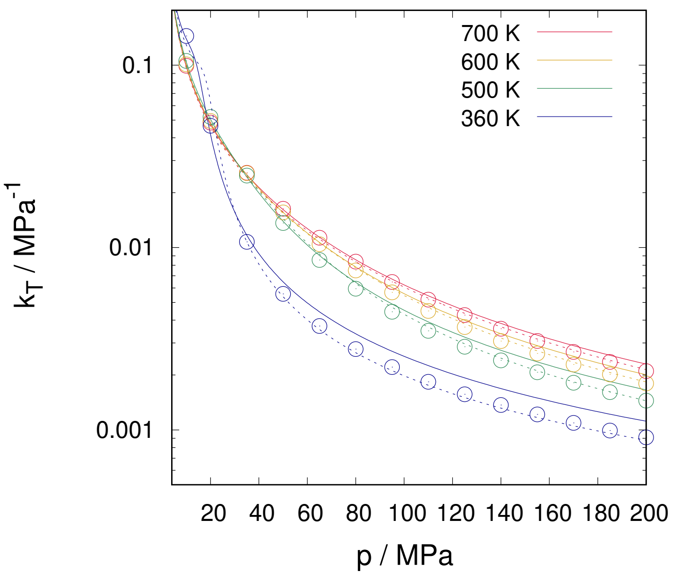


(b)

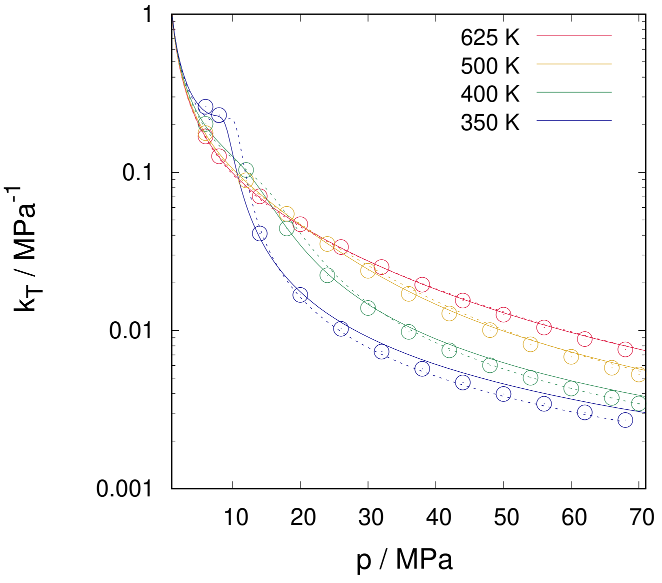
Figure 2.11: Joule-Thomson coefficient: (a) carbon dioxide (b) ethane Open symbols, NIST data (Linstrom and Mallard). Continuous lines, our proposed equation of state. Dotted lines, SAFT-VR SW (Gil-Villegas et al., 1997).

the assumption of an ellipsoidal geometry seems to be more adequate to represent these molecules than the original SAFT-VR SW (Gil-Villegas et al., 1997) approach for which the Helmholtz free energy of spherical segments forming a chain is calculated with Wertheim's first-order thermodynamic perturbation theory.

The proposed formulation has also the benefit of eliminating the apparent physical issue of a non-integer number of segments. Nevertheless, the fitted parameters for the



(a)

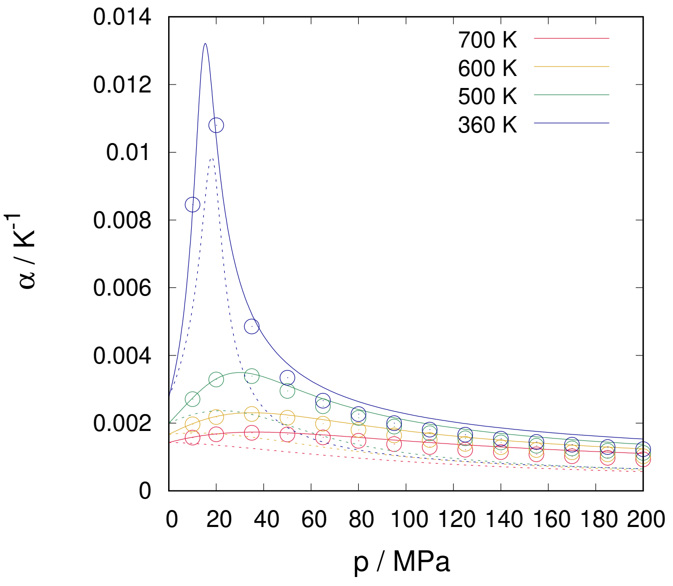


(b)

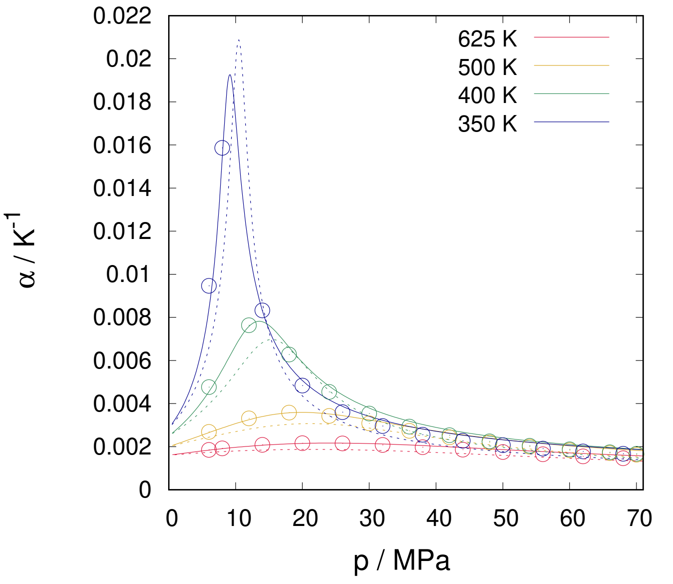
Figure 2.12: Isothermal compressibility: (a) carbon dioxide (b) ethane Open symbols, NIST data (Linstrom and Mallard). Continuous lines, our proposed equation of state. Dotted lines, SAFT-VR SW (Gil-Villegas et al., 1997).

proposed EoS assuming an ellipsoidal geometry must be physically sound. A simple way to check this is to compare the shape and the volume of the fitted ellipsoid with the molecular models for ethane and carbon dioxide.

The Transferable Potential for Phase Equilibria (TraPPE) (Martin and Siepmann, 1998) is a united atom molecular model using Lennard-Jones potential frequently applied in molecular simulations to calculate phase equilibrium and thermodynamic properties



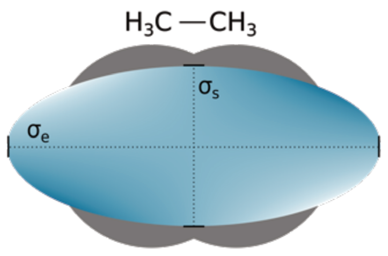
(a)



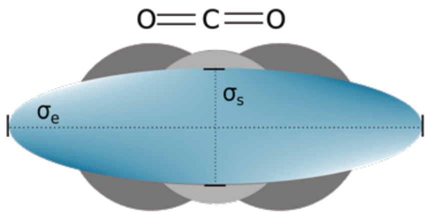
(b)

Figure 2.13: Thermal expansion: (a) carbon dioxide (b) ethane Open symbols, NIST data (Linstrom and Mallard). Continuous lines, our proposed equation of state. Dotted lines, SAFT-VR SW (Gil-Villegas et al., 1997).

(Aimoli et al., 2014). For TraPPE (Martin and Siepmann, 1998), ethane is represented as two spherical particles. The distance between these two particles is kept fix as 1.54 Å. Assuming the diameter of such spherical particles as the distance at which the intermolecular potential is zero, then the volume of a single ethane molecule is 43.7 Å^3. The volume of the ellipsoid, calculated with the fitted parameters shown in Table 2.1, is 31.7 Å^3. Therefore, the ratio between the volume of the fitted ellipsoid and the volume obtained with TraPPE molecular model is 0.73. For carbon dioxide, TraPPE model (Potoff and Siepmann, 2001) gives a volume of 22.9 Å^3, while the fitted ellipsoid has a volume of 18.0 Å^3, giving a ratio between the volume of the fitted ellipsoid and the volume obtained with TraPPE molecular model of 0.79. Being the calculated volume ratios for ethane and carbon dioxide close to 1, one might conclude that the fitted parameters provide a physically meaningul geometry for both ethane and carbon dioxide, as also illustrated in Figure 2.14.



(a) Ethane



(b) Carbon Dioxide

Figure 2.14: Illustrative comparison between TraPPE molecular model for ethane(Martin and Siepmann, 1998) and carbon dioxide(Potoff and Siepmann, 2001) and the ellipsoidal geometry obtained with the fitted parameters for the proposed EoS.
.

An explanation for why the ellipsoids volumes are lower than the ones calculated with TraPPE model might reside in a compensation for the use of the square-well potential as the perturbed term. Taking the potential well-depth proposed by Berne and Pechukas (1972) and analyzing the four site molecule example in Gay and Berne (1981), one may see that the well-depth of nonspherical particles tend to be larger than that for spherical particles. Thus, the attractive term on perturbation theory might be larger if one applies an anisotropic potential. Therefore, a smaller volume in the proposed EoS reduces the repulsive contribution to be compatible to the attractive contribution given by the square-well perturbed potential.

2.4 Conclusion

We have formulated an alternative equation of state for nonspherical molecules based on a perturbation theory, in which the Hard Gaussian Overlap model is used as the reference potential and the perturbed contribution is given by a square-well potential. The vapor-liquid equilibrium for ethane and carbon dioxide was successfully correlated with the proposed EoS. Moreover, the proposed EoS predicts more accurate critical properties, when compared to SAFT-VR SW. For suprecritical thermodynamic derivative properties, the proposed EoS generally provides better estimates than the original SAFT-VR SW for both ethane and carbon dioxide. The results obtained with the proposed EoS might imply that the choice of a single ellipsoid to represent such small molecules is an adequate alternative as to fitting a non-integer number of segments, as commonly done in SAFT framework using Wertheim's first-order thermodynamic perturbation theory. In addition to that, we showed that this approach is capable of significantly improving the derivative properties prediction even using the square-well as the perturbed potential. The comparison between the ellipsoid volume resulting from the fitted parameters with molecular models such as TraPPE shows that the proposed EoS parameters are physically meaningful. Finally, the extension of the proposed model for larger and associating molecules remains to be addressed.

" ... pendurou-se-me uma ideia no trapézio que eu tinha no cérebro. Uma vez pendurada, entrou a bracejar, a pernear, a fazer as mais arrojadas cabriolas de volatim, que é possível crer. Eu deixei-me estar a contemplá-la. Súbito, deu um grande salto, estendeu-se os braços e as pernas, até tomar a forma de um X: decifra-me ou devoro-te." [1]

— Machado de Assis, Memórias Póstumas de Brás Cubas

3 A top-down approach for ellipsoids to investigate the isochoric heat capacity prediction.

The content of this chapter was reprinted (adapted) from Lopes, J. T. and Franco, L. F. M. Prediction of isochoric heat capacity: Discrete versus continuous potentials. Fluid Phase Equilib. 506, 112380 (2020). https://doi.org/10.1016/j.fluid.2019.112380 Lopes and Franco (2020)

[1]' ... an idea took hold of the trapeze that I used to carry about in my head. Once it had taken hold, it flexed its arms and legs and began to do the most daring acrobatic feats one can possible imagine. I just stood and watched it. Suddenly it made a great leap, extended its arms and legs until it formed an X, and said, "Decipher me or I devour thee."' - Epitaph of a Small Winner, Machado de Assis.

3.1 Introduction

In Chapter 2, theoretical approximations were made to develop a molecular based equation of state for the intermolecular potential in Equation 3.1, modeling carbon dioxide and ethane as ellipsoidal particles.

$$u(\vec{r},\hat{\Omega}_1,\hat{\Omega}_2) = \begin{cases} \infty, & \text{if} \quad |r| < \sigma_{\text{HGO}}(\vec{r},\hat{\Omega}_1,\hat{\Omega}_2) \\ -\varepsilon, & \text{if} \quad \sigma_{\text{HGO}}(\vec{r},\hat{\Omega}_1,\hat{\Omega}_2) \leq |r| < \lambda\sigma_{\text{sphere}} \\ 0, & \text{if} \quad |r| \geq \lambda\sigma_{\text{sphere}} \end{cases} \tag{3.1}$$

In this chapter, the same intermolecular potential is studied in Monte Carlo molecular simulations, and the parameters optimized using the equation of state are used as a force-field, in a fashion known as “top-down” approach. The advantage of molecular simulation is that the solution for the chosen potential is exact, that is, no approximation is needed. In this way, comparing the results of the equation of state and of the molecular simulations, it is possible to discriminate the influence of the theoretical approximations and of the choice of the intermolecular potential on the structural and thermodynamic properties prediction.

As recognized in Chapter 2, neither of the studied equations of state (Gil-Villegas et al., 1997; Lopes and Franco, 2019) were able to capture the trends of the isochoric heat capacity. Lafitte et al. (2006) suggested that deficiencies of SAFT-VR SW (Gil-Villegas et al., 1997) in predicting derivative properties could result from the choice of the SW model as intermolecular potential. We, however, showed on the previous chapter that the calculation of derivative properties could be improved solely by changing the repulsive potential to an anisotropic one, while keeping the attractive spherical square-well potential as the perturbed part. Llovell and Vega (2006) have examined separately the different contributions of soft-SAFT (Blas and Vega, 1997) (reference fluid, chain, and association) to the total derivative properties, which allowed a better understanding of the significance of each contribution to the calculation of each property. In a similar fashion, Maghari and Sadeghi (2007), proposing a modified version of the SAFT-BACK (Chen and Mi, 2001), have analyzed the different contributions to the residual isochoric heat capacity (hard-convex body, chain, chain dispersion, and dispersion).

Although equations of state derived for both discrete and continuous potentials have been extensively studied, a thorough investigation through molecular simulation to check the effects of theoretical approximations on the heat capacity was yet to be addressed. Therefore, besides the proof-of-concept of our top-down approach to generate a coarse-grained force field for carbon dioxide as an ellipsoidal particle, in this chapter we aim at shedding some light in the effects, validity, and limitations of theoretical approximations and of the choice of intermolecular potentials in the prediction of the isochoric heat capacity. CO_2 has been chosen as a case study for its extreme relevance in our today's society, and because carbon dioxide is a reasonably small non-polar molecule for which parameters for the chosen models are available in the open literature.

To investigate the isochoric capacity behavior, the NVT Monte Carlo simulations are more suitable, since, in this case, the residual isochoric molar heat capacity c_v^R can be calculated from the potential energy fluctuations extracted from the simulation, as show in Equation 3.2.

$$\frac{c_v^{\mathrm{R}}}{k_B} = \frac{\beta^2}{N}\left(\langle U^2\rangle - \langle U\rangle^2\right) \tag{3.2}$$

where U is the total intermolecular potential energy.

We compare results of Monte Carlo simulations using SAFT-γ Mie and the force field proposed in this work to SAFT-VR SW (Gil-Villegas et al., 1997), SAFT-VR Mie (Lafitte et al., 2013), and our recently proposed equation of state (Lopes and Franco, 2019). The choice of these three models among so many versions of SAFT is due to the similar fundamental basis in their derivation that allows a fairer comparison. The Mie potential is expressed as:

$$\frac{u^{\mathrm{Mie}}(r)}{\varepsilon} = \frac{\lambda_r}{\lambda_r - \lambda_a}\left(\frac{\lambda_r}{\lambda_a}\right)^{\frac{\lambda_a}{\lambda_r - \lambda_a}}\left[\left(\frac{\sigma}{r}\right)^{\lambda_r} - \left(\frac{\sigma}{r}\right)^{\lambda_a}\right] \tag{3.3}$$

where ε is the potential well-depth, σ is the distance at which the potential becomes zero, and λ_r and λ_a are the repulsive and attraction exponents, respectively.

3.2 Equations of state

The theoretical development and formulation of SAFT-VR SW and our proposed EoS is addressed in Chapter 2.

3.2.1 SAFT-VR Mie

Lafitte et al. (2013) formulated a new version of SAFT for both repulsive and attractive potentials of variable range. Mie potential (Equation 3.3) is applied to model spherical segments interaction, as a strategy to develop a more versatile and accurate equation of state, especially in the description of derivative properties.

The authors express the molar Helmholtz free energy of the repulsive potential as the free energy of a temperature-dependent hard-sphere system with an effective diameter:

$$d(T) = \int_0^\sigma \left[1 - \exp\left(-\beta u^{\mathrm{Mie}}(r)\right)\right] \mathrm{d}r \tag{3.4}$$

Thus an effective packing fraction to calculate the reference contribution is defined:

$$\eta_{\mathrm{ehs}} = \frac{\rho \pi d(T)^3}{6} \tag{3.5}$$

where η_{ehs} is the packing fraction of an effective hard-sphere system.

In the SAFT-VR MIE equation of state (Lafitte et al., 2013), an improved form of the second-order perturbation term a_2 is used. In Equation 1.45, Υ is a function of $\pi\sigma^3/6$, and also depends on the repulsive and attractive exponents λ_r and λ_a of the Mie potential. The function coefficients were adjusted to match accurate values of a_2 obtained via molecular simulations. In addition to that, an empirical expression to account for higher-order terms Zwanzig's high-temperature expansion is included, represented by a_3^{Mie}. Finally, for non-associating fluids, the residual molar Helmholtz free energy is given by:

$$\beta a^{\mathrm{R}} = \beta a^{\mathrm{HS}} + \beta a_1^{\mathrm{Mie}} + \beta^2 a_2^{\mathrm{Mie}} + \beta^3 a_3^{\mathrm{Mie}} + \beta a^{\mathrm{CHAIN}} \tag{3.6}$$

The reader is referred to the original work (Lafitte et al., 2013) for more details regarding the calculation of these terms. The intermolecular potential parameters for carbon dioxide were taken as (Avendaño et al., 2011): $\varepsilon/k_B = 361.69$ K, $\sigma = 3.741$ Å, $\lambda_r = 23.0$ and $\lambda_a = 6.66$. In this case, CO_2 molecule is represented as a single-site spherical particle.

3.3 Monte Carlo Simulations

A Monte Carlo algorithm is a probabilistic interpretation of mathematical problems in which the solution is found by a stochastic sampling of the system. Different types of Monte Carlo algorithms are used to solve complex mathematical problems. In our case, the Metropolis Monte Carlo algorithm is used to solve the statistical mechanics integrals introduced in Chapter 1.

3.3.1 Metropolis Monte Carlo

From statistical mechanics, a property Φ of a system is equal to its average over all possible microstates (Equation 1.3). The Monte Carlo method consists of selecting random configurations (Γ_τ), and averaging Φ over these systems, rather than following the dynamic evolution of the system:

$$\langle\Phi\rangle_{\mathrm{NVT}} = \frac{\sum\limits_{\tau=1}^{\tau_{\max}} \Phi(\Gamma_\tau)\exp(-\beta U(\Gamma_\tau))}{\sum\limits_{\tau=1}^{\tau_{\max}} \exp(-\beta U(\Gamma_\tau))} \tag{3.7}$$

where $\tau_{\max}$ is the number of sampled microstates, and Γ_τ the configuration of each state. If every possible state had the same probability of occurring, *i.e.*, if the probability distribution was uniform, the configurations could be picked without criteria, and the ensemble average would be a simple average of the properties over the sampled microstates. Nonetheless, the probability distribution is not uniform, and it is related to the Boltzmann factor, as presented on Chapter 1.

One can say that the Boltzmann factor $\exp(-\beta U)$ weights the property in a given configuration: if the state has a large probability, the property calculated in that given calculation should be more significant to the ensemble average. For a close-packed configuration, a large number of sampled microstates would give a very large and positive potential energy due to particles overlap ($U_m \to \infty$). As a consequence, the Boltzmann factor would be very small, and the calculated property of that given configuration would have very low weight. Metropolis et al. (1953) presented a method to sample the microstates in such a way that, by the end of the simulation, the number of occurrences of each state is consistent with its probability. As a consequence, an average over the sampled microstates gives the ensemble average. The authors proposed a procedure to set up a Markov chain; provided that enough steps are taken, the system goes to the desired distribution (equilibrium distribution) regardless of the initial configuration.

From Equation 1.2, the probability in the canonical ensemble that a system in state m goes to a state n is given by:

$$\frac{P(\Gamma_n)}{P(\Gamma_m)} = \frac{\exp(-\beta U_n)}{\exp(-\beta U_m)} = \frac{\exp(-\beta U_m)\exp(-\beta(U_n - U_m))}{\exp(-\beta U_m)} = \exp(-\beta \Delta U_{mn}) \tag{3.8}$$

A flowchart of the method is given in Figure 5.1. The method is initiated with a system in a random state m (configuration Γ_m and total potential energy U_m). A new state Γ_n is generated by moving randomly one particle. The potential energy U_n is calculated for this new state. If $\Delta U_{mn} < 0$, the number of systems in state n is greater than m ($\exp(-\beta\Delta U_{mn}) > 1$), therefore the new configuration Γ_n is accepted. If the change in the potential energy is greater than 0 ($\exp(-\beta\Delta U_{mn}) < 1$), the movement is accepted with the probability of $\frac{P(\Gamma_n)}{P(\Gamma_m)}$ by generating a random number ζ between 0 and 1, and comparing it with the ratio. If the selected random number is less than $\exp(-\beta\Delta U_{mn})$, the movement is accepted, otherwise the particle remains at its old position Γ_m. The procedure is repeated as many times as necessary to reach the equilibrium distribution.

A new state can be generated solely by moving one particle. A trial move is the generation of a new state by changing the position and orientation of a particle, which can be defined as a cycle. In the present chapter, we try to move all the N particles per step, therefore each step consists of N cycles.

Change in position

A new position r^{new} is generated by applying Equation 3.9.

$$r^{\text{new}}(i) = r^{\text{old}}(i) + (2\zeta - 1)\Delta r^{\max} \tag{3.9}$$

where $i = 1$ to 3 and represents each axis, ζ is a number randomly selected from a uniform distribution between zero and one, and $\Delta r^{\max}$ is the maximum displacement allowed, which is an adjustable parameter.

Change in orientation

To be able to define and adjust a maximum rotation, rotation quaternions are applied to obtain a new orientation. A succinct introduction to quaternions is given in appendix B.1. The procedure is described below:

1. a random angle θ is generated in a similar fashion of Equation 3.9.
2. a random axis $a = [a_1, a_2, a_3]$ is obtained by creating a random unit vector.
3. a rotation quaternion is calculated (Equation B.2).
4. a new orientation is obtained by applying Equation B.4.

There are two important points to take into consideration when applying the method. If the potential energy U_n is too large (when $\beta\Delta U_{mn} > 75$, for example), the Boltzmann factor would be very small. Hence, to avoid underflow and to save computational time, the move should be immediately rejected. This is one way of avoiding considerable overlap of particles too (Allen and Tildesley, 2017). The second point is related to the maximum allowed displacement of the particle. If it is too large, many moves would be rejected, and the phase space would be poorly explored. On the other hand, if it is too small, most of the moves would be accepted; nonetheless, systems would be highly correlated, and it would take a long time to explore the phase space. To avoid these extremes, the maximum displacement should be adjusted in function of the percentage of movements accepted.

3.3.2 Calculation of nematic order parameter and phase director

The nematic order parameter s is obtained from Equation 1.46. The phase director n is defined as the vector that maximizes s. The computation of the order parameter, however, is tricky, since n is unknown in the beginning of the calculation. The Q tensor method (Zannoni, 1979) is applied to calculate s and the phase director n. Q is defined as a symmetric traceless 3×3 matrix, in such a way that Equation 3.10 is true.

$$s = \left\langle \frac{3}{2}(\hat{n} \cdot \hat{\Omega})^2 - \frac{1}{2} \right\rangle = n^T \cdot \langle Q \rangle \cdot n = \sum_{i=1}^{3} \sum_{j=1}^{3} n_i n_j Q_{ij} \delta_i \delta_j \tag{3.10}$$

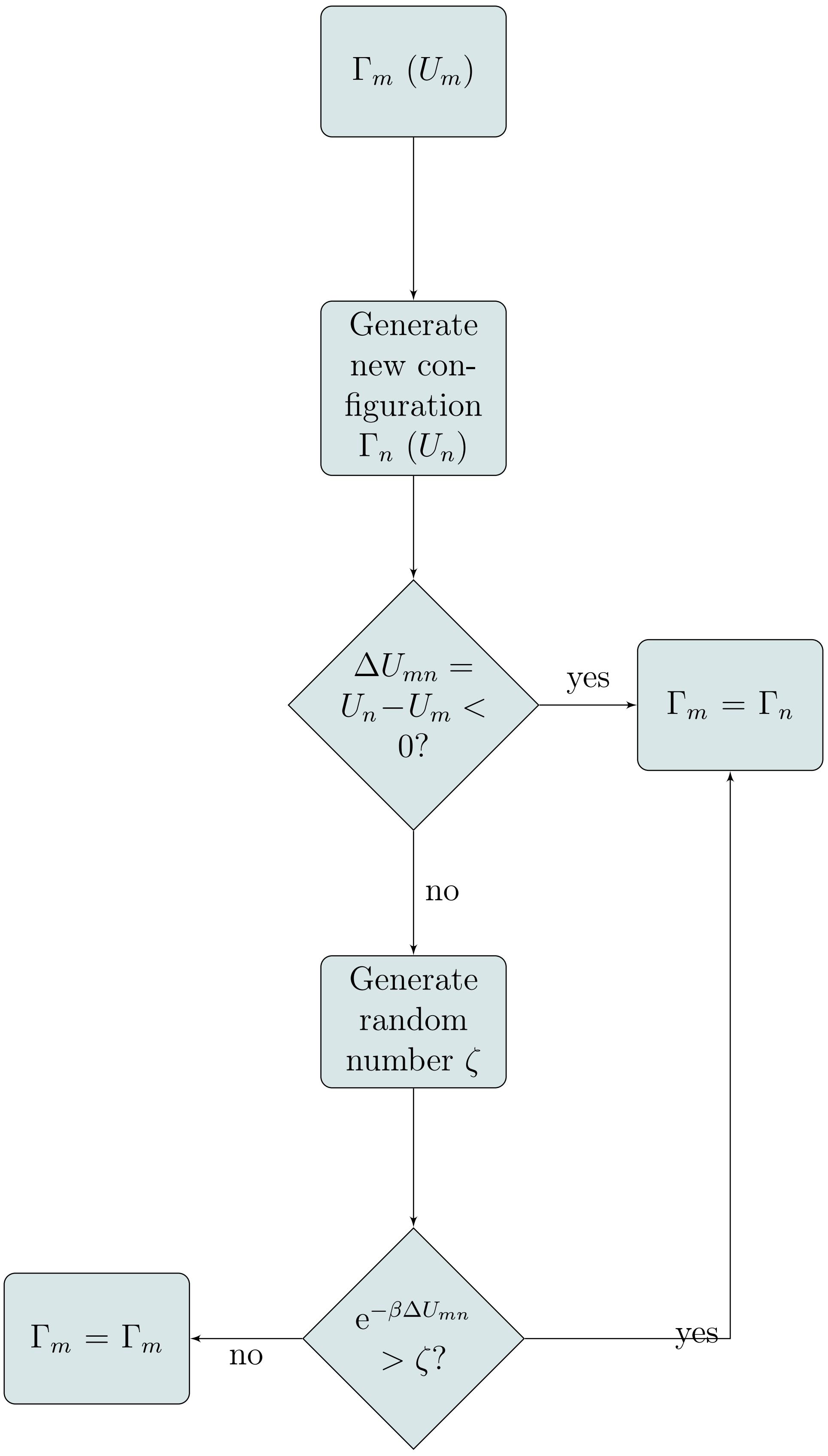

Γ_m (U_m)
Generate new configuration Γ_n (U_n)
$\Delta U_{mn} = U_n - U_m < 0$?
yes
$\Gamma_m = \Gamma_n$
no
Generate random number ζ
$e^{-\beta \Delta U_{mn}} > \zeta$?
no
$\Gamma_m = \Gamma_m$
yes

Figure 3.1: Metropolis method flowchart for a cycle in the NVT ensemble.

where $n = \sum_{i=1}^{3} n_i\delta_i$ and $Q = \sum_{i=1}^{3}\sum_{j=1}^{3}\langle Q_{ij}\rangle\delta_i\delta_j$. The right hand side of Equation 3.10 is written as:

$$\begin{aligned} n^T \cdot \langle Q\rangle \cdot n = \;& n_1^2\langle Q_{11}\rangle + n_1n_2\langle Q_{12}\rangle + n_1n_3\langle Q_{13}\rangle \quad + \\ & n_2n_1\langle Q_{21}\rangle + n_2^2\langle Q_{22}\rangle + n_2n_3\langle Q_{23}\rangle \quad + \\ & n_3n_1\langle Q_{31}\rangle + n_3n_2\langle Q_{32}\rangle + n_3^2\langle Q_{33}\rangle \end{aligned} \tag{3.11}$$

Since Q is symmetric, $Q_{ij} = Q_{ji}$, replacing Equation 3.11 into Equation 1.46 gives:

$$\begin{aligned} s =\;& \left\langle \frac{3}{2}\left(n_1^2\Omega_1^2 + n_2^2\Omega_2^2 + n_3^2\Omega_3^2 + 2n_2n_3\Omega_2\Omega_3 + 2n_1n_2\Omega_1\Omega_3\right)\right\rangle - \frac{1}{2} \\ =\;& n_1^2\langle Q_{11}\rangle + 2n_1n_2\langle Q_{12}\rangle + 2n_1n_3\langle Q_{13}\rangle \quad + \\ & n_2^2\langle Q_{22}\rangle + 2n_2n_3\langle Q_{23}\rangle + n_3^2\langle Q_{33}\rangle \end{aligned} \tag{3.12}$$

Equating the coefficients of Equation 3.12, Q_{ij} could be in principle be defined as:

$$Q_{ij} = \frac{3}{2}\Omega_i\Omega_j \tag{3.13}$$

If Q were given as in Equation 3.13, a $-\frac{1}{2}$ would still be missing on the right hand side of Equation 3.12 for both sides to be equal. Hence, we redefine Q as:

$$Q_{ij} = \frac{3}{2}\Omega_i\Omega_j - \delta_{ij} \tag{3.14}$$

Substituting Equation 3.14 into Equation 3.12:

$$s = \left\langle \frac{3}{2}\left(n_1^2\Omega_1^2 + n_2^2\Omega_2^2 + n_3^2\Omega_3^2 + 2n_2n_3\Omega_2\Omega_3 + 2n_1n_2\Omega_1\Omega_3\right)\right\rangle - \frac{n_1^2}{2} - \frac{n_2^2}{2} - \frac{n_3^2}{2} \tag{3.15}$$

Since n is a unit vector, $-\frac{n_1^2}{2} - \frac{n_2^2}{2} - \frac{n_3^2}{2} = -\frac{1}{2}(n_1^2+n_2^2+n_3^2) = -\frac{1}{2}$. Hence, the definition of Q in Equation 3.14 satisfies Equation 3.10.

As stated earlier in this section, n maximizes s, thus we apply the method of the Lagrange multipliers to find n with the constraint that it should be a unit vector:

$$g_1 = n^T \cdot n - 1 = 0 \tag{3.16}$$

$$\mathcal{L} = n^T \cdot \langle Q\rangle \cdot n - \lambda(n^T \cdot n - 1) = \sum_{i=1}^{3}\sum_{j=1}^{3} n_in_jQ_{ij}\delta_i\delta_j - \lambda\left(\sum_{k=1} 3n_k^2 \quad - 1\right) \tag{3.17}$$

$$\nabla \mathcal{L} = \sum_{i=1}^{3}\sum_{j=1}^{3}\sum_{l=1}^{3} \frac{\partial}{\partial n_l} n_i n_j Q_{ij}\delta_l - 2\lambda \sum_{k=1}^{3} n_k \delta_k \tag{3.18}$$

From Equation 3.18:

$$2\sum_{i=1}^{3}\sum_{j=1}^{3} n_j Q_{ij}\delta_i = 2\lambda \sum_{k=1}^{3} n_k \delta_k \tag{3.19}$$

An equivalent expression for Equation 3.19 is:

$$Qn = \lambda n \tag{3.20}$$

Therefore, n is the eigenvector, and λ the eigenvalue of Q. Multiplying both sides of equation 3.19 by n, and bearing in mind that n is a unit vector:

$$\sum_{i=1}^{3}\sum_{j=1}^{3} n_i n_j Q_{ij} = \lambda \sum_{k=1}^{3} n_k^2 = \lambda \tag{3.21}$$

Comparing Equations 3.21 and 3.10, we infer that $s = \lambda$, *i.e.*, the order parameter is the eigenvalue of Q.

Since Q is a symmetric 3x3 matrix, we apply the method proposed by Smith (1961) to find its the eigenvalues.

$$\begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} m + 2\sqrt{p}\cos\phi \\ m - \sqrt{p}(\cos\phi + 3\sqrt(3)\sin\phi) \\ m - \sqrt{p}(\cos\phi - 3\sqrt(3)\sin\phi) \end{bmatrix} \tag{3.22}$$

where

$$\begin{aligned} m &= \frac{\mathrm{tr}(Q)}{3} \\ q &= \frac{1}{2}\det(Q - mI) \\ p &= \frac{1}{6}\sum_{i=1}^{3}\sum_{j=1}^{3}(Q_{ij} - m\delta_{ij})^2 \\ \phi &= \frac{1}{3}\tan^{-1}\frac{p^3 - q^2}{q} \end{aligned} \tag{3.23}$$

Once s is found, which is the largest eigenvalue, the phase director n is determined by applying the Gauss elimination method.

3.3.3 Simulation Details

Monte Carlo simulations with 864 particles were carried out in the canonical ensemble. We have run simulations with 15×10^4 cycles of 864 steps each. The properties were averaged over 75×10^3 production cycles divided into 5 blocks. The code is provided in the appendix B.2.1.

3.4 Results and Discussion

The potential parameters for carbon dioxide were optimized in the original works to fit saturated liquid density and vapor pressure, hence the models can be used in a predictive way to calculate the isochoric heat capacity. Table 3.4 summarizes the values of parameters of the models.

Table 3.1: Optimized Parameters for Carbon Dioxide

EoS	λ_r	λ_a^a	$\epsilon/k_b/K$	$\sigma_s^b/$	$\sigma_e/$	m^c
SAFT-VR SW (Galindo and Blas, 2002)	-	1.5157	179.27	2.7864	-	2
HGO + SW (Lopes and Franco, 2019)	-	1.626	275.0	2.144	7.494	-
SAFT-VR MIE (Avendaño et al., 2011)	23.0	6.66	-	3.741	-	1

[a] for SAFT-VR SW and HGO + SW, $\lambda_a = \lambda$.
[b] for SAFT-VR SW and SAFT-VR MIE, $\sigma_s = \sigma$.
[c] m stands for the number of spherical segments in a chain.

Figures 3.2 and 3.3 present the supercritical isochoric heat capacity for carbon dioxide as function of density at 360 K and 700 K, respectively. NIST data (Linstrom and Mallard) are taken as a reference for comparison. Calculated results using equations of state (SAFT-VR SW, HGO + SW, and SAFT-VR Mie) and force fields (SAFT-γ Mie and HGO + SW) show that all the tested models fail in predicting quantitatively the reference values for the heat capacity.

At a higher temperature, the results from molecular simulations agree quite accurately to the results obtained with the equations of state from which their parameters were derived. At a lower temperature, however, some discrepancies emerge, and molecular simulations do a better job when compared to the reference values. A possible explanation for this could be the truncation of higher-order perturbation terms considered in the development of the equations of state. In Zwanzig's perturbation theory (Zwanzig, 1954), the perturbed contribution to the Helmholtz free energy is written as an expansion on

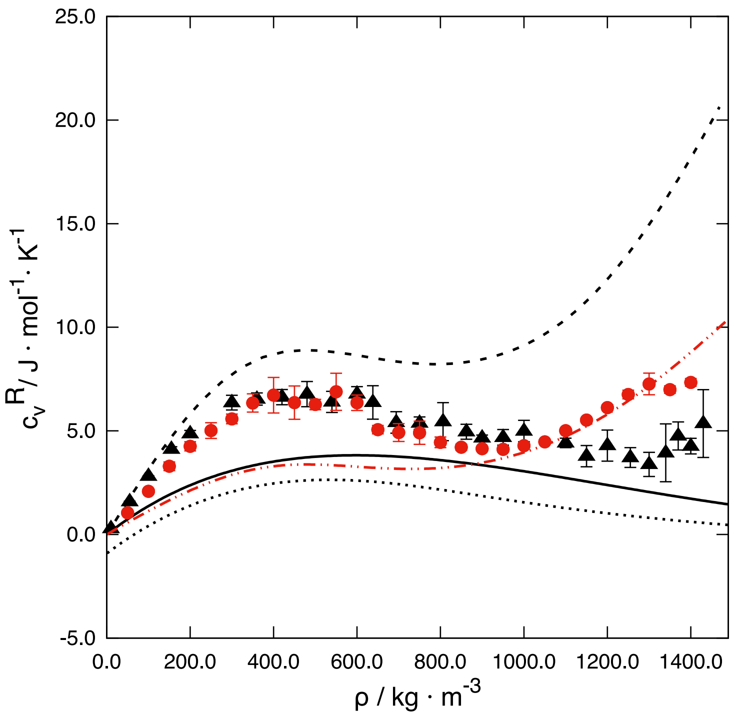


Figure 3.2: Isochoric heat capacity for carbon dioxide at 360 K. Dashed line, NIST data (Linstrom and Mallard). Closed black triangles, MC simulations for HGO + SW force field. Closed red circles, MC simulations for SAFT-γ Mie force field (Avendaño et al., 2011). Dash-double-dotted red line, SAFT-VR Mie equation of state (Lafitte et al., 2013). Continuous black line, HGO + SW equation of state (Lopes and Franco, 2019). Dotted line, SAFT-VR SW equation of state (Gil-Villegas et al., 1997; Galindo and Blas, 2002).

the inverse of temperature, β:

$$A^{(1)} = \sum_{n=1}^{+\infty} \frac{\omega_n}{n!} (-\beta)^{n-1} \tag{3.24}$$

where $A^{(1)}$ is the perturbed contribution to the Helmholtz free energy.

Therefore, as temperature decreases, the contribution from high-order terms increases. Since all the used equations of state consist of a truncation of such an expansion, they fail to provide an exact value of the Helmholtz free energy, especially at lower temperatures. Molecular simulations, however, prescind from such an approximation, hence they provide exact values for a given potential model. One interesting finding is that, taking the same parameters fitted using the equation of state, molecular simulations give better results. This is a valuable advantage for force fields based on top-down approaches.

SAFT-VR SW provides a negative value for the residual heat capacity at the limit of zero density. This seems a bit odd, since at this limit the residual heat capacity should be zero. Such an inconsistency is a consequence of an approximation taken in the

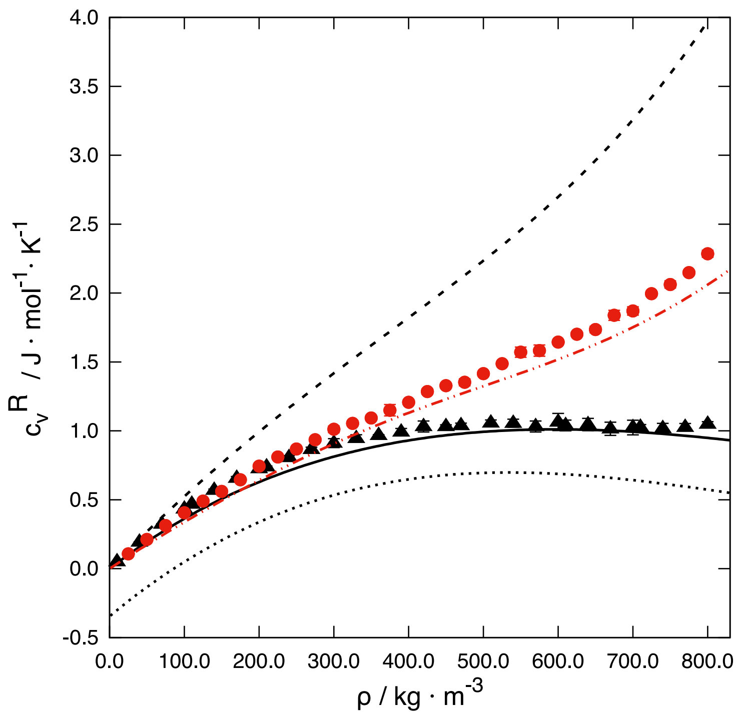


Figure 3.3: Isochoric heat capacity for carbon dioxide at 700 K. Dashed line, NIST data (Linstrom and Mallard). Closed black triangles, MC simulations for HGO + SW force field. Closed red circles, MC simulations for SAFT-γ Mie force field (Avendaño et al., 2011). Dash-double-dotted red line, SAFT-VR Mie equation of state(Lafitte et al., 2013). Continuous black line, HGO + SW equation of state (Lopes and Franco, 2019). Dotted line, SAFT-VR SW equation of state (Gil-Villegas et al., 1997; Galindo and Blas, 2002).

calculation of the chain contribution to the Helmholtz free energy. At the low density limit, $a^{\text{CHAIN}} \to 0$, and consequently $y^M(\sigma) \to 1$. Nevertheless, taking the zero density limit in the expression proposed by the authors in the original work (Gil-Villegas et al., 1997), $y^M(\sigma) \to (1+\beta\varepsilon)/\exp(\beta\varepsilon)$, which gives a negative value for the chain contribution to the heat capacity. At high temperatures, or low values of ε, such an inconsistency is less pronounced.

Overall, SAFT-γ Mie force field provides the best predictions for the heat capacity. At 360 K, HGO + SW equation of state is unable to capture the increasing trend of the heat capacity at high densities. The same happens to SAFT-VR SW. Besides that for SAFT-VR SW and HGO + SW the perturbed potential is a discrete potential and that for SAFT-VR Mie is a continuous potential, the reference potentials are different as well. Whereas for SAFT-VR Mie the reference potential is calculated with an effective temperature-dependent diameter, which emulates somehow a softer repulsion, for SAFT-VR SW and HGO + SW the reference potential is athermal. For SAFT-γ Mie and HGO + SW force fields, a similar analysis is possible, since SAFT-γ Mie has a soft repulsive potential and HGO + SW has a hard repulsive potential. The isochoric heat capacity is obtained by the second derivative of the Hemlholtz free energy with respect to temperature. Therefore, the contribution of an athermal reference potential to the heat capacity vanishes.

Figures 3.4 and 3.5 present the residual isochoric heat capacity for carbon dioxide at 360 K and 700 K respectively, calculated by SAFT-VR Mie equation of state (Lafitte et al., 2013), expliciting the different contributions of the repulsive reference and the attractive perturbed potentials. The contributions of each term on the perturbed temperature expansion are also presented. At high densities, and at high temperature, the repulsive contribution is quite relevant to the heat capacity. Since both SAFT-VR SW and HGO + SW lack such a contribution, the athermal repulsive contribution might explain why they fail to give even a qualitative description of the heat capacity at high densities. Boshkova and Deiters (2010) and Cañas-Marín et al. (2019) have investigated the role of soft repulsion upon the prediction of the characteristic curves of Brown (Brown, 1960). The authors point out that at high densities and temperatures the magnitude of the repulsion have a significant impact on the thermodynamics, since the closeness of molecules and collision at high speed in such conditions forces the molecules into repulsive regions of their pair potential.

The attractive contribution also seems to play an important role in the heat capacity, especially at a lower temperature. The second-order term seems to be the most relevant one for the attractive contribution. Anyway, as the density increases, the importance of the first-order term also increases. For both SAFT-VR SW and HGO + SW, the second derivative of the first-order term with respect to temperature vanishes, which means that only the second order contributes to heat capacity.

In HGO + SW formulation (Lopes and Franco, 2019), Barker-Henderson macroscopic compressibility approximation (MCA) for the second-order perturbation term is applied. Zhang (1999) proposed an improvement for the second-order perturbation contribution. Applying such a modified second-order perturbation theory to HGO + SW (Equation 1.45, section 1.1.2), and optimizing the potential parameters, a considerable improvement is observed in the prediction of the heat capacity as shown in Figures 3.6 and 3.7. But still HGO + SW is unable to capture the trend of increase in the heat capacity at higher densities. In contrast, Lafitte et al. (2013) used a more generic expression for Υ that

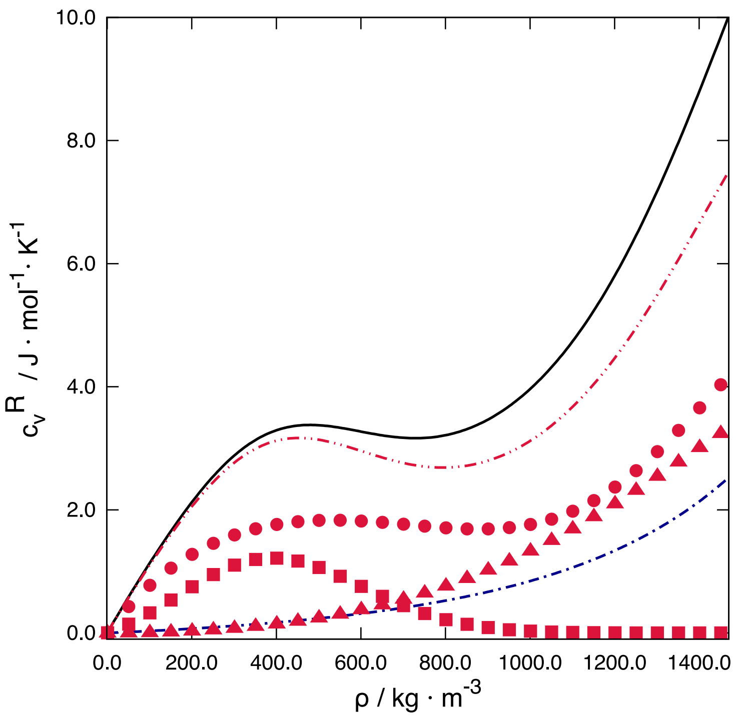


Figure 3.4: Different contributions of the SAFT-VR Mie (Lafitte et al., 2013) equation of state to the residual heat capacity of carbon dioxide at 360 K. Continuous black line, residual heat capacity. Dash-dotted blue line, repulsive contribution to heat capacity. Dash-double-dotted red line, attractive contribution to heat capacity. Triangles, first-order contribution to heat capacity. Circles, second-order contribution to heat capacity. Squares, higher-order contribution to heat capacity.

was fitted to a_2 values of molecular simulations and also depends on the soft repulsive core exponent. They showed that their approach captures the a_2 simulation complex non-monotonic trend at high densities, hence, this probably explains the behavior of the a_2 contribution to the c_v in Figures 3.4 and 3.5.

Another approximation made in the formulation of HGO + SW equation of state (Lopes and Franco, 2019) is that the fluid is isotropic, and therefore there is no preferred orientation. This assumption simplifies the formulation of the Helmholtz free energy, since the isotropic orientational distribution is uniform. Such an approximation is impossible to be tested with the equation of state itself, but applying a top-down approach generating a coarse-grained force field it is possible to check the validity of such an approximation. Figure 3.8 presents the order parameter $\langle S \rangle$ as a function of the density. $S \rightarrow 0$ for isotropic phases, $S \rightarrow 1$, for nematic phases and $S \approx 0.6$ for the isotropic-nematic phase transition (Mottram and Newton, 2014). $\langle S \rangle$ increases with density, as expected, since the orientational entropy tends to be smaller at higher densities due to the geometric limitations. Nonetheless, $\langle S \rangle$ is pretty close to zero even for extremely high densities,

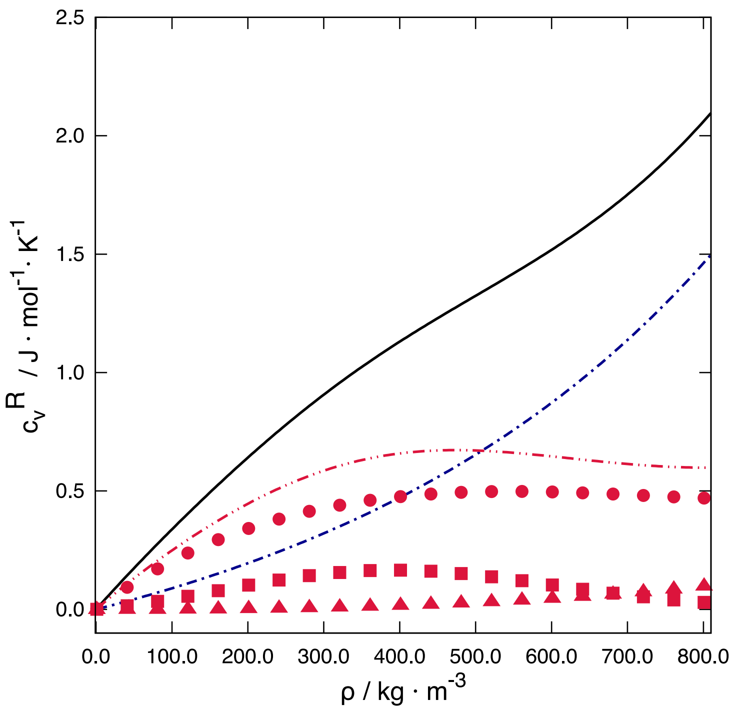


Figure 3.5: Different contributions of the SAFT-VR Mie (Lafitte et al., 2013) equation of state to the residual heat capacity of carbon dioxide at 700 K. Continuous black line, residual heat capacity. Dash-dotted blue line, repulsive contribution to heat capacity. Dash-double-dotted red line, attractive contribution to heat capacity. Triangles, first-order contribution to heat capacity. Circles, second-order contribution to heat capacity. Squares, higher-order contribution to heat capacity.

which leads us to conclude that the isotropic orientational distribution is a reasonable approximation in the theoretical development of this equation of state.

3.5 Conclusion

The isochoric heat capacity of supercritical carbon dioxide has been calculated using three different equations of state (SAFT-VR SW (Gil-Villegas et al., 1997; Galindo and Blas, 2002), HGO + SW (Lopes and Franco, 2019), and SAFT-VR Mie (Lafitte et al., 2013)) and two different coarse-grained force fields (SAFT-γ Mie (Avendaño et al., 2011) and HGO + SW). The role of the repulsive reference potential and the attractive perturbed potential in the prediction of this derivative property has been investigated, as well as the differences between the use of discrete or continuous models for the intermolecular potential. Molecular simulations prove that the truncation of the temperature expansion on the perturbed potential affects the prediction of heat capacity at low temperatures. An athermal reference potential, naturally employed when dealing with discrete potentials,

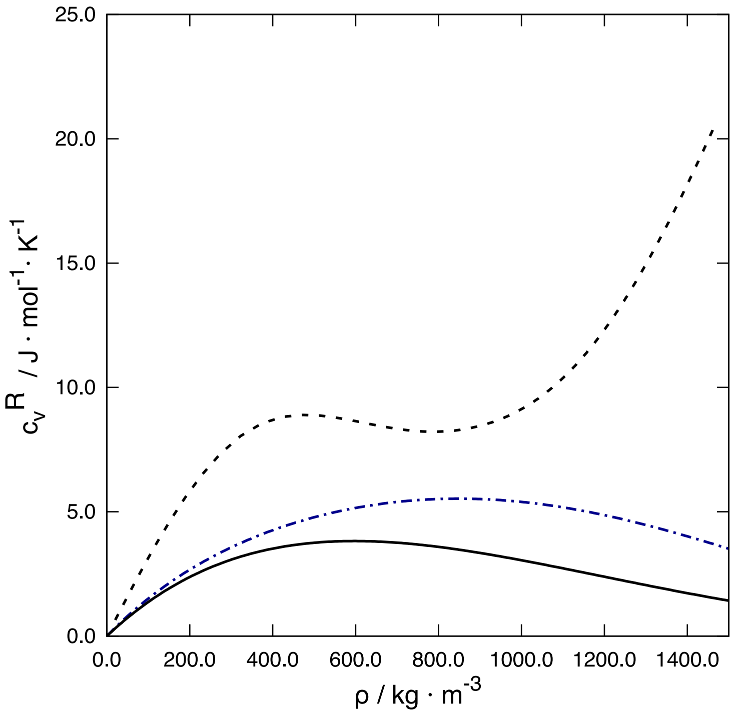


Figure 3.6: Isochoric heat capacity for carbon dioxide at 360 K. Dashed line, NIST data (Linstrom and Mallard). Continuous black line, original HGO + SW equation of state. Dash-dotted blue line, HGO + SW equation of state using Zhang's correction to the macroscopic compressibility approximation.

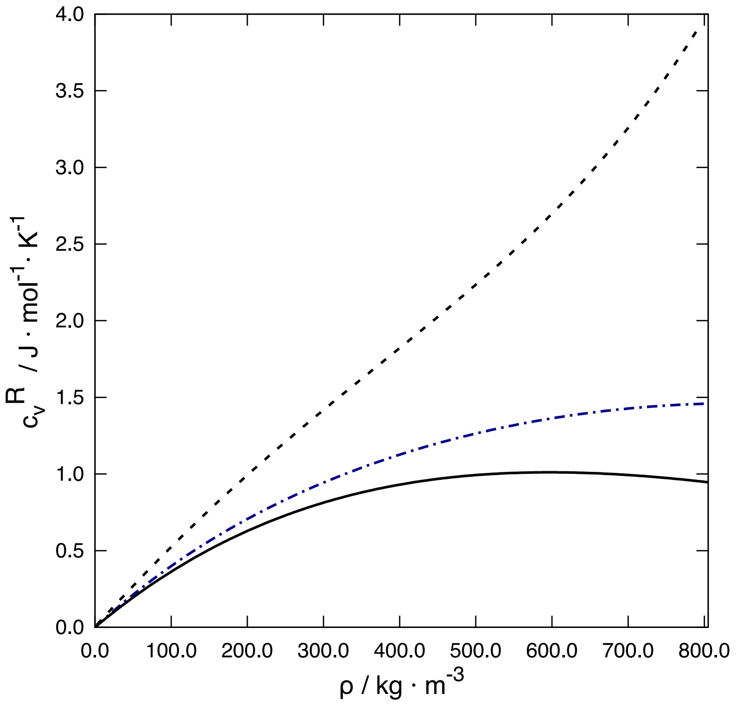


Figure 3.7: Isochoric heat capacity for carbon dioxide at 700 K. Dashed line, NIST data (Linstrom and Mallard). Continuous black line, original HGO + SW equation of state. Dash-dotted blue line, HGO + SW equation of state using Zhang's correction to the macroscopic compressibility approximation.

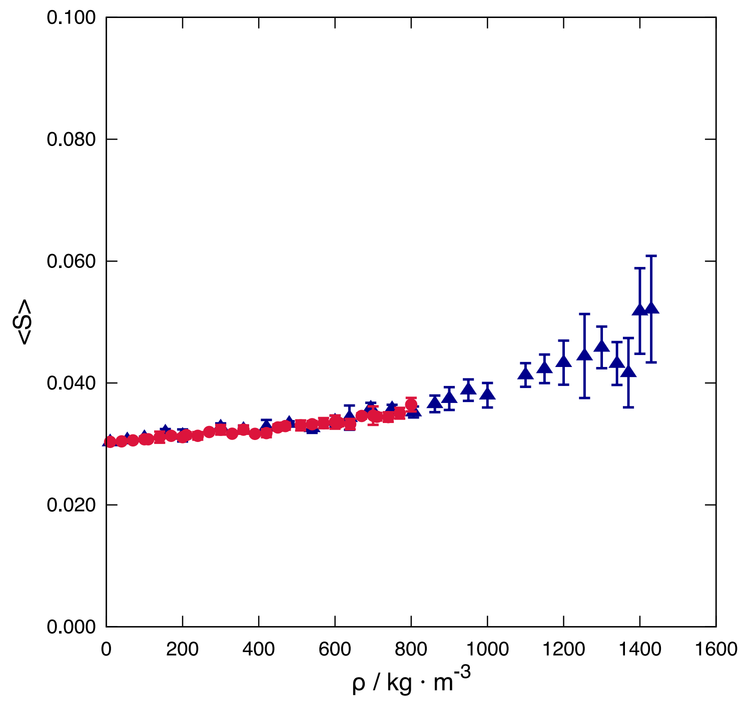


Figure 3.8: Order parameter for HGO + SW force field as a function of density at different temperature. Blue closed triangles, 360 K. Red closed circles, 700 K.

causes a great loss of accuracy, especially at high densities. The macroscopic compressibility approximation applied in the calculation of the second-order perturbation term is found to affect to a great extent the prediction of heat capacity behavior at high densities. SAFT-γ Mie coarse-grained force field, based on a continuous potential, is the most accurate model, among the studied ones, for predicting carbon dioxide heat capacity. The results of this investigation suggest that a possible promising path to improve the accuracy of molecular-based equations of state in the prediction of derivative properties might be the development of models that account for higher-order terms of the perturbation theory, besides the application of continuous intermolecular potentials.

"Everything is physics and math."

— Katherine Johnson

4 Equations of state for ellipsoidal, cylindrical, and spherocylindrical particles

4.1 Introduction

On Chapter 2 we have shown that, substituting the spherical segment and chain contribution in the original SAFT approach for a single contribution of an ellipsoidal segment, one could improve the thermodynamic properties prediction of small molecules such as carbon dioxide and ethane. On this Chapter, we test the same approach for longer molecules (n-alkanes up to octane and n-perfluoroalkanes) and disk-like molecules (benzene and toluene). We include Zhang's correction to the Barker-Henderson's Macroscopic Compressibility Approximation, as discussed on Chapter 3. Furthermore, we also test the performance of equations of state for different geometries such as cylindrical and spherocylindrical particles.

To take into account the nonsphericity of molecules, Chen and Kreglewski (1977) combined the equation of state for hard convex bodies formulated by Boublik (1974) with the dispersion term developed by Alder et al. (1972) and introduced the BACK EoS (Boublik-Alder-Chen-Kreglewski). A parameter α characterizes the degree of non-sphericity of particles in BACK EoS. The model, however, does not attribute any specific geometry to the particle and, even though it predicts better critical properties of small molecules when compared to SAFT approach, it cannot be used for longer chains.

Pfohl and Brunner (1998) combined SAFT with BACK to model supercritical solvent with the latter, as a small convex body, and the other molecules as a chain of spherical segments. Chen and Mi (2001) completely combined SAFT and BACK (SAFT-BACK) to describe long chain fluids, where the molecule is described as a set of nonspherical segments. It was shown that the SAFT-BACK EoS improves critical points when compared to the SAFT approach (Dargahi and Jafari, 2015).

Among others, when compared to SAFT-BACK, the main difference of our approach is that we eradicate the chain contribution altogether, describing the whole molecule as a nonspherical segment. The model for the spherocylinders is very similar to the one proposed by Williamson and Guevara (1999) with some small differences.

4.2 Formulation

The formulation of the equations of state in this Chapter follows the same approach as previously described in Chapters 2 and 3, applying the modification proposed by Zhang (1999) ($\Upsilon = 8.23\eta^2$ in Equation 1.45, section 1.1.2). To develop the equations of state for cylinders and spherocylinders, we use the excluded volume expressions formulated by Onsager (1949):

$$\langle v_{\mathrm{C}}^{\mathrm{exc}} \rangle_\theta = \frac{\pi}{2}\left(L^2 D + \frac{1}{2} L D^2 (\pi + 3) + \frac{\pi}{4} D^3\right) \tag{4.1}$$

$$\langle v_{\mathrm{SC}}^{\mathrm{exc}} \rangle_\theta = \frac{4\pi}{3} D^3 + 2\pi D^2 L + + \frac{D L^2 \pi}{2} \tag{4.2}$$

where $v_{\mathrm{C}}^{\mathrm{exc}}$ and $v_{\mathrm{SC}}^{\mathrm{exc}}$ are the excluded volumes of the cylinder and spherocylinder, respectively, $<>_\theta$ is the average over orientations, θ is the angle between two particles, L and D are the length and the diameter of the particle.

Applying the excluded volumes in Equation 2.9 and considering only the residual contribution:

$$\beta a^{\mathrm{C}} = \frac{1}{8}\frac{4\eta - 3\eta^2}{(1-\eta)^2}\left(2L/D + \pi + 3 + \frac{\pi}{2L/D}\right) \tag{4.3}$$

$$\beta a^{\mathrm{SC}} = \frac{1}{8}\frac{4\eta - 3\eta^2}{(1-\eta)^2}\left[\frac{12\left(\frac{4}{3} + 2L/D + \frac{(L/D)^2}{2}\right)}{3L/D + 2}\right] \tag{4.4}$$

where a is the molar Helmholtz free energy. The general equation of state can be written as:

$$a^{\mathrm{R}} = a^{\mathrm{nonspherical}} + a_1^{\mathrm{sw}}\beta + a_2^{\mathrm{sw}}\beta^2 \tag{4.5}$$

where $a^{\mathrm{nonspherical}}$ is either a^{HGO} (considering only the residual part of Equation 2.11), a^{HC} or a^{HSC}.

4.3 Results

The equations of state for ellipsoids (HGO), cylinders (HC), and spherocylinders (HSC) were used to calculate the vapor-liquid equilibrium for carbon dioxide, benzene, toluene, n-alkanes and n-perfluoroalkanes. The parameters λ, ϵ, σ_{sphere}, and the aspect ratio, L/D, were optimized to fit vapor pressure and saturated liquid density. L/D is equivalent to σ_e/σ_s in the case of the ellipsoids and σ_{sphere} is the diameter of a sphere with the same volume of the particle. The optimized parameters are outlined in Table 4.1. The SAFT-VR SW parameters were taken from Gil-Villegas et al. (1997), with exception of the parameters for carbon dioxide, that were taken from Galindo and Blas (2002). The results were compared to NIST data (Linstrom and Mallard).

The adjusted aspect ratios are larger for longer molecules, and for benzene and toluene L/D is lower than one, characterizing oblates/disks. In this sense, with exception of the HSC model for the aromatics, the models are physically sound (Figure 4.1).

The HGO anisotropy parameter χ (Equation 2.4) is lower than zero for oblates and larger than zero for prolates. For the same absolute value of χ, however, there is no difference in the calculations for negative or positive values (Equation 2.11). That is, for the calculations, it does not matter whether the particle is a prolate or an oblate, and χ is the relevant value rather than the aspect ratio. For cylinders, however, there is no

Table 4.1: Parameters optimized to fit vapor pressure and saturated liquid density data from NIST (Linstrom and Mallard).

compound	EoS	λ	ϵ / k_B/K	σ / Å	L/D^a	compound	EoS	λ	ϵ / k_B/K	σ / Å	L/D^a
CH_4	HGO	1.513	152.603	3.549	0.604	C_8H_{18}	HGO	1.466	764.922	5.650	4.999
	HC	1.558	150.000	3.489	0.878		HC	1.589	685.456	5.396	6.990
	HSC	1.461	159.823	3.610	0.217		HSC	1.455	770.294	5.681	4.999
	SAFT-VR SW	1.444	168.800	3.670	1.000		SAFT-VR SW	1.574	250.300	3.945	3.300
C_2H_6	HGO	1.546	269.066	3.917	2.750	CO_2	HGO	1.627	290.557	3.180	4.398
	HC	1.479	290.558	4.016	1.728		HC	1.594	299.599	3.220	4.248
	HSC	1.516	265.322	3.997	1.430		HSC	1.511	321.679	3.318	3.281
	SAFT-VR SW	1.448	241.800	3.788	1.300		SAFT-VR SW	1.516	179.270	2.786	2.000
C_3H_8	HGO	1.543	351.964	4.297	3.276	C_6H_6	HGO	1.472	613.440	4.721	0.298
	HC	1.531	356.953	4.313	2.809		HC	1.510	590.141	4.666	0.238
	HSC	1.527	356.851	4.321	2.620		HSC	1.444	636.152	4.762	2.744
	SAFT-VR SW	1.452	261.900	3.873	1.600		SAFT-VR SW	-	-	-	-
C_4H_{10}	HGO	1.468	463.340	4.722	3.319	C_7H_8	HGO	1.526	652.679	4.919	0.240
	HC	1.485	455.309	4.696	3.069		HC	1.539	650.839	4.888	0.176
	HSC	1.477	459.362	4.708	2.894		HSC	1.440	698.573	5.075	3.000
	SAFT-VR SW	1.501	257.200	3.887	2.000		SAFT-VR SW	-	-	-	-
C_5H_{12}	HGO	1.474	537.300	4.992	3.735	CF_4	HGO	1.495	234.709	3.880	3.306
	HC	1.511	522.183	4.920	4.037		HC	1.497	233.972	3.878	2.924
	HSC	1.474	539.644	4.988	3.476		HSC	1.491	235.352	3.888	2.743
	SAFT-VR SW	1.505	265.000	3.931	2.300		SAFT-VR SW	1.287	278.600	4.346	1.000
C_6H_{14}	HGO	1.548	574.850	5.100	4.775	C_2F_6	HGO	1.483	331.969	4.451	3.848
	HC	1.540	579.321	5.113	4.990		HC	1.483	331.969	4.451	3.848
	HSC	1.496	604.519	5.181	4.384		HSC	1.488	330.515	4.444	3.594
	SAFT-VR SW	1.552	250.400	3.920	2.600		SAFT-VR SW	1.339	289.000	4.436	1.370
C_7H_{16}	HGO	1.514	664.700	5.361	4.990	C_3F_8	HGO	1.514	410.163	4.786	4.725
	HC	1.486	679.234	5.417	5.000		HC	1.514	410.163	4.786	4.725
	HSC	1.494	674.863	5.400	4.923		HSC	1.477	424.338	4.858	4.430
	SAFT-VR SW	1.563	251.300	3.933	3.000		SAFT-VR SW	1.359	298.800	4.474	1.740

[a] for HGO, L/D stands for σ_e/σ_s and for SAFT-VR SW it stands for the number of spherical segments in a chain. σ_{sphere} stands for the diameter of the sphere with the same volume of the particle.

such equivalence. The HSC model is unable to describe disk-like particles (in the limit of $L/D \to 0$, the particle is a sphere).

Larger adjusted aspect ratios are observed for carbon dioxide and ethane when compared to the ones optimized in Chapter 2. A possible explanation for this might be that, since the Zhang's correction captures better the second-order perturbed term a_2, the attractive contribution is larger than the Barker-Henderson macroscopic approximation used in Chapter 2 for the same packing fraction. It can thus be suggested that the anisotropy is increased to compensate the larger attractive contribution.

The HGO and HC models have a superior performance for predicting the vapor-liquid equilibrium and enthalpy of vaporization of carbon dioxide (Figures 4.2 and 4.4), while the SAFT-VR SW shows the larger deviations from empirical data (Linstrom and Mallard). No significant differences between the models were observed for the vapor pressure (Figure 4.3). The average absolute relative deviations are presented in Table 4.3.

For smaller alkane molecules, CH_4 and C_2H_6, SAFT-VR SW predictions have slightly

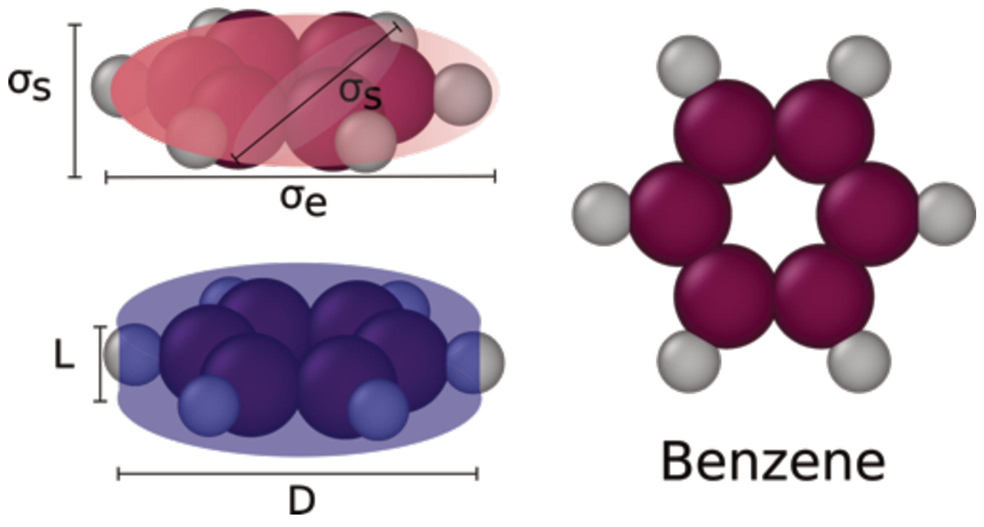


Figure 4.1: Illustration of a molecule of benzene (right) and the HGO (red) and HC (blue) models for the component.

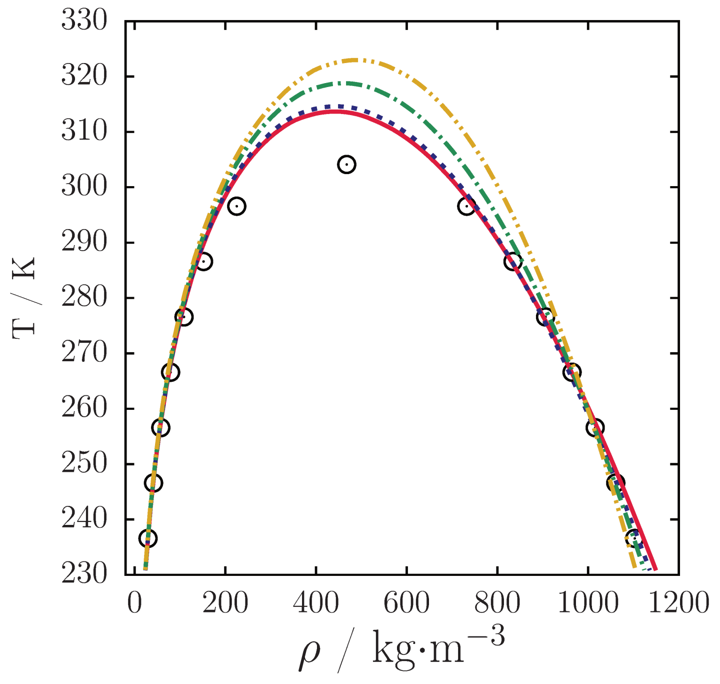


Figure 4.2: Vapor-liquid equilibria for carbon dioxide. Circles: NIST, continuous red line: HGO-SAFT, dashed blue line: HC-SAFT, dash-dotted line: HSC-SAFT, dash-double-dotted yellow line: SAFT-VR SW.

smaller deviations from empirical data (Table 4.2) when it comes to the saturated vapor density. For the other components, however, even if the model predicts more accurately a specific property, it is unable to do so without considerably increasing the deviations of other properties. Overall, the anisotropic models, HGO, HC, and HSC, predict the vapor-liquid equilibrium properties more accurately than SAFT-VR SW (Figure 4.5). The larger deviations in the SAFT-VR SW results are observed in the prediction of the vapor pressure

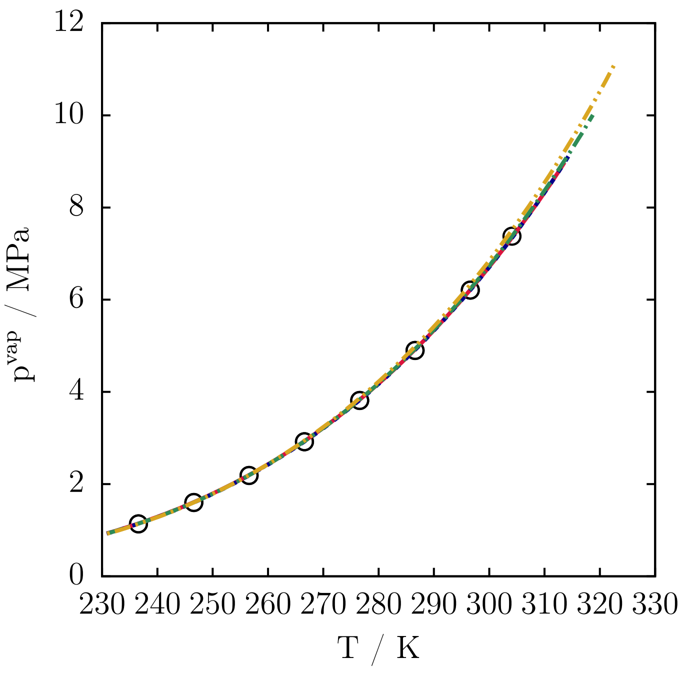


Figure 4.3: Vapor pressure as a function of temperature for carbon dioxide. Circles: NIST, continuous red line: HGO-SAFT, dashed blue line: HC-SAFT, dash-dotted line: HSC-SAFT, dash-double-dotted yellow line: SAFT-VR SW.

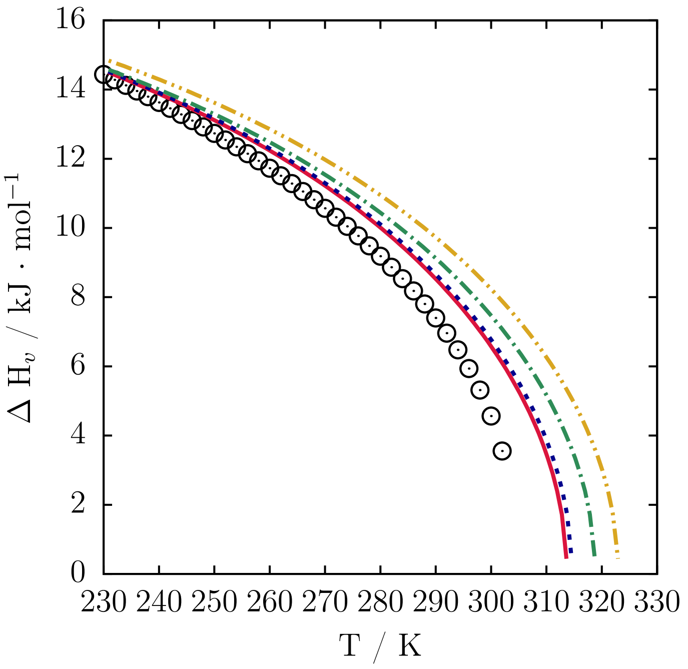


Figure 4.4: Enthalpy of vaporization as a function of temperature for carbon dioxide. Circles: NIST, continuous red line: HGO-SAFT, dashed blue line: HC-SAFT, dash-dotted line: HSC-SAFT, dash-double-dotted yellow line: SAFT-VR SW.

(Figure 4.6) and enthalpy of vaporization (Figure 4.7) of larger chains. These results raise the possibility of the chain contribution in SAFT-VR SW being altogether dispensable

Table 4.2: Average Absolute Relative Deviation (%)

compound	Range T(K)	EoS	p^{vap}	ρ_l	ρ_v	$\Delta_{\text{vap}}h$
CH_4	101-181	HGO	0.328	1.177	5.525	7.131
		HC	2.073	0.972	5.234	7.000
		HSC	0.461	1.880	5.902	8.324
		SAFT-VR SW	0.767	1.297	4.020	6.463
C_2H_6	130-300	HGO	1.739	1.348	6.022	7.814
		HC	1.333	2.636	6.389	9.218
		HSC	10.210	2.475	13.732	9.021
		SAFT-VR SW	5.420	2.843	5.879	9.247
C_3H_8	195-365	HGO	0.528	1.391	5.627	8.872
		HC	0.528	1.363	5.612	9.174
		HSC	0.557	1.390	5.915	9.389
		SAFT-VR SW	24.269	3.772	19.619	6.207
C_4H_{10}	225-420	HGO	1.279	1.560	6.277	11.154
		HC	0.941	1.470	6.058	10.555
		HSC	1.090	1.498	5.880	10.734
		SAFT-VR SW	4.573	2.629	8.918	14.731
C_5H_{12}	223-463	HGO	1.376	1.126	6.003	8.790
		HC	1.298	1.264	4.627	7.291
		HSC	1.395	1.091	5.334	8.498
		SAFT-VR SW	8.541	1.541	4.748	9.437
C_6H_{14}	223-503	HGO	1.848	1.982	5.111	5.869
		HC	1.841	1.822	5.160	6.091
		HSC	1.824	1.201	5.083	7.160
		SAFT-VR SW	16.855	2.807	13.207	8.372
C_7H_{16}	283-523	HGO	1.248	1.483	4.799	6.212
		HC	1.343	1.116	5.213	6.967
		HSC	1.311	1.199	5.083	6.749
		SAFT-VR SW	4.469	1.791	8.733	11.834
C_8H_{18}	306-566	HGO	1.514	1.721	5.502	11.716
		HC	0.854	3.232	4.849	7.619
		HSC	1.556	1.698	6.402	12.645
		SAFT-VR SW	3.959	2.997	4.534	17.491

provided that the anisotropy of the molecule is taken into account in the geometry of the segment, which turns out to be simpler in terms of calculation.

The HGO and HC models predictions of vapor-liquid equilibrium properties of benzene and toluene showed lower deviations from the empirical data when compared to the HSC equation of state, with exception of the saturated vapor density of benzene (Table 4.3). This was expected since the HSC model has no physical meaning for the compounds. Apart from the toluene saturated liquid density, the HC EoS presented a better perfor-

Table 4.3: Average Absolute Relative Deviation (%)

compound	Range T(K)	EoS	p^{vap}	ρ_l	ρ_v	$\Delta_{\mathrm{vap}}h$
CO_2	230-302	HGO	0.488	1.210	6.470	10.227
		HC	0.189	1.089	7.218	11.319
		HSC	0.453	1.993	8.714	15.057
		SAFT-VR SW	0.937	3.956	10.431	20.478
C_6H_6	349-559	HGO	0.855	2.200	7.001	14.339
		HC	0.494	1.691	6.858	12.819
		HSC	1.891	2.458	6.484	15.361
C_7H_8	298-588	HGO	1.091	1.466	6.966	11.249
		HC	1.034	1.609	6.172	10.453
		HSC	1.253	2.246	8.379	14.677
CF_4	120-220	HGO	0.612	0.782	3.875	5.261
		HC	0.601	0.788	3.880	5.227
		HSC	0.610	0.776	3.961	5.370
		SAFT-VR SW	3.812	4.847	3.342	10.796
C_2F_6	180-290	HGO	0.453	1.325	5.588	9.781
		HC	6.819	1.947	6.113	6.266
		HSC	0.443	1.321	5.484	9.566
		SAFT-VR SW	1.282	4.812	7.228	15.911
C_3F_8	180-330	HGO	1.470	1.721	3.871	4.128
		HC	12.327	3.707	16.110	7.415
		HSC	1.465	1.037	3.584	4.602
		SAFT-VR SW	1.925	2.209	4.504	8.076

mance than the HGO for predicting the VLE properties of cyclic aromatics (Figures 4.8, 4.9 and 4.10).

Except for the saturated vapor density of CF_4, when compared to SAFT-VR SW, the HGO and HSC predict more accurate equilibrium properties for all three perfluoroalkanes (Figures 4.11, 4.12 and 4.13), as shown in Table 4.3. For the C_2F_6 and C_3F_8, the HC model yield results for the vapor pressure (Figure 4.14) and saturated vapor density with deviations from empirical data considerably larger than the other models. The HGO and HSC EoSs yield better estimates for the enthalpy of vaporization (Figure 4.15) of C_3F_8, while for C_2F_6 the HC EoS has smaller deviations. The three anisotropic models predict better the enthalpy of vaporization of n-perfluoroalkanes when compared to SAFT-VR SW.

Tables 4.4 and 4.5 show the predictions of critical temperatures, T_c, pressures, P_c, compressibility factors, Z_c, critical densities, ρ_c, and acentric factors predicted with each model. For smaller n-alkanes and carbon dioxide, the HGO EoS yields better estimates of T_c and P_c, with exception of the critical temperature of methane, for which SAFT-VR SW prediction is slightly more accurate. SAFT-VR SW predicts a more accurate critical temperature for propane, but it loses a great deal of accuracy in the prediction of the vapor pressure and saturated vapor density for the fluid, with significant larger deviations when compared to the other models, as shown in Table 4.2. For larger n-alkanes chains (starting from butane) and the aromatics, the HC model yields better results for critical temperatures and pressures. All three nonspherical models describe the critical points better than SAFT-VR SW for the n-perfluoroalkanes.

Although the HGO, HC, and HSC models generally improve the calculation of critical properties when compared to SAFT-VR SW, all the models overpredict the critical point. As discussed on previous Chapters, the overprediction is likely to be attributed in part to the truncation of perturbation theory on the second-term. Lafitte et al. (2013) showed that incorporating higher-order terms improved properties prediction near the critical region and Ghobadi and Elliott (2015) introduced the Gaussian extrapolation method, an extrapolation of perturbation theory to infinite order that can also improve the critical properties prediction.

Interestingly, for the perfluoroalkanes and carbon dioxide, the anisotropic models not only yield better estimates of the critical points (Table 4.5) when compared to SAFT-VR SW, but also the overprediction observed for other components is less pronounced. These results might suggest that another possible explanation for the overshooting of the critical point is the use of a hard potential as the reference, since the these components have a more repulsive core (represented by higher adjusted repulsive coefficients on SAFT-VR Mie (Lafitte et al., 2013)), the impact of the application of a discrete potential is minimized.

4.3.1 Anisotropy *versus* acentric factor

The acentric factor (Pitzer, 1955) is somehow related to the nonsphericity of a molecule (Liu and Chen, 1996). In this regard, the HGO and HSC models have the characteristic of describing a spherical particle in the limit of $\chi \to 0$ (Equation 2.4) and $L/D \to 0$, respectively. Williamson and Guevara (1999) have presented the acentric factor as a function of the aspect ratio ($L/D \leq 3$) for a HSC model similar to ours, showing a linear behavior for values of the acentric factor $\omega < 0.15$.

Table 4.4: Critical properties and acentric factors

compound		NIST	HGO	HC	HSC	SAFT-VR SW
CO_2	T_c (K)	304.13	313.68	314.64	318.80	322.98
	P_c (MPa)	7.38	8.99	9.17	10.02	11.18
	Z_c	0.275	0.343	0.347	0.361	0.374
	ρ_c (kg·m^{-3})	467.60	441.78	444.45	461.02	490.17
	ω	0.224	0.178	0.176	0.163	0.164
C_6H_6	T_c (K)	562.05	594.44	591.26	594.76	-
	P_c (MPa)	4.89	7.11	6.78	7.32	-
	Z_c	0.265	0.370	0.363	0.376	-
	ρ_c (kg·m^{-3})	309.00	303.50	297.06	307.71	-
	ω	0.209	0.152	0.152	0.162	-
C_7H_8	T_c (K)	591.75	617.45	613.67	630.89	-
	P_c (MPa)	4.13	5.46	5.31	6.28	-
	Z_c	0.265	0.352	0.348	0.373	-
	ρ_c (kg·m^{-3})	292.00	278.60	275.35	296.06	-
	ω	0.262	0.215	0.228	0.190	-
CF_4	T_c (K)	227.51	238.28	238.29	238.58	249.40
	P_c (MPa)	3.75	5.01	5.01	5.05	7.19
	Z_c	0.279	0.368	0.368	0.370	0.426
	ρ_c (kg·m^{-3})	625.66	604.38	604.08	605.58	716.38
	ω	0.178	0.130	0.129	0.128	0.102
C_2F_6	T_c (K)	293.03	305.37	301.48	305.13	315.67
	P_c (MPa)	3.05	4.01	3.91	3.99	5.09
	Z_c	0.282	0.360	0.359	0.360	0.393
	ρ_c (kg·m^{-3})	613.30	605.12	600.06	603.60	679.66
	ω	0.257	0.207	0.219	0.207	0.175
C_3F_8	T_c (K)	345.02	354.40	362.69	355.66	365.60
	P_c (MPa)	2.64	3.20	3.36	3.33	4.19
	Z_c	0.276	0.344	0.347	0.351	0.382
	ρ_c (kg·m^{-3})	628.00	592.81	603.19	603.38	678.33
	ω	0.317	0.290	0.268	0.290	0.270

When plotting $|\chi|$ (calculated from the optimized aspect ratio) and L/D^{HSC} as a function of the acentric factor ω (Linstrom and Mallard) (Table 4.5), we have observed that for $\omega > 0.14$, both functions exhibited a sigmoidal behavior. Correlating $|\chi|$ and L/D^{HSC} with the acentric factor ω of the n-alkanes (from propane to octane), we have adjusted the following functions:

$$\chi(\omega) = \frac{0.0920159}{1 + \exp(-55.79(\omega - 0.260047))} + 0.831352 \tag{4.6}$$

$$L/D^{\mathrm{HSC}} = \frac{2.5222}{1 + \exp(-29.1866(\omega - 0.270769))} + 2.58321 \tag{4.7}$$

Table 4.5: Critical properties and acentric factors

compound		NIST	HGO	HC	HSC	SAFT-VR SW
CH_4	T_c (K)	190.56	205.89	201.43	208.66	204.19
	P_c (MPa)	4.60	6.79	6.25	7.28	6.71
	Z_c	0.286	0.392	0.379	0.405	0.382
	ρ_c (kg·m^{-3})	162.66	162.40	158.07	166.15	165.90
	ω	0.011	-0.067	-0.034	-0.079	-0.048
C_2H_6	T_c (K)	305.33	322.94	325.18	327.62	325.33
	P_c (MPa)	4.87	6.67	7.01	7.02	7.43
	Z_c	0.279	0.371	0.384	0.383	0.378
	ρ_c (kg·m^{-3})	207.00	201.02	203.05	202.68	218.23
	ω	0.098	0.036	0.041	-0.005	0.043
C_3H_8	T_c (K)	369.82	388.33	388.75	389.64	382.83
	P_c (MPa)	4.25	5.69	5.75	5.80	6.59
	Z_c	0.277	0.364	0.366	0.367	0.376
	ρ_c (kg·m^{-3})	220.00	213.53	214.46	215.12	242.49
	ω	0.149	0.098	0.099	0.095	0.123
C_4H_{10}	T_c (K)	425.12	448.43	447.42	447.58	449.71
	P_c (MPa)	3.80	5.42	5.31	5.35	5.80
	Z_c	0.274	0.371	0.368	0.369	0.374
	ρ_c (kg·m^{-3})	228.00	227.42	225.42	226.30	241.30
	ω	0.197	0.150	0.150	0.151	0.177
C_5H_{12}	T_c (K)	469.70	493.81	488.45	492.25	490.10
	P_c (MPa)	3.37	4.73	4.45	4.69	5.27
	Z_c	0.268	0.363	0.354	0.362	0.374
	ρ_c (kg·m^{-3})	232.00	228.81	223.11	228.16	249.35
	ω	0.251	0.200	0.214	0.207	0.237
C_6H_{14}	T_c (K)	507.82	522.77	523.39	526.31	532.22
	P_c (MPa)	3.03	3.73	3.77	3.98	4.89
	Z_c	0.266	0.342	0.343	0.350	0.375
	ρ_c (kg·m^{-3})	233.18	216.41	217.61	224.01	253.85
	ω	0.304	0.268	0.268	0.269	0.250
C_7H_{16}	T_c (K)	540.13	556.65	559.81	558.90	572.48
	P_c (MPa)	2.74	3.45	3.60	3.56	4.43
	Z_c	0.263	0.341	0.346	0.344	0.376
	ρ_c (kg·m^{-3})	232.00	219.30	224.12	222.79	247.73
	ω	0.346	0.321	0.314	0.316	0.310
C_8H_{18}	T_c (K)	569.32	588.04	580.25	590.55	600.98
	P_c (MPa)	2.50	3.29	2.88	3.37	4.11
	Z_c	0.257	0.343	0.328	0.346	0.377
	ρ_c (kg·m^{-3})	234.90	224.44	207.59	226.69	249.43
	ω	0.396	0.369	0.370	0.363	0.350

We have then used the adjusted function to predict the aspect ratios and anisotropies of CF_4, C_2F_6 and C_3F_8 (red circles in Figure 4.16). In this way, only λ, ϵ and σ_{sphere} were optimized for the components and the HGO aspect ratio was calculated using Equation 2.4.

The optimized HGO anisotropy of benzene and toluene are close to the calculated with Equation 4.6. The HSC aspect ratio calculated with Equation 4.7 for these components, however, are not as close. Conversely, the optimized L/D for carbon dioxide is close to the one calculated with the function (Equation 4.7), whereas the optimized anisotropy is far from the calculated one (Equation 4.6).

The aspect ratios of the n-perfluoroalkanes calculated using the correlated functions are the same as the optimized for CF_4 and C_2F_6 (red circles and open yellow squares, respectively). As for the C_3F_8, though not exactly the same, the calculated and optimized values of the aspect ratio are close. We have calculated the average absolute deviation (Table 4.6) and the critical properties (Table 4.7) using the correlated functions to calculate the aspect ratio of C_3F_8 and then optimizing λ, ϵ, and σ_{sphere} to compare with the results obtained by optimizing all four parameters. Comparing these results with the ones obtained on the previous section, the differences are insignificant. Thus, the correlated functions appear to be good adjustments for predicting the aspect ratio.

Table 4.6: Average Absolute Relative Deviation (%) for vapor pressure, saturated liquid density, saturated vapor density and enthalpy of vaporization of C_3F_8.

compound	T range	EoS	P^{vap}	ρ_l	ρ_v	ΔH_{v}
C_3F_8	180-330	HGO	1.297	2.136	2.671	3.478
		HSC	1.434	1.295	3.095	4.432

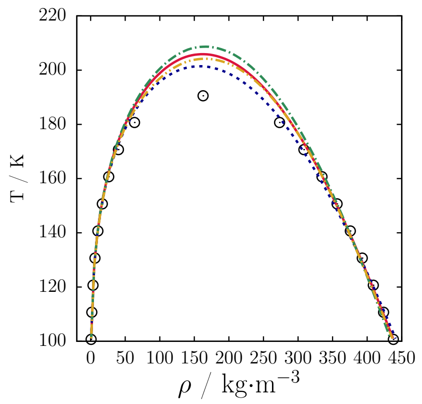
Table 4.7: Critical properties and acentric factor of C_3F_8.

compound		HGO	HSC
C_3F_8	T_c (K)	352.20	354.48
	P_c (MPa)	3.11	3.28
	Z_c	0.342	0.349
	ρ_c (kg $\cdot$m$^-$3)	584.46	600.19
	ω	0.302	0.299

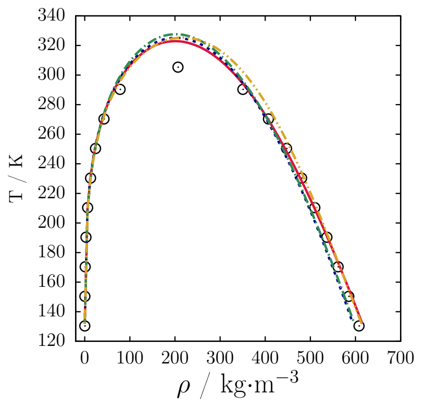
Deviations are slightly larger when the optimized aspect ratios are used, with exception of the saturated liquid density. Since the vapor pressure and saturated liquid density were used in the objective function, one possible explanation is that the saturated liquid density error was being minimized at the expense of the vapor pressure error.

4.4 Conclusion

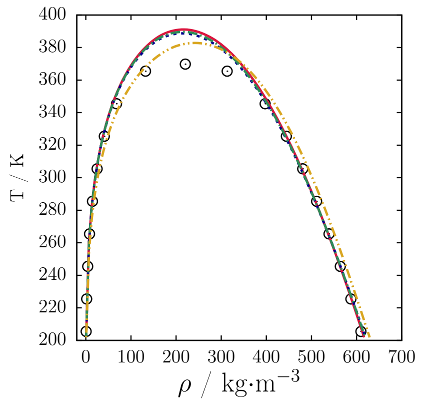
Vapor-liquid equilibrium properties of several molecules (carbon dioxide, n-alkanes, cyclic aromatics and perfluoroalkanes) were calculated using equations of state for ellipsoidal (HGO), cylindrical (HC), and spherocylindrical (HSC) particles. The results were tested against empirical data (Linstrom and Mallard) and SAFT-VR SW. We have shown that, overall, the results are improved solely by modeling the molecules as a single anisotropic segment (ellipsoid, cylinder, or spherocylinder) instead of a set of spherical segments, even while applying the anisotropic potential as the reference but keeping the isotropic square potential to model the segment dispersion. The results suggest that, for the molecules investigated in this work, the chain contribution in SAFT approach could be completely eliminated provided that the segment contribution is modeled with a nonspherical geometry. The prediction of the vapor-liquid equilibrium properties of the n-perfluoralkanes were considerably better than SAFT-VR SW results. In addition to that, the overprediction of the critical point was less pronounced for the n-perfluoroalkanes when compared to the other molecules, raising the possibility that the overshooting could be related to, besides the truncation of perturbation theory on the second-order term, the use of a discrete potential as the reference. Since the n-perfluoroalkanes have a more repulsive core, the impact of the application of a hard potential on the critical properties prediction is minimized. We have correlated the optimized aspect ratio of n-alkanes with the acentric factor of the molecules to predict the aspect ratio of n-perfluoralkanes. In this way, we only had to optimize three parameters for the components: the well-depth ϵ, the potential range λ, and the volume, represented by the diameter σ_{sphere} of a sphere with the same volume of the particle. The results were satisfactory, implying that the correlated functions could be used to predict the aspect ratio of similar molecules instead of having to optimized it. A natural progression of this work is to include higher-order terms in the perturbation theory expansion and to develop a soft anisotropic potential, for instance, by including an effective diameter dependent on the temperature. We believe that those would be fruitful approaches for further development.



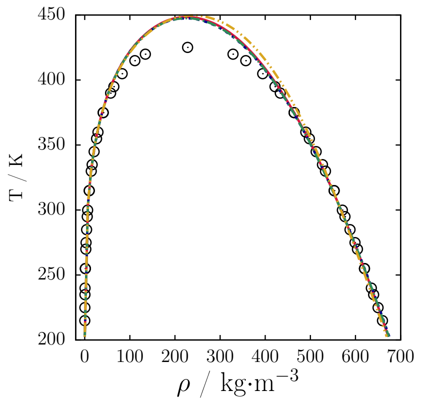
(a) CH_4



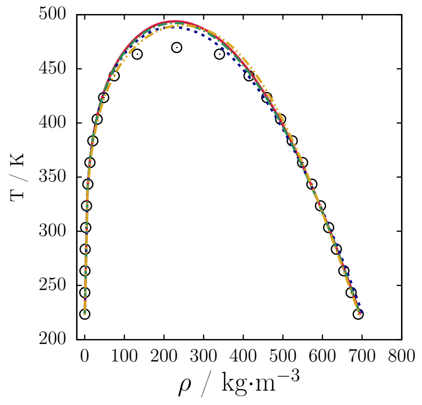
(b) C_2H_6



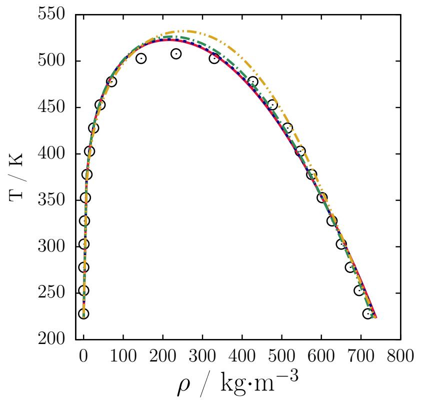
(c) C_3H_8



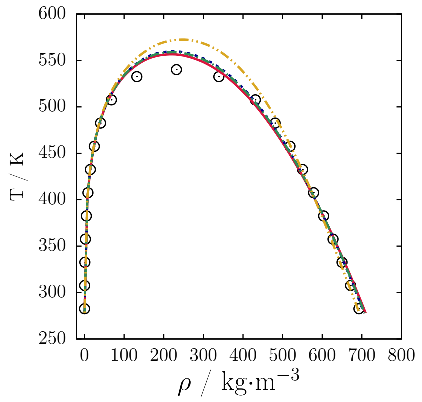
(d) C_4H_{10}



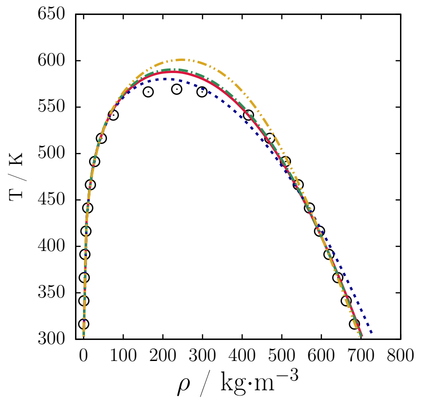
(e) C_5H_{12}



(f) C_6H_{14}



(g) C_7H_{16}



(h) C_8H_{18}

Figure 4.5: Vapor-liquid equilibria for *n*-alkanes. Circles: NIST, continuous red line: HGO-SAFT, dashed blue line: HC-SAFT, dash-dotted line: HSC-SAFT, dash-double-dotted yellow line: SAFT-VR SW.

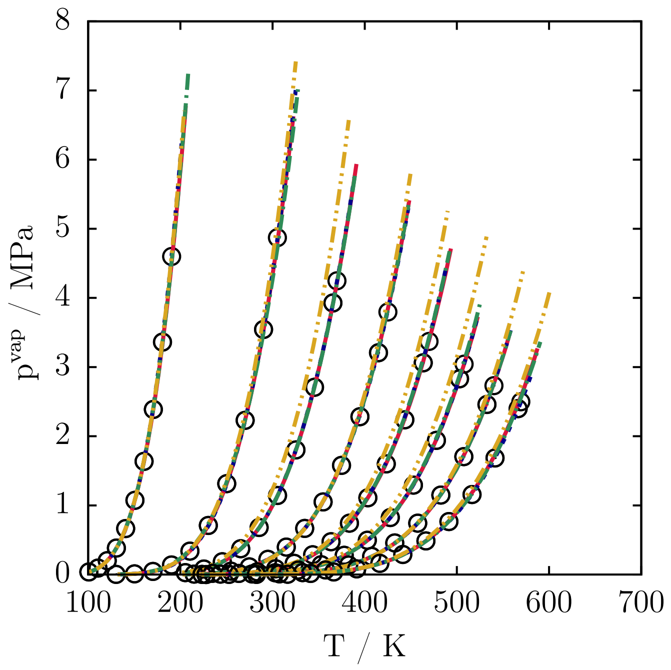


Figure 4.6: Vapor pressure as a function of temperature for *n*-alkanes. CH_4 to C_8H_{18} from left to right. Circles: NIST, continuous red line: HGO-SAFT, dashed blue line: HC-SAFT, dash-dotted line: HSC-SAFT, dash-double-dotted yellow line: SAFT-VR SW.

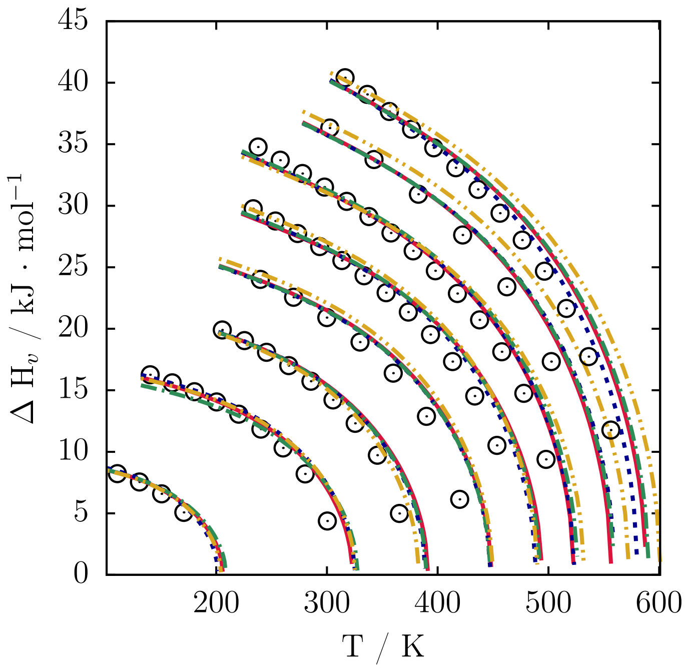
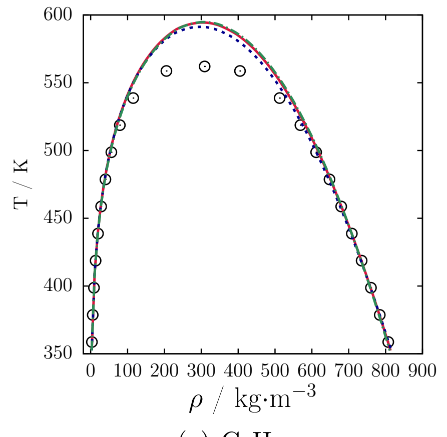
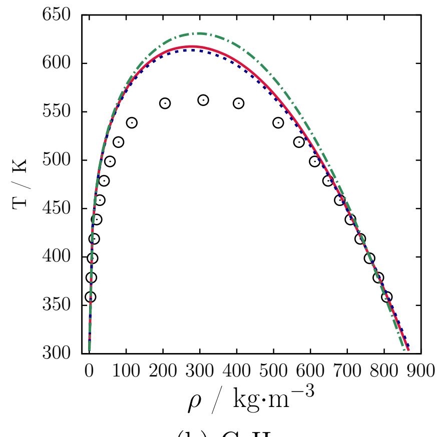


Figure 4.7: Enthalpy of vaporization as a function of temperature for *n*-alkanes. CH_4 to C_8H_{18} from left to right. Circles: NIST, continuous red line: HGO-SAFT, dashed blue line: HC-SAFT, dash-dotted line: HSC-SAFT, dash-double-dotted yellow line: SAFT-VR SW.



(a) C_6H_6



(b) C_7H_8

Figure 4.8: Vapor-liquid equilibria for benzene and toluene. Circles: NIST, continuous red line: HGO-SAFT, dashed blue line: HC-SAFT, dash-dotted line: HSC-SAFT, dash-double-dotted.

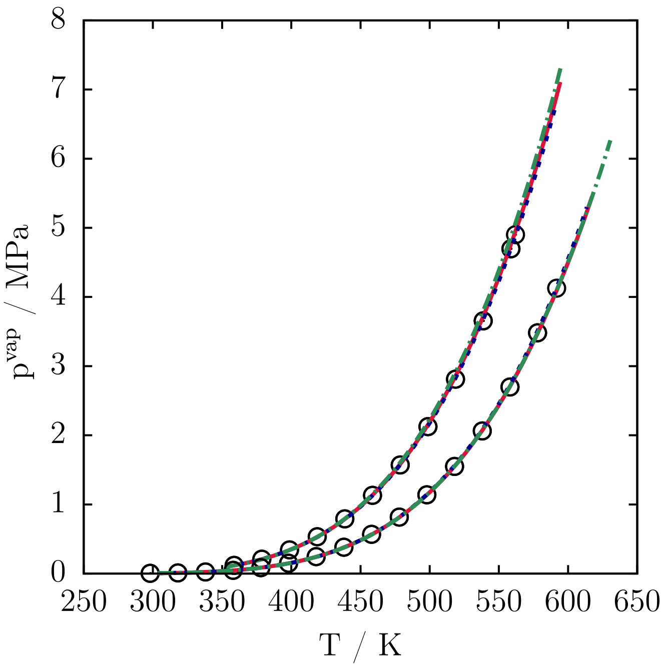


Figure 4.9: Vapor pressure as a function of temperature for benzene and tolune. Circles: NIST, continuous red line: HGO-SAFT, dashed blue line: HC-SAFT, dash-dotted line: HSC-SAFT, dash-double-dotted yellow line: SAFT-VR SW.

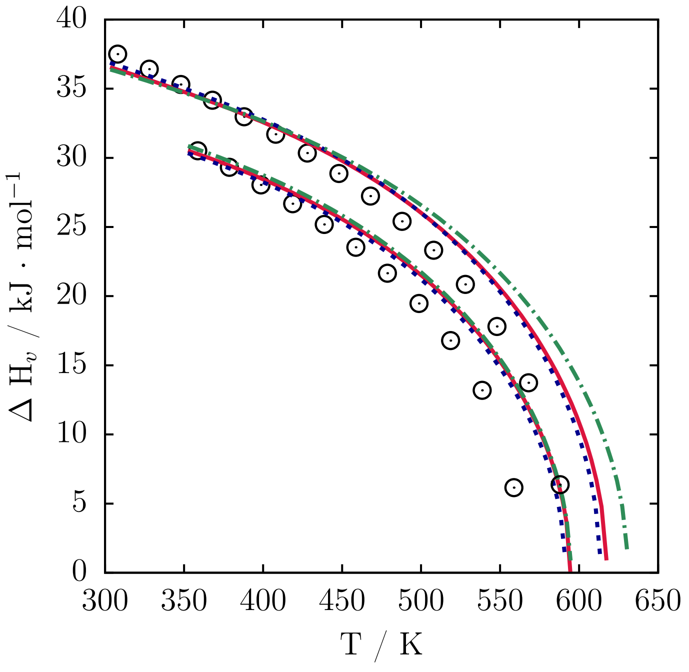


Figure 4.10: Enthalpy of vaporization as a function of temperature for benzene and toluene. Circles: NIST, continuous red line: HGO-SAFT, dashed blue line: HC-SAFT, dash-dotted line: HSC-SAFT, dash-double-dotted yellow line: SAFT-VR SW.

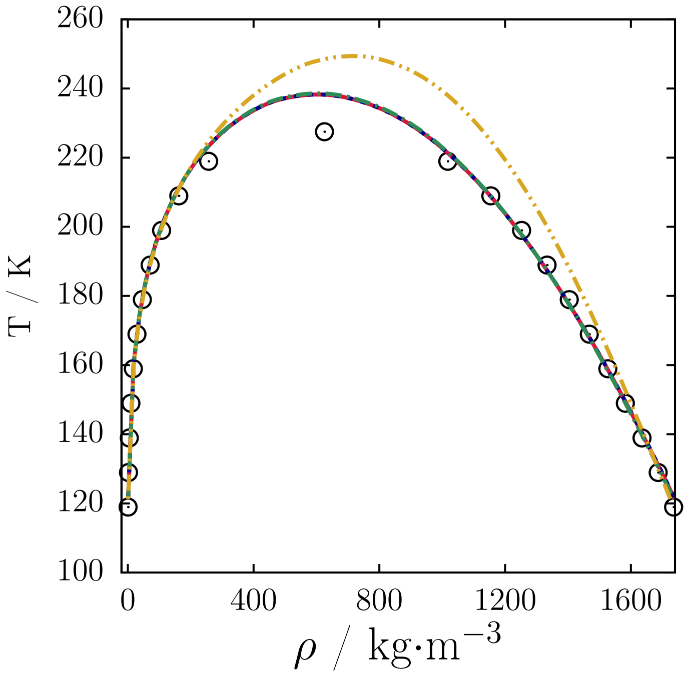


Figure 4.11: Vapor-liquid equilibria for CF_4. Circles: NIST, continuous red line: HGO-SAFT, yellow line: SAFT-VR SW.

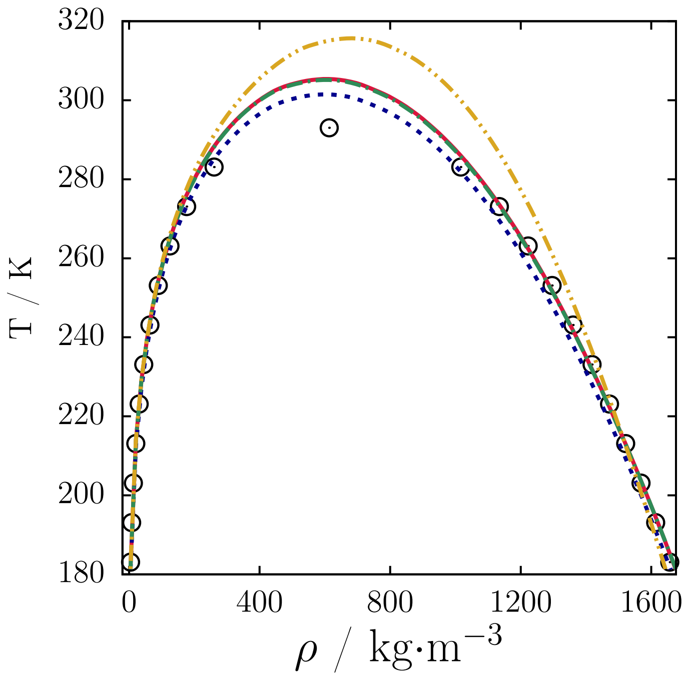


Figure 4.12: Vapor-liquid equilibria for C_2F_6. Circles: NIST, continuous red line: HGO-SAFT, yellow line: SAFT-VR SW.

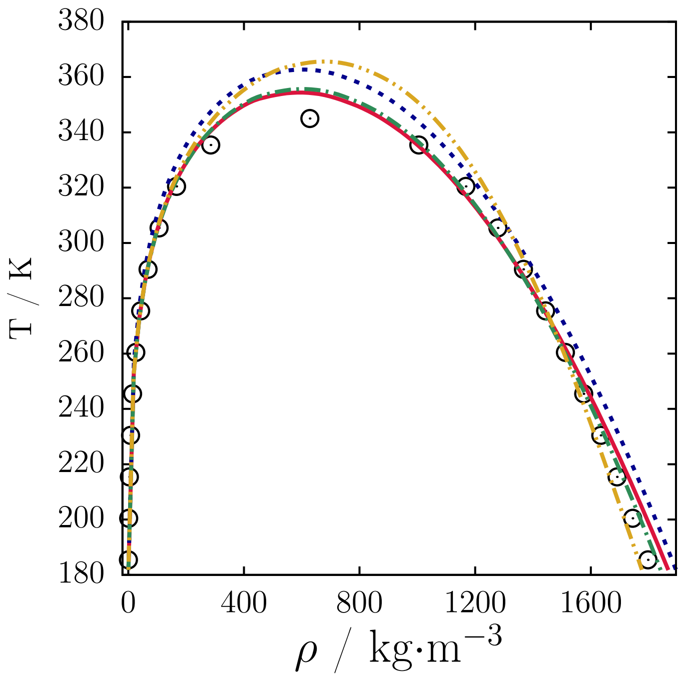


Figure 4.13: Vapor-liquid equilibria for C_3F_8. Circles: NIST, continuous red line: HGO-SAFT, yellow line: SAFT-VR SW.

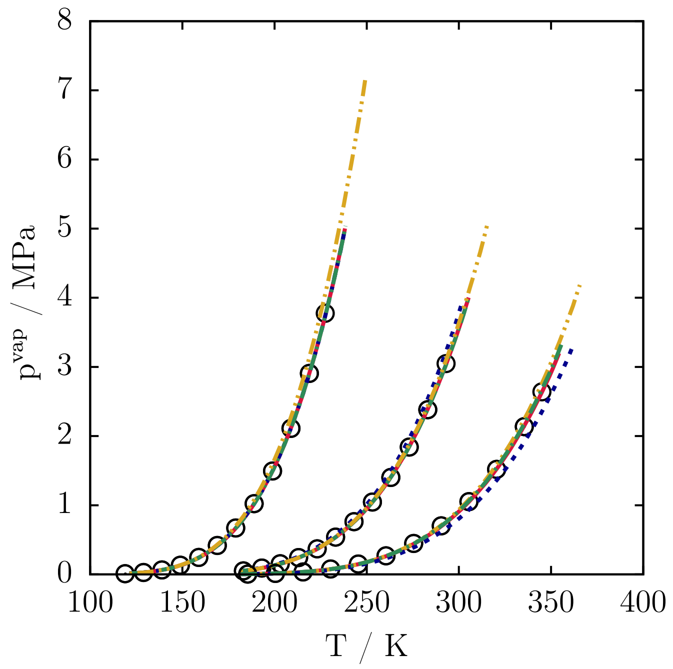


Figure 4.14: Vapor pressure as a function of temperature for n-perfluoroalkanes. Circles: NIST, continuous red line: HGO-SAFT, yellow line: SAFT-VR SW.

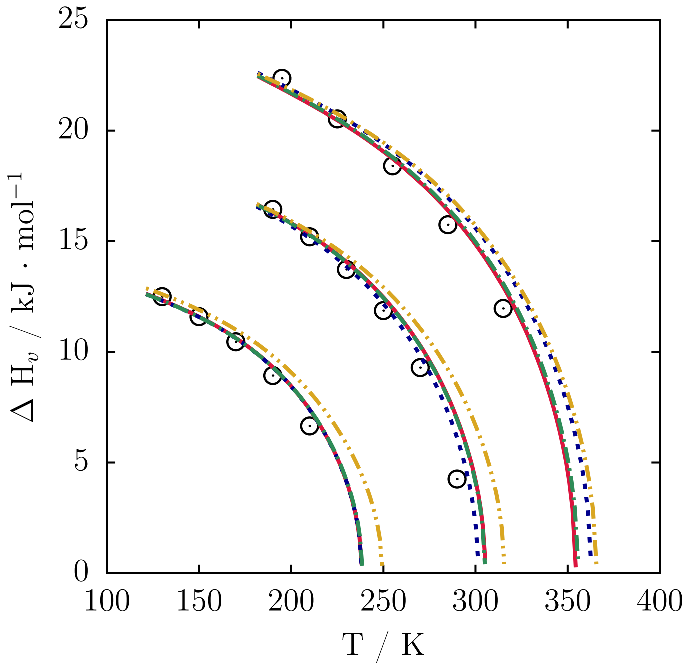
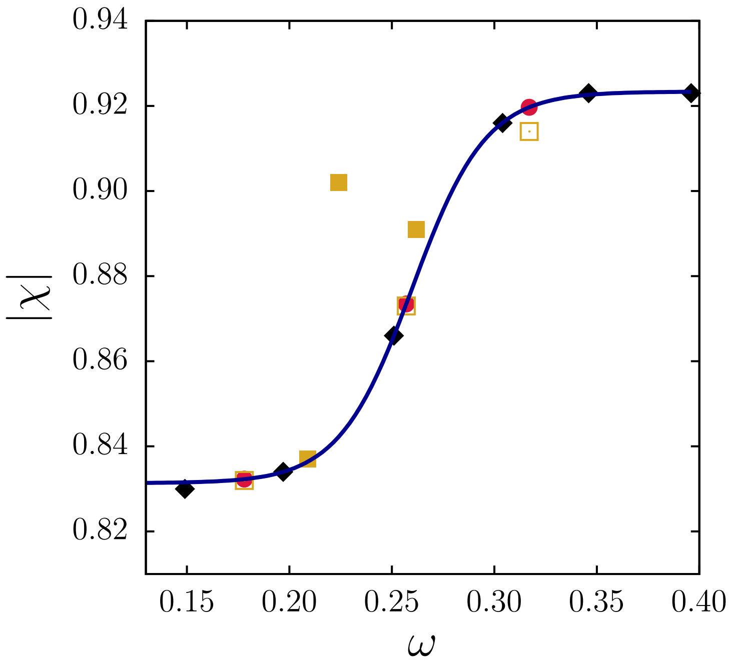
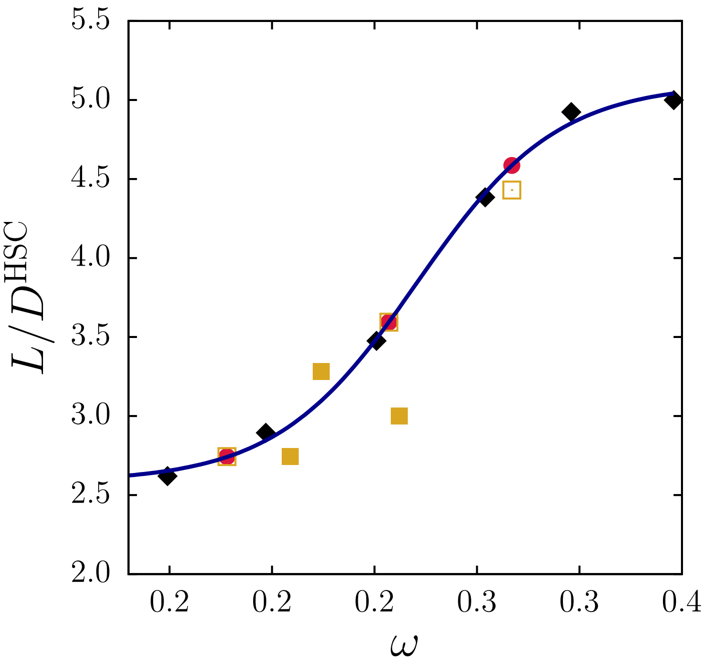


Figure 4.15: Enthalpy of vaporization as a function of temperature n-perfluoroalkanes. Circles: NIST, continuous red line: HGO-SAFT, yellow line: SAFT-VR SW.



(a) HGO



(b) HSC

Figure 4.16: Black diamonds: optimized anisotropy used to adjust the function, continuous blue line: adjusted function (Equation 4.6/Equation 4.7), red circles: ω and L/D calculated using the adjusted function (CF_4, C_2F_6, and C_3F_8), yellow open squares: optimized ω and L/D of CF_4, C_2F_6, and C_3F_8, yellow closed squares: optimized ω and L/D of C_6H_6, CO_2, and C_7H_8.

“ . . . quanto alla verità di che ci danno cognizione le dimostrazioni matematiche, ella è l’istessa che conosce la sapienza divina;”

— Galileo Galilei, Dialogo sopra i due massimi sistemi

5 Investigation of the phase boundaries of hard cylinders

The content of the next two chapters is the result of a research period at Università Ca’ Foscari Venezia under the supervision of Professor Achille Giacometti.

5.1 Introduction

On chapter 4, we have derived equations of state for fluids of industrial relevance based on the equations of state for the isotropic phase of the hard Gaussian overlap fluid, spherocylinders, and cylinders. Nonspherical particles like these are often used to study liquid crystals, since their shape anisotropy promotes the formation of organized phases at sufficient large packing fractions and aspect ratios.

The phase transitions of the hard Gaussian overlap (De Miguel and Del Río, 2001) and hard spherocylinder (Bolhuis and Frenkel, 1997) have been studied, but, although the hard cylinder has been used as the base model to study more complex systems such as hard cylinders with attractive patches (Orellana et al., 2018; Nguyen et al., 2014; De Michele, 2019; De Michele et al., 2012)), to our knowledge no systematic investigation of the phase boundaries of hard cylinders has been carried out.

Therefore, in this chapter, we aim at providing a benchmark for the development of more complex models based on the hard cylinder fluid. To this end, we investigate the liquid crystalline phase formation over a wide range of aspect ratios, including both rod and disk-like particles.

5.2 NPT Monte Carlo simulations

An introduction to the Monte Carlo method was given in section 3.3. In this chapter, we develop Monte Carlo simulations in the isobaric-isothermal ensemble (NPT), since most real experiments are performed under conditions of controlled temperature and pressure. In this case, the total volume of the system is allowed to change to stabilize the system at the given pressure. The probability of the system going from a state m to a state n in the NPT ensemble is:

$$\begin{aligned} \frac{P(\Gamma_n)}{P(\Gamma_m)} &= \frac{V_n^N \exp(-\beta(U_n + PV_n))}{V_m^N \exp(-\beta(U_m + PV_m))} \\ &= \exp(-\beta(\Delta U_{mn} + P\Delta V_{mn}) + N \ln \Delta V_{mn}) \end{aligned} \tag{5.1}$$

In comparison to the prescription given in section 3.3, there are two main differences with regard to the implementation of the MC code in the NPT ensemble: a trial move consists of either changing randomly the position and orientation, or the volume, and the acceptance criterion is a function of the following quantity $\Delta H = \Delta U_{mn} + P\Delta V_{mn} + k_B T N \ln \Delta V_{mn}$. A flowchart for the method is given in Figure 5.2.

Eppenga and Frenkel (1984) pointed out that a change in the natural logarithm of the volume would be more convenient rather the in the volume itself. Nevertheless, ΔH has to change to $\Delta H = \Delta U_{mn} + P\Delta V_{mn} + k_B T (N+1) \ln \Delta V_{mn}$. The Fortran 90 pseudo-code for the volume move can be written as follows:

```
Call random_number(rnum)


! Make a random change in the logarithm:
```

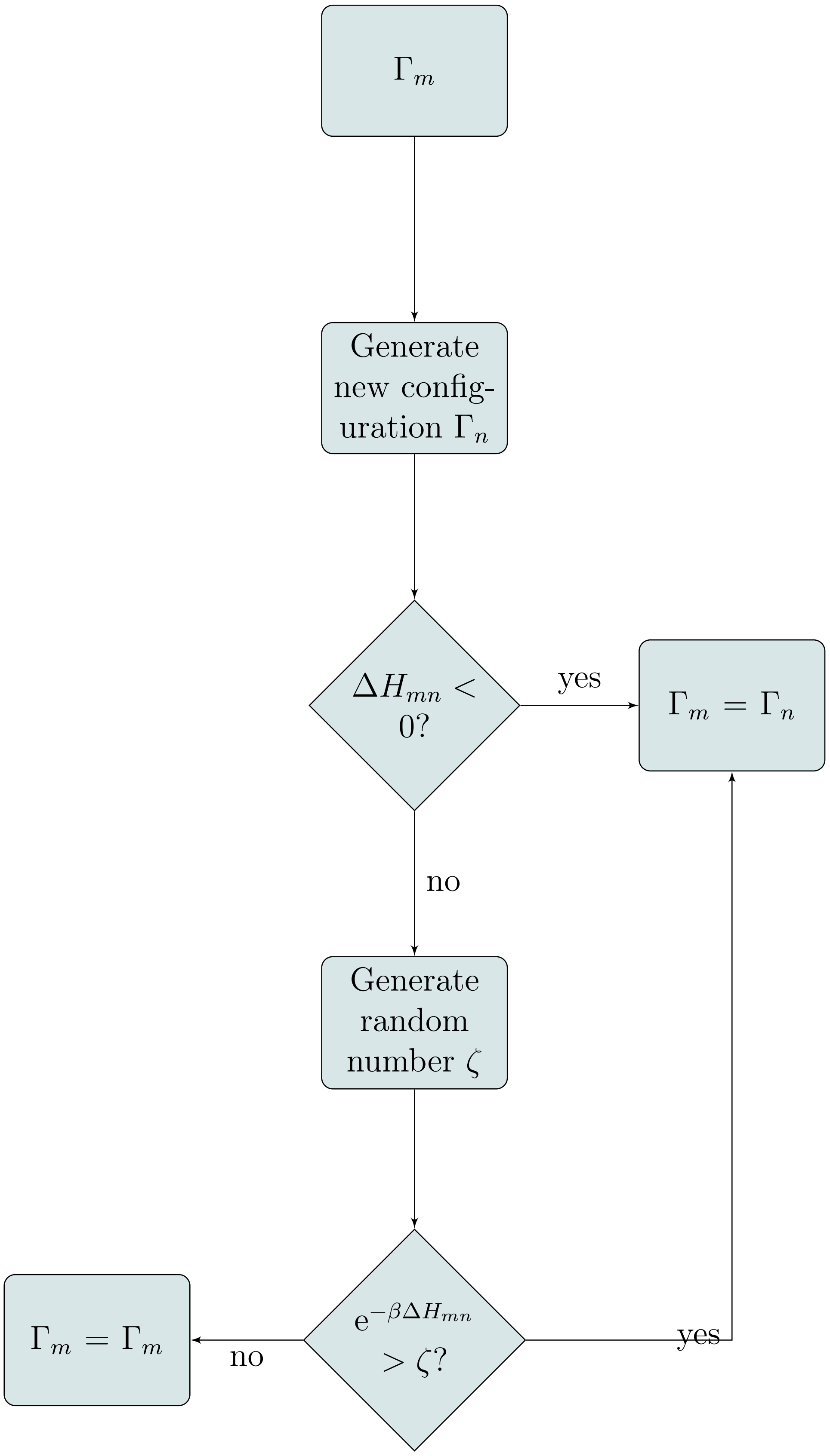

Γ_m
Generate new configuration Γ_n
$\Delta H_{mn} < 0$?
yes
$\Gamma_m = \Gamma_n$
no
Generate random number ζ
$e^{-\beta \Delta H_{mn}} > \zeta$?
yes
no
$\Gamma_m = \Gamma_m$

Figure 5.1: Metropolis method flowchart for a cycle in the NPT ensemble.

```
    lnvnew = dlog(v) + (2.d0*rnum - 1.d0)*max_v
    ! Calculate the equivalent change in the volume:
    vnew = dexp(lnvnew)
    ! Calculate the scale ratio of the change
    boxr = (vnew/v)**(1.d0/3.d0)
    ! Calculate the new sizes of the box (on x, y and z)
    boxlnew(:) = boxr*boxl(:)

    ! Scale the positions of the particles
    Do j=1,n
           rnew(1,j) = boxr*r(1,j)
           rnew(2,j) = boxr*r(2,j)
           rnew(3,j) = boxr*r(3,j)
    End Do
```

5.2.1 Floppy-box Monte Carlo

The floppy-box is used to allow the box shape to fluctuate and obtain an isotropic pressure in smectic and crystalline phases. In this case, we choose randomly one of the three axes to make a change, as presented below in the pseudo-code:

```
boxlnew = boxl
rnew = r
lmax = max_v**(1.d0/3.d0)
Call random_number(rnum)
if (rnum .lt. 0.3333333d0) then
  axis = 1
else if (rnum .lt. 0.6666666d0) then
  axis = 2
else
  axis = 3
end if
Call random_number(rnum)
```

```
boxlnew(axis) = boxl(axis) + (2.d0*rnum - 1.d0)*lmax
boxr = boxlnew(axis)/boxl(axis)
Do j=1,n
rnew(axis,j) = boxr*r(axis,j)
End Do
vnew = boxlnew(1)*boxlnew(2)*boxlnew(3)
```

For hard core potentials, an essential part for the development of Monte Carlo simulations is to check whether the trial move results or not in an overlap between the particles. Monte Carlo simulations of spherocylinders are abundant in the literature, since the test for the overlap is relatively simple. For cylinders, however, there are a few (Orellana et al., 2018) but insufficient data in the literature, since checking the overlap between two cylinders is considerably more complex, and computationally expensive. On the next sections, we provide a detailed procedure of how to check the overlap between two cylinders.

5.3 Overlap between two cylinders

We define L and D as being the length and diameter of two identical cylinders, respectively. The orientations are defined as $\widehat{\Omega}_1$ and $\widehat{\Omega}_2$. The overlap of two cylinders can occur in one of the three manners: disk-rim, rim-rim, or disk-disk (Figure 5.2). Therefore, to ensure that the cylinders do not overlap, we have to check if the overlap occurs in any of these possible configurations.

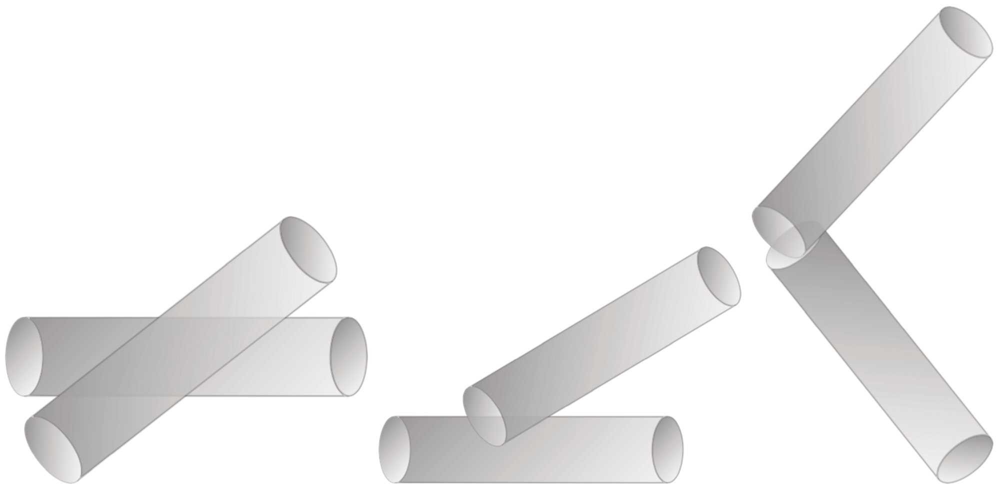

(a) Rim-rim overlap (b) Disk-rim overlap (c) Disk-disk overlap

Figure 5.2: Possible overlap configurations between two cylinders.

To optimize the simulation, we start from the simpler and less expensive tests, as shown in Figure 5.3. The first step is to check if the spheres that enclose the cylinders overlap; if they do not, the cylinders cannot overlap either. If the spheres do overlap, the test is then done for the spherocylinders enclosing the particles. Only if the spherocylinders overlap, one should check the overlap between two cylinders.

On the next sections, we provide detailed explanation on how to proceed with each test.

5.3.1 Spherocylinders

When two spherocylinders touch each other, the shortest distance between their segments is always equal to the diameter D, as shown in Figure 5.4. Hence, the test for overlap comes down to finding the shortest distance s_d between the two segments of length L. In this sense, Vega and Lago (1994) proposed a fast algorithm to calculate the shortest distance between two segments. The prescription for calculating the distance between parallel rods was later improved by Abreu et al. (2003). We shall use, however, the original Vega and Lago's algorithm to calculate the shortest distance between two segments. To avoid the *"go to"* used by the authors, some small modifications were applied to the original algorithm. The code is provided in the appendix.

5.3.2 Cylinders

Parallel Cylinders

If two cylinders are parallel, $\widehat{\Omega}_1 \cdot \widehat{\Omega}_2 = \pm 1$, the overlap can occur between disk-disk or rim-rim only, and it can be easily checked. We decompose the vector joining the two centers of mass $\mathbf{r}_{12}$ into a vector parallel to the orientations $\mathbf{r}_{ij\parallel}$, and one perpendicular to it $\widehat{r}_{ij\perp}$, as presented in Equation 5.2.

$$\begin{aligned} \mathbf{r}_{12\parallel} =& (\mathbf{r}_{12} \cdot \widehat{\Omega}_1)\widehat{\Omega}_1 \\ \mathbf{r}_{12\perp} =& \mathbf{r}_{12} - (\mathbf{r}_{12} \cdot \widehat{\Omega}_1)\widehat{\Omega}_1 \end{aligned} \tag{5.2}$$

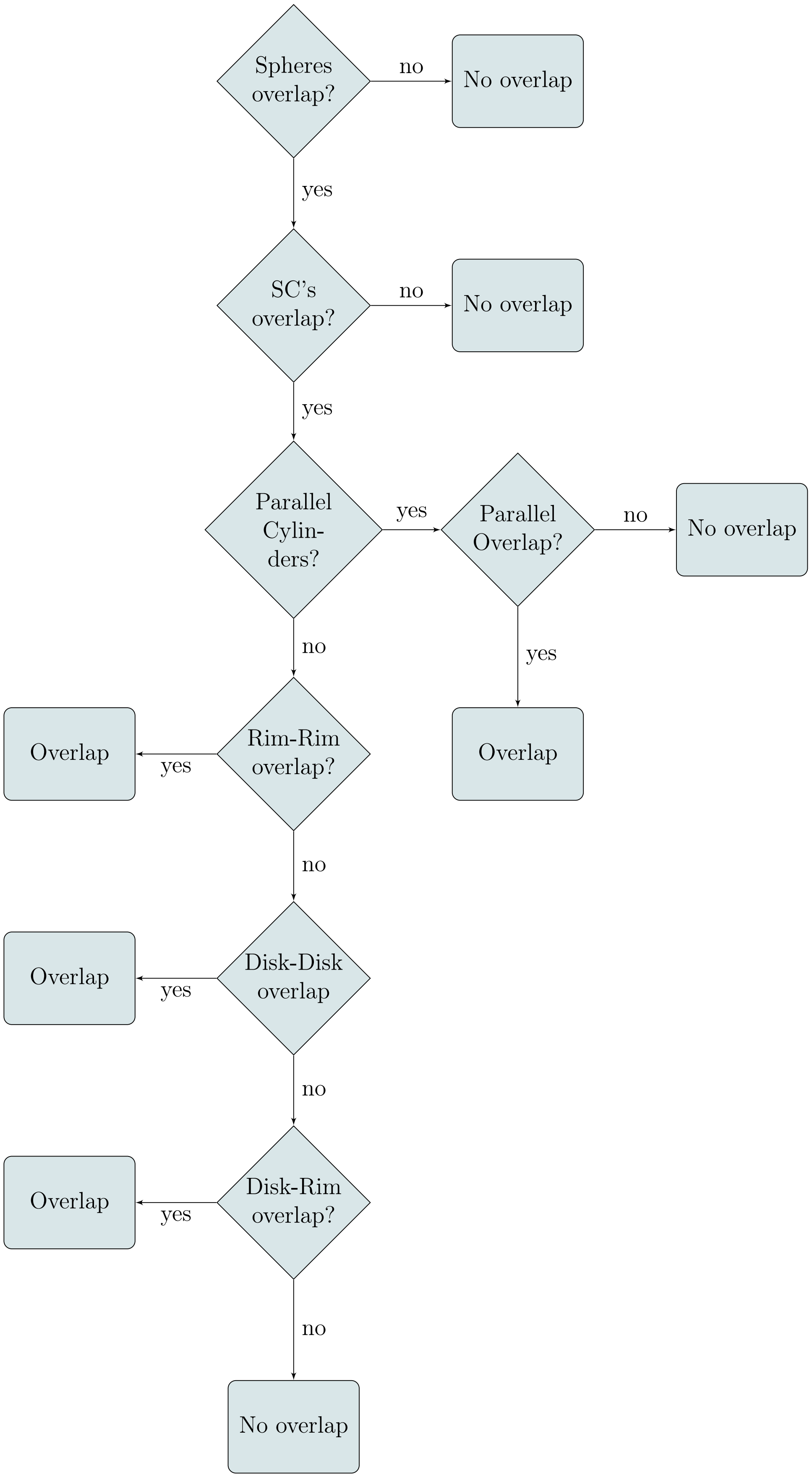


Figure 5.3: Cylinder Overlap Flowchart.

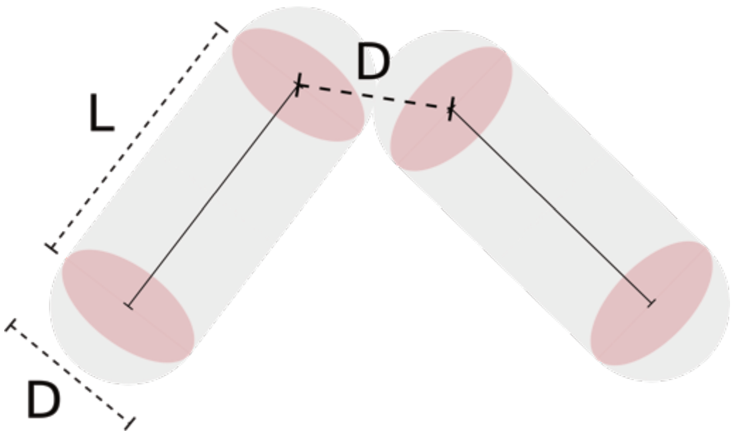


Figure 5.4: Contact of two spherocylinders.

The overlap occurs if both of the conditions in Equation 5.3 are satisfied.

$$\begin{aligned} |\mathbf{r}_{ij\parallel}| &\leq L \\ |\mathbf{r}_{ij\perp}| &\leq D \end{aligned} \tag{5.3}$$

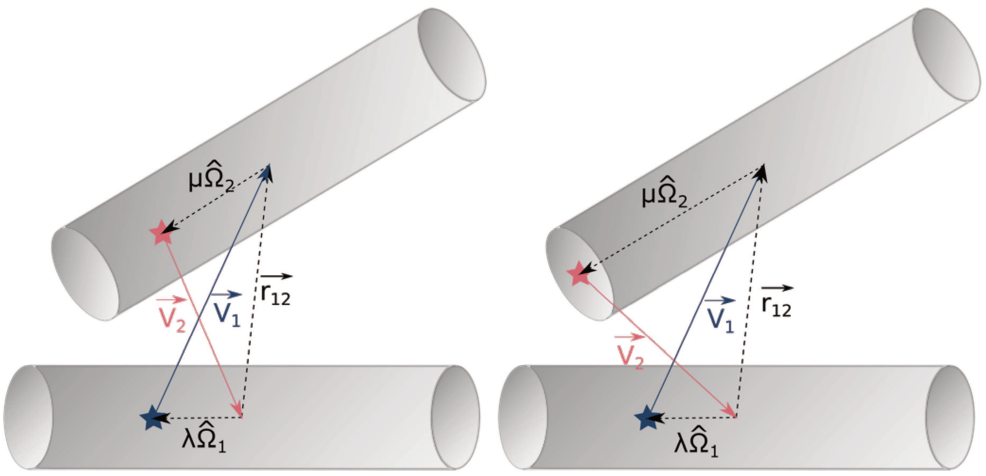
Rim-rim overlap - (Dr. Flavio Romano, Università Ca' Foscari Venezia, personal communication, 2019)

Since the overlap between spherocylinders is the first test that is done, and the rim of a spherocylinder is similar to the rim of a cylinder, if the spherocylinders overlap, the cylinders will certainly overlap as well. Hence, to check if there is an overlap between two rims, a sufficient test is to check whether the cylinders are in a rim-rim configuration.

To that end, we define the vectors $\mathbf{V}_1 = -\mathbf{r}_{12} + \lambda\widehat{\Omega}_1$ and $\mathbf{V}_2 = \mathbf{r}_{12} + \mu\widehat{\Omega}_2$, where λ and μ are the points of closest approach between the two cylinders, which is calculated using the Vega and Lago (1994)'s algorithm. If the cylinders are in a rim-rim configuration, the two conditions below are satisfied.

- $|\mathbf{V}_1 \cdot \widehat{\Omega}_2| < L/2$
- $|\mathbf{V}_2 \cdot \widehat{\Omega}_1| < L/2$

In Figure 5.3.2, we see that in the case of a disk-rim configuration, for instance, the projection of $\mathbf{V}_1$ on the direction of $\widehat{\Omega_2}$ is larger than $L/2$.



(a) Rim-rim configuration (b) Disk-rim configuration

Figure 5.5: The star symbols represent the points of closest approach on each cylinder.

Disk-disk overlap - (Allen et al. (2007))

The orientations of the cylinders are perpendicular to the planes of the disks. The planes of the two disks intersect in a line parallel to $\widehat{\Omega}_1 \times \widehat{\Omega}_2$. We define P_1 and P_2 as being the points in the intersection line that are closer to the disks centers d_1 and d_2, respectively, as shown in Figure 5.8.

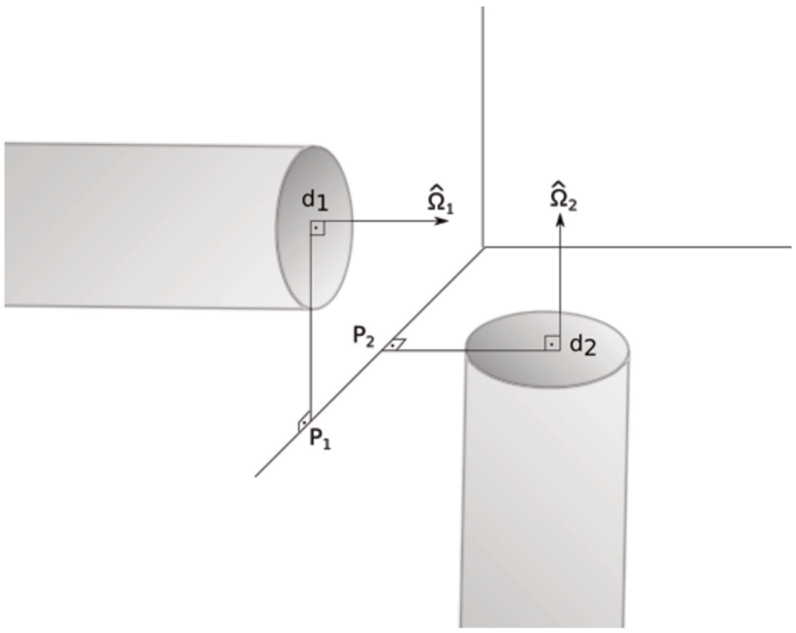


Figure 5.6: Disks of two cylinders.

To find P_1, we minimize $(P_1 - d_2)^2$, which is equivalent to minimizing $|P_1 - d_1|$. The minimization can be done by applying the Lagrange multipliers with two constraints:

$$(P_1 - d_1) \cdot \widehat{\Omega}_1 = 0 \tag{5.4a}$$

$$(P_1 - d_2) \cdot \widehat{\Omega}_2 = 0 \tag{5.4b}$$

The constraints presented in Equation 5.4 ensure that P_1 is in a line perpendicular to both $\widehat{\Omega}_1$ and $\widehat{\Omega}_2$. Applying the Lagrange multipliers:

$$\mathcal{L} = (P_1 - d_1)^2 - \lambda(P_1 - d_1) \cdot \widehat{\Omega}_1 - \mu(P_1 - d_2) \cdot \widehat{\Omega}_2 \tag{5.5}$$

From the optimality condition, for which $\nabla\mathcal{L} = 0$, one has:

$$P_1 = d_1 + \frac{\lambda\widehat{\Omega}_1}{2} + \frac{\mu\widehat{\Omega}_2}{2} \tag{5.6}$$

Replacing Equation 5.4a into Equation 5.6:

$$\lambda = -\mu(\widehat{\Omega}_1 \cdot \widehat{\Omega}_2) \tag{5.7}$$

Substituting Equations 5.4b and 5.7 into 5.6 yields:

$$\mu = \frac{-2(d_1 - d_2) \cdot \widehat{\Omega}_2}{1 - (\widehat{\Omega}_1 \cdot \widehat{\Omega}_2)^2} \tag{5.8}$$

Replacing Equation 5.8 into 5.7:

$$\lambda = \frac{2[(d_1 - d_2) \cdot \widehat{\Omega}_2] \cdot (\widehat{\Omega}_1 \cdot \widehat{\Omega}_2)}{1 - (\widehat{\Omega}_1 \cdot \widehat{\Omega}_2)^2} \tag{5.9}$$

Replacing Equations 5.9 and 5.8 into 5.6:

$$P_1 = d_1 + \frac{[(d_1 - d_2) \cdot \widehat{\Omega}_2] \cdot ((\widehat{\Omega}_1 \cdot \widehat{\Omega}_2) \cdot \widehat{\Omega}_1 - \widehat{\Omega}_2)}{1 - (\widehat{\Omega}_1 \cdot \widehat{\Omega}_2)^2} \tag{5.10}$$

We define $d_{12} = d_2 - d_1$ and $\Delta_1^2 = (P_1 - d_1)^2$, and rewrite Equation 5.10 as:

$$\Delta_1^2 = \frac{(d_{12} \cdot \widehat{\Omega}_2)^2 \cdot ((\widehat{\Omega}_1 \cdot \widehat{\Omega}_2)^2 - 2(\widehat{\Omega}_1 \cdot \widehat{\Omega}_2)^2 + 1)}{(1 - (\widehat{\Omega}_1 \cdot \widehat{\Omega}_2)^2)^2} \tag{5.11}$$

Simplifying Equation 5.11:

$$\Delta_1^2 = \frac{(d_{12} \cdot \widehat{\Omega}_2)^2}{1 - (\widehat{\Omega}_1 \cdot \widehat{\Omega}_2)^2} \tag{5.12}$$

Similarly for disk 2:

$$\Delta_2^2 = \frac{(d_{12} \cdot \widehat{\Omega}_1)^2}{1 - (\widehat{\Omega}_1 \cdot \widehat{\Omega}_2)^2} \tag{5.13}$$

A necessary, but insufficient, condition for the overlap to occur is that both Δ_1 and Δ_2 have to be less than the cylinder radius, $\frac{D}{2}$. If this condition is satisfied, the intersection line crosses both disks through segments of length $2\delta_1$ and $2\delta_2$, as presented in Figure 5.7.

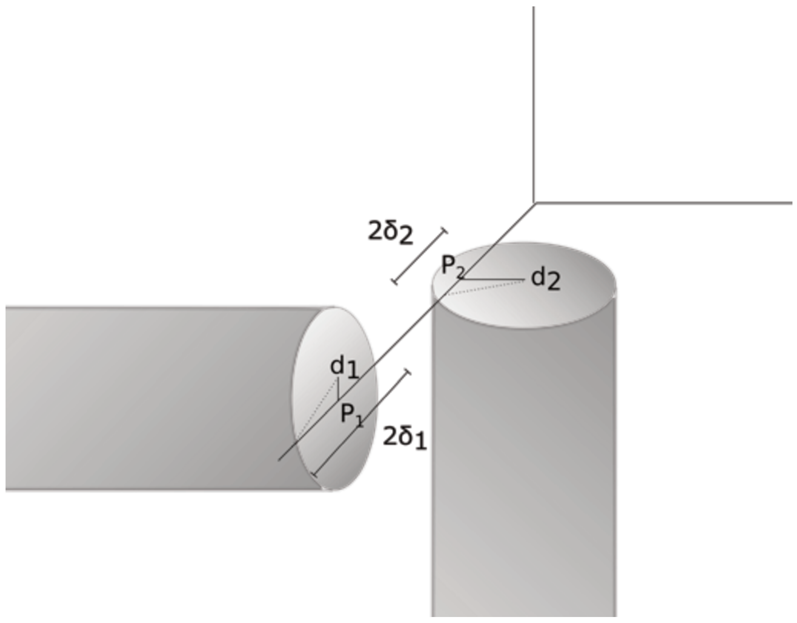


Figure 5.7: Disks of two cylinders.

The expressions to calculate δ_1 and δ_2 are presented in Equation 5.14.

$$\begin{aligned} \delta_1 &= \sqrt{\frac{D^2}{4} - \Delta_1^2} \\ \delta_2 &= \sqrt{\frac{D^2}{4} - \Delta_2^2} \end{aligned} \tag{5.14}$$

The overlap will occur if the condition in Equation 5.15 is true.

$$|P_2 - P_1| = \left| d_{12} \cdot \frac{(\widehat{\Omega}_1 \times \widehat{\Omega}_1)}{|\widehat{\Omega}_1 \times \widehat{\Omega}_2|} \right| \leq \delta_1 + \delta_2 \tag{5.15}$$

An alternative formulation for the calculation of P_1 is given below. Reformulating

Equation 5.10:

$$P_1 = d_1 + \frac{(d_1 \cdot \widehat{\Omega}_2)(\widehat{\Omega}_1 \times N)}{|N|^2} - \frac{(d_2 \cdot \widehat{\Omega}_2)(\widehat{\Omega}_1 \times N)}{|N|^2} \tag{5.16}$$

where $N = \widehat{\Omega}_1 \times \widehat{\Omega}_2$. Knowing that:

$$\widehat{\Omega}_2 \times ((\widehat{\Omega}_1 \times N) \times d_1) = (\widehat{\Omega}_2 \cdot d_1)(\widehat{\Omega}_1 \times N) - (\widehat{\Omega}_2 \cdot (\widehat{\Omega}_1 \times N))d_1 \tag{5.17}$$

one can write as:

$$(\widehat{\Omega}_2 \cdot d_1)(\widehat{\Omega}_1 \times N) = \widehat{\Omega}_2 \times ((\widehat{\Omega}_1 \times N) \times d_1) + (\widehat{\Omega}_2 \cdot (\widehat{\Omega}_1 \times N))d_1 \tag{5.18}$$

Working on the first term of equation 5.18:

$$\begin{aligned} -\widehat{\Omega}_2 \times (d_1 \times (\widehat{\Omega}_1 \times N)) &= (d_1 \cdot n)(\widehat{\Omega}_1 \times \widehat{\Omega}_2) + (d_1 \cdot \widehat{\Omega}_1)(\widehat{\Omega}_2 \times N) \\ &= (d_1 \cdot n)n + (d_1 \cdot \widehat{\Omega}_1)(\widehat{\Omega}_2 \times N) \end{aligned} \tag{5.19}$$

Now working on the second term of Equation 5.18:

$$(\widehat{\Omega}_2 \cdot (\widehat{\Omega}_1 \times N))d_1 = (N \cdot (\widehat{\Omega}_2 \times \widehat{\Omega}_1))d_1 = -|N|^2 d_1 \tag{5.20}$$

Replacing Equations 5.19 and 5.20 into Equation 5.18, and then applying the result in Equation 5.16:

$$P_1 = \frac{d_1|n|^2 + (d_1 \cdot N)N + (d_1 \cdot \widehat{\Omega}_1)(\widehat{\Omega}_2 \times N) - d_1|N|^2 - (d_2 \cdot \widehat{\Omega}_2)(\widehat{\Omega}_1 \times N)}{|N|^2} \tag{5.21}$$

which gives:

$$P_1 = \frac{(d_1 \cdot n)n + (d_1 \cdot \widehat{\Omega}_1)(\widehat{\Omega}_2 \times N) - (d_2 \cdot \widehat{\Omega}_2)(\widehat{\Omega}_1 \times N)}{|N|^2} \tag{5.22}$$

Disk-rim overlap - (Dr. Flavio Romano, Università Ca' Foscari Venezia, personal communication, 2019)

We define the variables as follows:

- d_j : center of disk j
- r_i : center of cylinder i
- w_j, v_j, u_j : axis system fixed on cylinder j ($\widehat{\Omega}_j = u_j$)

- U_i : point on cylinder i that is the closest to d_j
- P_d : point on the disk j that is the closest to cylinder i
- P_c : point on cylinder i that is the closest to disk j
- ϕ : angle between w_j and $\mathbf{d_j P_d}$

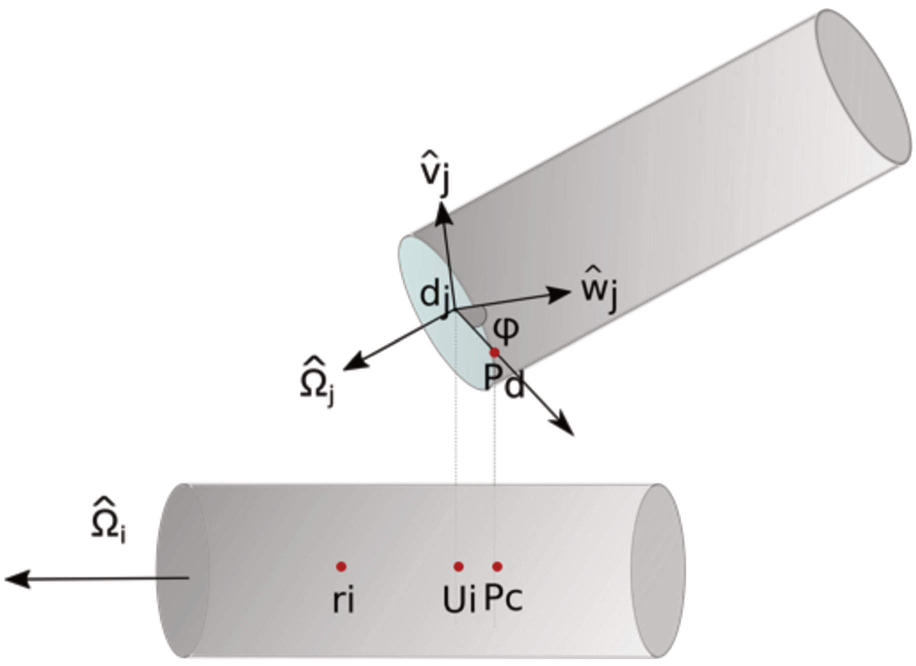


Figure 5.8: Disk-rim configuration.

U_i is obtained from:

$$U_i = r_i + [(d_j - r_i) \cdot \widehat{\Omega}_i]\widehat{\Omega}_i \tag{5.23}$$

First, we test the following conditions:

1. If $|d_j - U_i| > d$: there is no overlap
2. If $|d_j - U_1| < d/2$ and $|d_j - r_i| > L/2$: the overlap would be a disk-disk kind and not a disk-rim, therefore, we do not test it here.
3. If $|d_j - u_1| < d/2$ and $|(d_j - r_i)| < L/2$: there is an overlap (the center of the disk j is within cylinder i).

Test number 3 is a sufficient, but not necessary, condition for the overlap to occur, since another point can be touching cylinder j even if d_j is not within cylinder i. Hence, if condition 3 is unsatisfied, we have to find p_d, *i. e.*, the closest point in disk j to cylinder i. Arbitrary points on disk j (d), and on the line of cylinder i (c) are defined as:

$$d = d_j + \frac{d}{2}\cos(\phi)\widehat{w}_j + \frac{d}{2}\sin(\phi)\widehat{v}_j \tag{5.24a}$$

$$c = r_i + \lambda\widehat{\Omega}_i \tag{5.24b}$$

We define $\Gamma = (d-c)^2$.

$$\begin{aligned}\Gamma =& \mathbf{d_j r_2}^2 + R^2 + \lambda^2 + 2R\cos\phi(\mathbf{d_j r_i}\cdot\widehat{w}_j) + 2r\sin\phi(\mathbf{d_j r_i}\cdot\widehat{v}_j)\\ &- 2\lambda(\mathbf{d_j r_i}\cdot\widehat{\Omega}_i) - 2\lambda r\cos\phi(\widehat{w}_j\cdot\widehat{\Omega}_i) - 2\lambda r\sin\phi(\widehat{v}_j\cdot\widehat{\Omega}_i)\end{aligned} \tag{5.25}$$

P_c and P_d are the points that minimize Γ, therefore:

$$\frac{\partial\Gamma}{\partial\lambda} = 0 = \lambda - r\cos\phi(\widehat{w}_j\cdot\widehat{\Omega}_i) - r\sin\phi(\widehat{v}_j\cdot\widehat{\Omega}_i) - (\mathbf{d_j r_i}\cdot\widehat{\Omega}_i) \tag{5.26a}$$

$$\frac{\partial\Gamma}{\partial\phi} = 0 = \sin\phi[\lambda(\widehat{w}_j\cdot\widehat{\Omega}_i) - (\mathbf{d_j r_i}\cdot\widehat{w}_j)] - \cos\phi[\lambda(\widehat{v}_j\cdot\widehat{\Omega}_i) - (\mathbf{d_j r_i}\cdot\widehat{v}_j)] \tag{5.26b}$$

Rewriting Equation 5.26b gives:

$$\frac{\sin\phi}{\cos\phi} = \frac{\lambda(\widehat{v}_j\cdot\widehat{\Omega}_i) - (\mathbf{d_j r_i}\cdot\widehat{v}_j)}{\lambda(\widehat{w}_j\cdot\widehat{\Omega}_i) - (\mathbf{d_j r_i}\cdot\widehat{w}_j)} \tag{5.27}$$

If the numerator and denominator of Equation 5.27 are taken as the catheti of a triangle, the hypotenuse can then be found to give the expressions for $\cos\phi$ and $\sin\phi$. Once we have these expressions, they are applied into Equation 5.26a, resulting in an equation for λ. Since the resulting expression is not trivial, a numerical method such as Newton-Raphson or bisection method is used to find λ.

Once P_d is obtained, we define $T = P_d - r_i$, and calculate the components of T that are parallel $T_{\|}$ and perpendicular $T_{\perp}$ to $\widehat{\Omega}_i$.

$$T_{\|} = (t\cdot\widehat{\Omega}_1)\widehat{\Omega}_1 \tag{5.28a}$$

$$T_{\perp} = T - t_{\|} \tag{5.28b}$$

Finally, the overlap only occurs if $|T_{\|}| \leq L/2$ and $|T_{\perp}| \leq D/2$.

5.4 Distribution functions

The calculation of the distribution functions is important to study the structure of fluids. Besides the radial distribution function $g(r)$, the parallel $g^{||}(r)$ and perpendicular $g^{\perp}(r)$ distribution functions are useful to identify layering and hexagonal order, respectively. The definition of the $g(r)$ is given in section 1.1, but, in general, distribution functions are the probability of finding a particle in the system at some specific places divided by the probability of the same condition in an ideal gas ($N_{\mathrm{vol}}/N_{\mathrm{ideal}}$). A Fortran 90 pseudo-code for the calculation of the radial, parallel and perpendicular distribution functions is given in Listing 5.1.

5.4.1 Radial distribution function

The radial distribution function $g(r)$ is the probability of finding two particles in a distance r from each other divided by the same probability in an ideal gas. In a finite system, we find the number of particles N_{vol} lying between a distance r_{lower} and r_{upper} from each other, that is, whose centers of mass are found in the volume represented by the gray area in Figure 5.9. Since in an ideal gas the probability distribution function is uniform, the number of particles lying in the same volume is equal to $N_{\mathrm{ideal}} = \rho 4\pi(r^3_{\mathrm{upper}} - r^3_{\mathrm{lower}})/3$, where ρ is the number density.

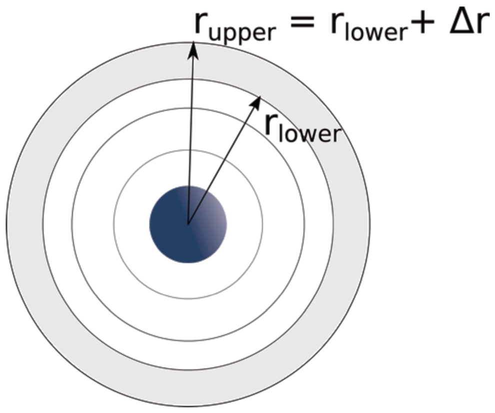


Figure 5.9: Illustration of the numerical calculation of the radial distribution function.

5.4.2 Parallel distribution function

Given the vector $\mathbf{r}_{ij}$ linking two particles i and j centers of masse, $g^{||}(r)$ is the probability that $|r^{||}_{ij}| = r_{ij} \cdot \widehat{n}$ lies between a distance r_{lower} and r_{upper}, where $\widehat{n}$ is the phase director, divided by the same probability in an ideal gas. N_{vol} in this case is the number of particles whose vector $r_{ij}^{||}$ lies in the volume represented by the gray area in Figure 5.10. $N_{\text{ideal}} = \rho 2 L_{\text{box}}^2 * \Delta r_{ul}$, where $\Delta r_{ul} = r_{\text{upper}} - r_{\text{lower}}$ and L_{box} is the dimension of the simulation box. Since we often work with a simulation box with different lengths, we take L_{box} as the larger length of the box.

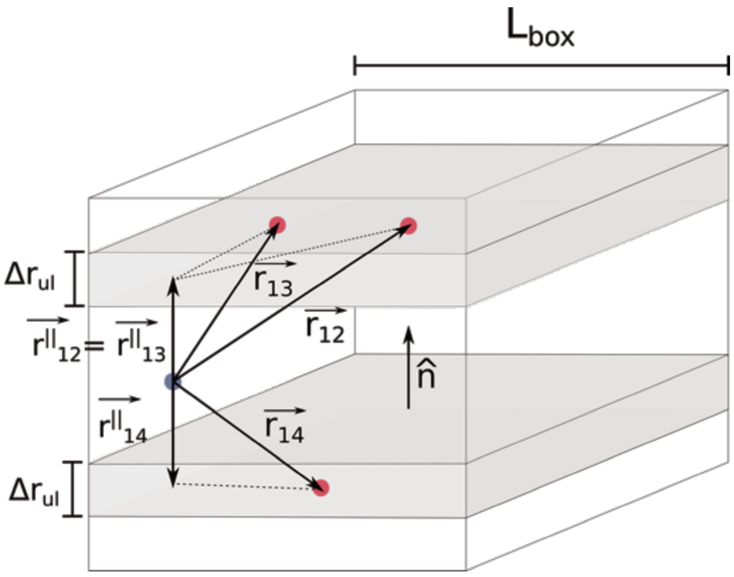


Figure 5.10: Illustration of the numerical calculation of the parallel distribution function.

5.4.3 Perpendicular distribution function

To calculate $g^{\perp}(r)$, we take the parallel component $r_{ij}^{\perp}$ of $r_{ij} \cdot \widehat{n}$, instead. The relevant volume is represented by the gray area in Figure 5.11 and $N_{\text{ideal}} = \rho L_{\text{box}} \pi (r_{\text{upper}}^2 - r_{\text{lower}}^2)$.

```
subroutine distribution_functions()
 const1 = 4.d0*pi*rho/3.d0
 const2 = 2.d0*rho*maxbox*maxbox
 const3 = pi*rho*maxbox
 hist(:) = 0.d0
 hist_par(:) = 0.d0
 hist_per(:) = 0.d0
 gr(:) = 0.d0
 gr_par(:) = 0.d0
 gr_per(:) = 0.d0
```

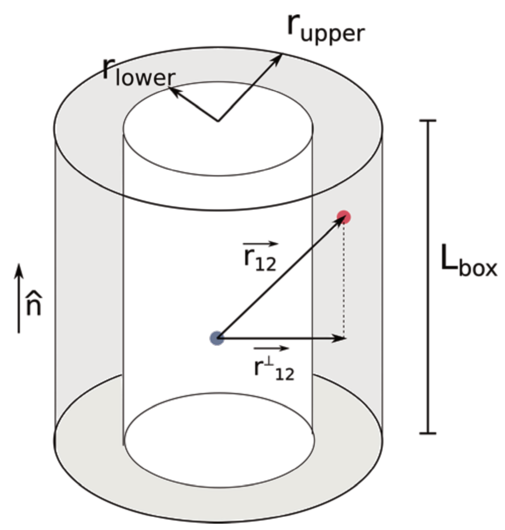


Figure 5.11: Illustration of the numerical calculation of the perpendicular distribution function.

```
do i = 1,n-1
     do j = i+1,n
            rl1(:) = r(:,i)
            rl2(:) = r(:,j)
            rl12(:) = rl2(:) - rl1(:)
            !Minimum Image
            rl12(:) = rl12(:) - boxl(:)*dnint(rl12(:)/boxl(:))
            rl12sq = rl12(1)*rl12(1) + rl12(2)*rl12(2) + rl12(3)*rl12(3)
            modrl = dsqrt(rl12sq)
            rpar = dabs(rl12(1)*pd(1) + rl12(2)*pd(2) + rl12(3)*pd(3))
            rper = dsqrt(modrl*modrl - rpar*rpar)
            Call rdf_iso()
            Call rdf_par()
            Call rdf_per()
     end do
end do
!Normalizing hist
Do bin =1,nbins
                    rlower = dble(bin - 1)*delr
                    rupper = rlower + delr
                    nideal = const1*(rupper**3.0 - rlower**3.0)
```

```
                      gr(bin) = hist(bin)/dble(n)/nideal
                      rlower = dble(bin - 1)*delr_par
                      rupper = rlower + delr_par
                      nideal = const2*delr_par
                      gr_par(bin) = hist_par(bin)/dble(n)/nideal
                      nideal = const3*(rupper**2.0 - rlower**2.0)
                      gr_per(bin) = hist_per(bin)/dble(n)/nideal
 end do
 return
end subroutine
subroutine rdf_iso()
              bin = floor(modrl/delr) + 1
              if (bin .le. nbins) then
                      hist(bin) = hist(bin) + 2.d0
              end if
end subroutine
subroutine rdf_par()
              bin = floor(rpar/delr_par) + 1
              if (bin .le. nbins) then
                      hist_par(bin) = hist_par(bin) + 2.d0
              end if
end subroutine
subroutine rdf_per()
              bin = floor(rper/delr_par) + 1
              if (bin .le. nbins) then
                      hist_per(bin) = hist_per(bin) + 2.d0
              end if
end subroutine
```

Listing 5.1: Pseudo-code to calculate distribution functions.

5.5 Equation of State for the Isotropic Phase

Peters et al. (2020) developed algebraic equations of state for the liquid crystalline phases of hard rods based on previous numerical calculations. Although an analytical description of phase behavior of hard nonspherical particles can be complex, an equation of state for the isotropic phase is easily obtained through Equation 2.9 (Lee, 1987), provided that an expression for the excluded volume of the particle is available.

Onsager (1949) formulated an expression for the excluded volume of cylinders 4.1. Ibarra-Avalos et al. (2007), however, found a small deviation between simulations and Onsager results, and proposed a new expression that reproduced Monte Carlo simulations and semianalytical values:

$$\frac{v^{\text{exc}}_{\text{IGR}}}{v_{\text{cyl}}} = 2 + \frac{8}{\pi} + \left(\frac{2}{L/D} + L/D\frac{8}{\pi}\right)\sin\theta + 2\left[\left(\frac{3\pi}{4} - 1\right)|\cos\theta| + \left(3 - \frac{4}{\pi}\left(10\frac{\pi}{4}\right)\cos^2\theta\right)\right] \tag{5.29}$$

where v_{cyl} is the volume of the cylinder and θ is the relative orientations. We define Γ as the average over θ of the ratio between the excluded volume and the volume of the particle, considering an isotropic distribution function:

$$\Gamma = \left\langle \frac{v^{\text{exc}}}{v_{\text{cyl}}} \right\rangle_\theta = \frac{\int\limits_0^{2\pi}\int\limits_0^{\pi} \frac{v^{\text{exc}}}{v_{\text{cyl}}} \sin\theta \mathrm{d}\theta \mathrm{d}\phi}{\int\limits_0^{2\pi}\int\limits_0^{\pi} \sin \mathrm{d}\theta \mathrm{d}\phi} \tag{5.30}$$

The Onsager's and Ibarra-Avalos et al.'s averages are:

$$\Gamma^{\text{IGR}} = 2 + \frac{1}{3} + 2L/D + \frac{\pi}{2L/D} + \frac{\pi}{2} + \frac{8}{\pi} - \frac{4}{3\pi} \tag{5.31}$$

$$\Gamma^{\text{ONS}} = 2L/D + \pi + 3 + \frac{\pi}{2L/D} \tag{5.32}$$

The difference between Γ^{ONS} and Γ^{IGA} is 0.116, that is, when the average over the relative orientations is taken, the models are quite similar.

Considering the residual part of Equation 2.9, the reduced pressure for a system of hard cylinders is obtained:

$$P^* = \frac{Pv_\mathrm{p}}{k_B T} = \frac{\Gamma}{8}\frac{4\eta^2 - 2\eta^3}{(1-\eta)^3} \tag{5.33}$$

5.6 Simulation Details

In this chapter, the number of cycles used is about twice the total number of particles. In each cycle, either the position and orientation of a particle or the total volume is chosen randomly to be changed. For this purpose, a random number between $1 \leq \zeta \leq N + 1$ is generated, if $\zeta \leq N$ a particle is moved, otherwise, the volume is changed. The codes are provided in the appendix. There is a main code for the hard cylinder system and the subroutines that are common for multiple systems are provided in three different modules:

- MC NPT for Hard Cylinders (C.2.1)
- Module to set global variables (C.2.2)
- Module to create initial configurations (C.2.3)
- Module with the main subroutines (including the overlap checks and calculation of the potentials) (C.2.4)

We used the OVITO visualization tool (Stukowski, 2010) to make the snapshots of the simulations. We added a python script to add color to the particles according to the angle between its orientation and the phase director. The scripts are provided in the appendix C.2.5.

5.7 Results

The systems were equilibrated using 5.85×10^6 Monte Carlo steps, with additional production runs of 1.5×10^5 steps. Simulations of cylindrical disks and rods with several different aspect ratios L/D.

The colors and symbols used to represent the rods and disks phases are outlined in Tables 5.1 and 5.2, respectively. Figures 5.12 and 5.13 present illustrations of each phase. The phases were identified using the radial, parallel and perpendicular distribution functions, by analyzing snapshots, and by calculating the nematic, smectic and hexatic order parameter, as described on sections 1.2.1, 1.2.2 and 1.2.3. To calculate the hexatic order parameter we have defined the nearest neighbors as the particles at a distance less than $1.3D$ from the central particle. The simulation data obtained for all aspect ratios are provided in the appendix C.1.

Table 5.1: Color and symbol used to represent rod phases.

Color	Phase	Notation	Symbol
red	isotropic	I	circle
yellow	nematic	N	triangle
green	smectic A	SmA	square
blue	Crystal	X	diamond

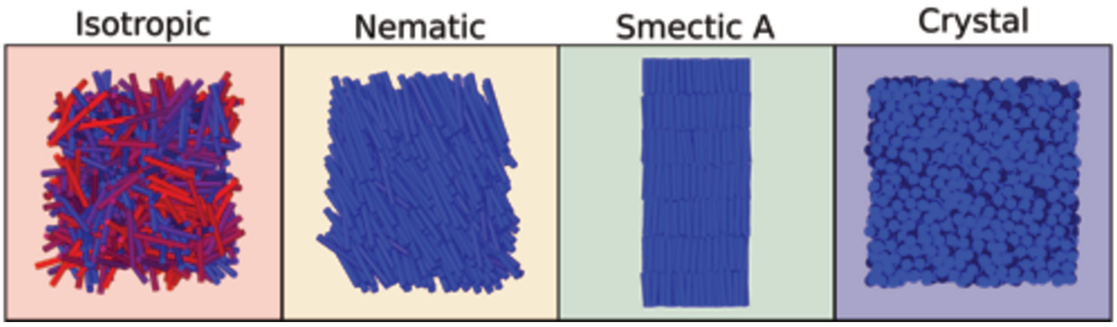


Figure 5.12: Illustration of rod phases.

Table 5.2: Color and symbol used to represent disk phases.

Color	Phase	Notation	Symbol
red	isotropic	I	circle
yellow	nematic	N	triangle
purple	cubatic	Cub	squares
blue	columnar	C	diamond

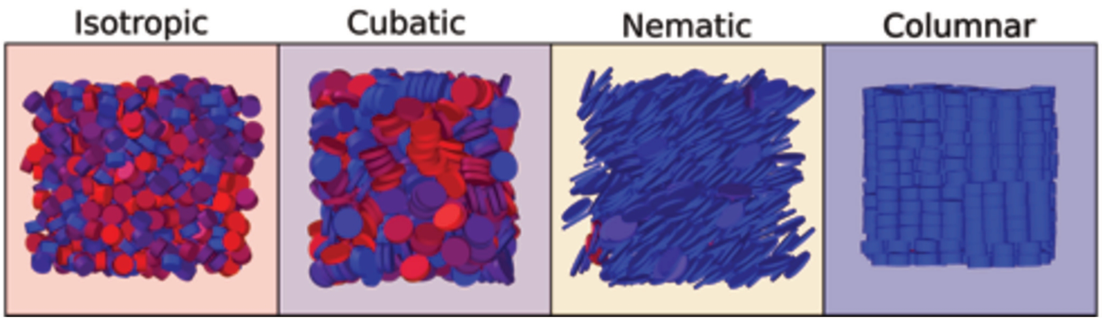


Figure 5.13: Illustrations of disk phases.

5.7.1 Rods

The initial configuration for the rods was parellel cylinders at a packing fraction $\eta \approx 0.77$ in an elongated box with 6 layers of particles parallel to the z-axis, since Dussi et al. (2018) showed that a mechanical unstable columnar phase is formed in systems with a small number of layers (≈ 4). Moreover, Bolhuis and Frenkel (1997) argue that the pressure of a smectic can become anisotropic, hence, to obtain an isotropic pressure we have used the floppy-box Monte Carlo, as describe previously. The simulated aspect ratios and number of particles are outlined in Table 5.3:

Table 5.3: Number of particles in the simulations of rods.

L/D	N	L/D	N
2.5	968	6.25	1350
3.0	1152	6.5	1350
3.25	1152	7.0	1536
3.5	1352	7.5	1536
5.0	1176	10.0	1944
6.0	1350		

A sketched phase diagram for hard cylinders with aspect ratios ranging from $L/D = 2.5$ to 10 is displayed in Figure 5.14, with the packing fraction η as a function of L/D. The phase behavior of the system is very rich, exhibiting isotropic, nematic, smectic A, and crystalline phases. For $L/D < 3.0$, no liquid crystalline phase was observed. This fact is in accordance with Onsager's theories, in the sense that, since the ratio between the covolume and volume of rods with lower L/D are not considerably larger than the one of a sphere, there is enough room for the orientational entropy to be maximize. The excluded volume effects then are insufficient to promote an organized phase.

At sufficiently high densities and aspect ratios, the maximization of the orientational entropy is limited by the geometry, that is, the excluded volume. The system then tends to promote orientational order to increase the translational entropy, minimizing the free energy. For any L/D, an isotropic phase is formed below a certain packing fraction. The longer the aspect ratio, the smaller is the packing fraction necessary to promote an organized phase, as longer particles have a larger excluded volume with respect to their volumes.

For increasing L/D, the first seen organized phase is a smectic A at $L/D = 3.0$. At $L/D = 6.5$, a nematic phase region appears. This result agrees with previous simulations

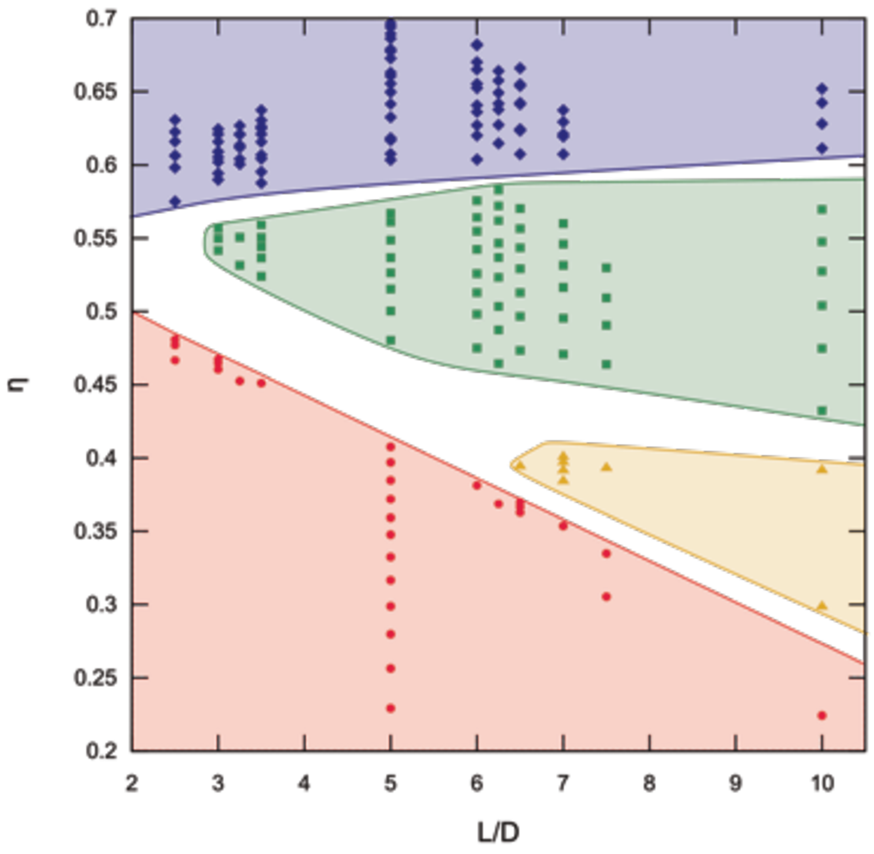
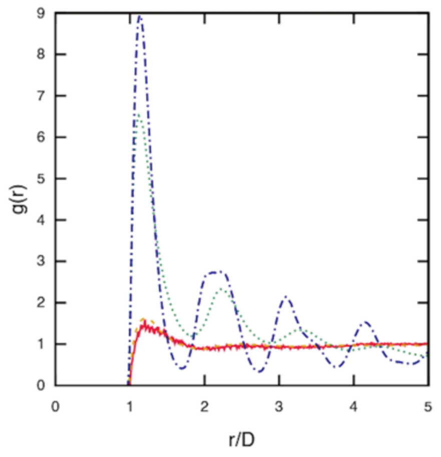


Figure 5.14: Sketched phase diagram of cylindrical rods. Color and symbol codes are presented in Table 5.1.

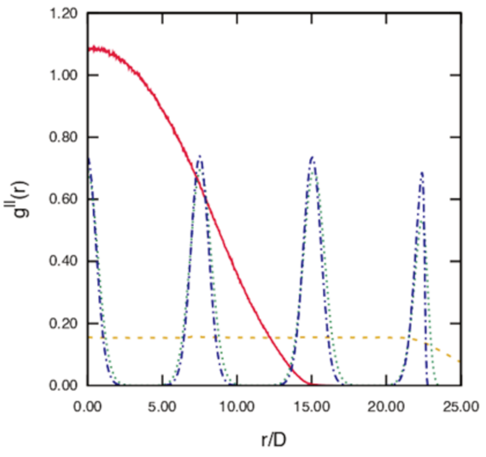
of HSC (Bolhuis and Frenkel, 1997; McGrother et al., 1996) that showed that the smectic phase stabilizes first, at shorter aspect ratios when compared to the nematic phase.

The sketched phase diagram in Figure 5.14 suggests that there might be an I-SmA-X triple point around $L/D = 3$, and an I-N-SmA around $L/D = 6.5$. The prediction of the I-SmA-X triple point location is quite close to the one in a system of hard spherocylinders, found at $L/D = 3.1$. On the other hand, for HSC, Bolhuis and Frenkel (1997) located an I-N-SmA triple point at $L/D = 3.7$, instead. That is, the nematic phase stabilizes at shorter aspect ratios in the HSC system when compared to the HC fluid. The region where the transition to a smectic A phase occurs directly from the isotropic one extends from $3 \geq L/D \leq 6.25$, whereas, in a system of HSC, the region is considerably shorter, ranging from about $3 < L/D < 4$.

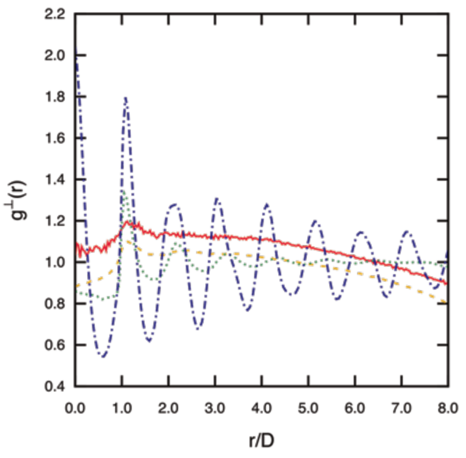
Figure 5.15 presents the radial, parallel, and perpendicular distribution functions of cylinders with $L/D = 7.0$ for increasing pressure. At $p^* = 3.96$ and $p^* = 4.40$, the system forms an isotropic and a nematic phase, respectively, and, thefore, do not exhibit positional order in any direction. At $p^* = 7.70$, the system forms a smectic A phase, exhibiting positional order in the parallel direction, but not in the perpendicular direction. At $p^* = 9.90$, there is long range positional ordering both in the parallel and perpendicular directions.



(a)



(b)

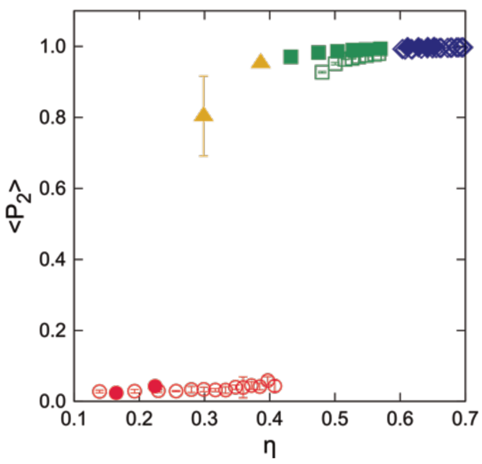


(c)

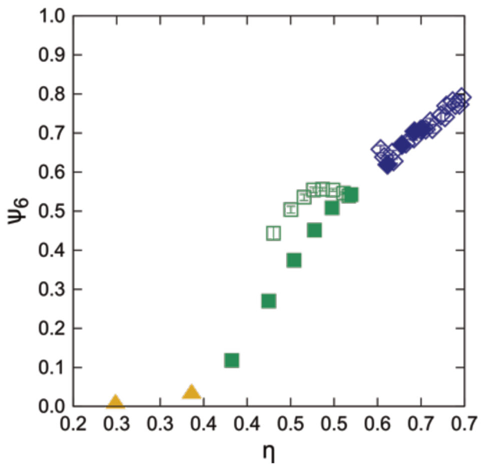
Figure 5.15: Distribution functions of cylinders with $L/D = 7.0$. $p^* = 3.96$: continuous red line, $p^* = 4.40$: yellow dashed line, $p^* = 7.70$: green dotted line, $p^* = 9.90$: dash-dotted line. Color code is outlined in Table 5.1.

The order parameters were calculated to help identify the phases. Figure 5.16 shows the nematic, hexatic and smectic order parameters for $L/D = 10$ and $L/D = 5$.

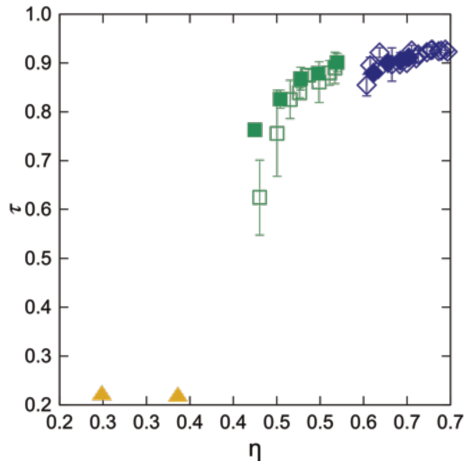
The nematic order parameter is only sufficient to identify the isotropic phase. The calculation of the smectic and hexatic order parameters (τ and ψ_6, respectively) are valuable to distinguish between the other phases. The nematic phase presents low values of the hexatic and smectic order parameters, while for the smectic phase $\tau > 0.5$ and $\psi_6 < 0.6$. The crystalline phase exhibits both layering and in-plane hexagonal order,



(a) Nematic order parameter



(b) Hexatic order parameter

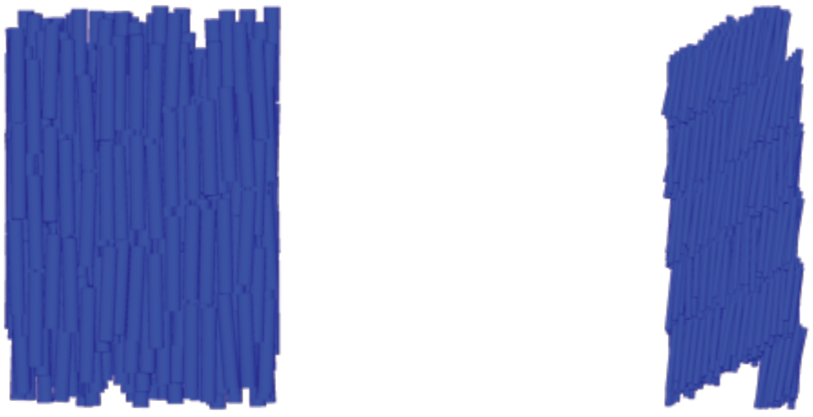


(c) Smectic order parameter

Figure 5.16: Order parameters for hard cylinders with $L/D = 10$ (closed symbols) and $L/D = 5$ (open symbols). Color code is outlined in Table 5.1.

thus, τ and ψ_6 are closer to one. The columnar phase presents hexagonal order but no layering.

We have observed the formation of a columnar phase when a simulation box with only two layers of particles was used, in accordance to the results observed by Dussi et al. (2018), who showed that an unstable columnar phase is formed in a system of HSC with a small number of layers. Moreover, a smectic C phase is observed when a rigid simulation box is used, since, as pointed out by Bolhuis and Frenkel (1997), the pressure of a smectic phase is anisotropic and a floppy-box might be used to obtain an isotropic pressure (Figure 5.7.1).


(a) Columnar (b) Smectic C

5.7.2 Disks

The initial configuration was roughly a cubix box with $\eta \approx 0.75$. The number of particles for each L/D is outlined in Table 5.4. The simulated aspect ratios and number of particles in each system are shown in Table 5.4.

Table 5.4: Number of particles in the simulations of disks.

L/D	N	L/D	N
0.05	540	0.2	625
0.1	640	0.25	864
0.11	576	0.3	720
0.12	528	0.35	612
0.125	528	0.5	686
0.15	825		

The sketched phase diagram for cylindrical disks with aspect ratios ranging from $L/D = 0.05$ to 0.5 is displayed in Figure 5.17. The results agree qualitatively with the theoretical predictions of Wensink and Lekkerkerker (2009). For flatter disks, there is a nematic phase region that becomes narrower as L/D increases, until it completely vanishes around $L/D = 0.1$. For $L/D > 0.3$, there is a transition from the isotropic directly to the columnar phase. It is important to note, however, that the the aforementioned authors do not take into account the formation of the cubatic phase, reported by Veerman and Frenkel (1992) in the simulation of cut spheres. In addition to that, the theoretical packing fractions predicted in the phase transitions are larger than our results. The values of $\eta_{IN} \approx \pi L/D$ for the isotropic-nematic transition and $\eta_{NC} \approx 0.4$ for the nematic-columnar one, however, are in line with the simulations results for cut-spheres (Duncan et al., 2009).

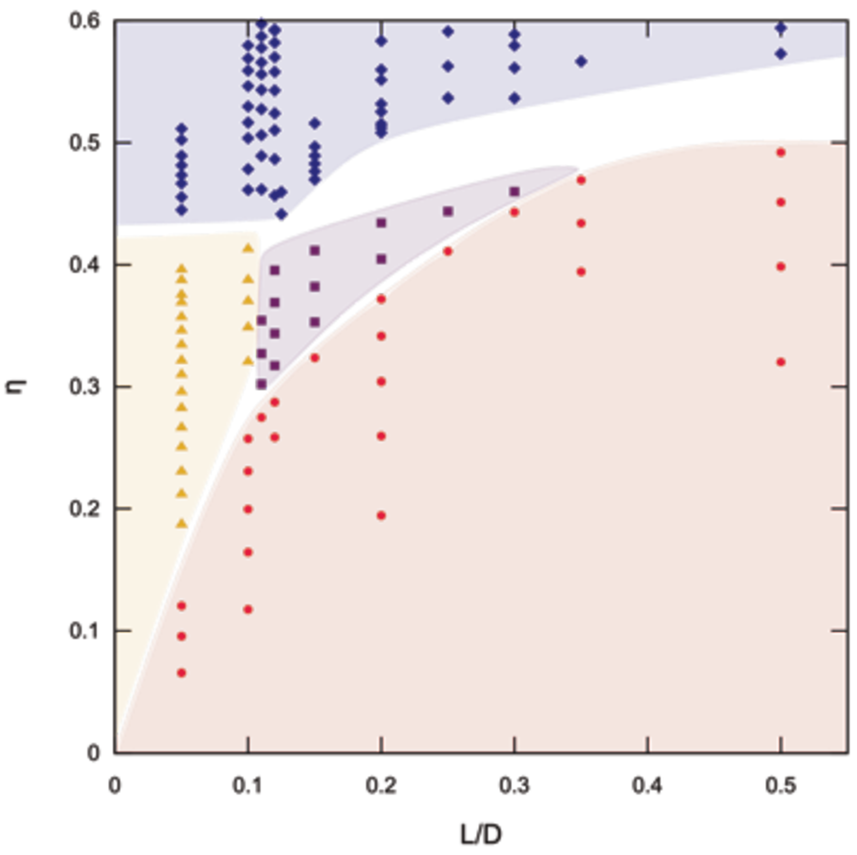
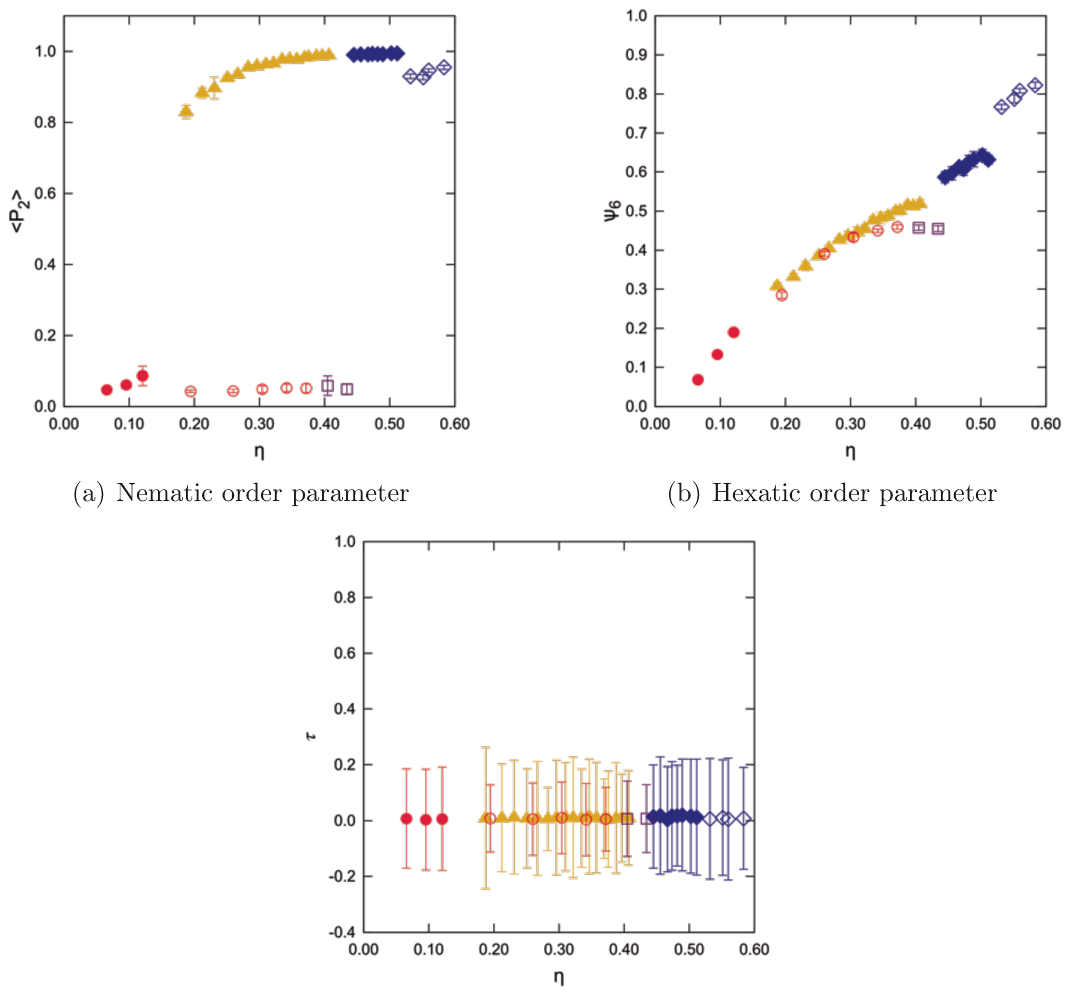


Figure 5.17: Sketched phase diagram of cylindrical disks. Color and symbol codes are presented in Table 5.2.

From $L/D = 0.11$ to 0.3, a cubatic phase appears between the isotropic and columnar phase. In the cubatic phase, the particles tend to assemble in short stacks of about four, five particles, and neighboring columns are usually perpendicular to each other. Veerman and Frenkel (1992) named the phase cubatic to differentiate it from the cubic phase, since, although it has extended cubic orientational order (particles aligned over three perpendicular axes), it does not have translational order. The nematic order parameter characterizes an isotropic phase, however, analyzing the simulations snapshots and distribution functions, the cubatic phase is identified.

Snapshots of simulations forming isotropic, nematic, cubatic, and columnar phases are shown in Figure 5.19. The correspondent radial, parallel, and perpendicular distribution functions are presented in Figure 5.20. Figure 5.18 shows the nematic, hexatic, and smectic order parameters for $L/D = 0.05$ and $L/D = 0.2$.

The radial distribution function of the cubatic phase is quite different from the isotropic



(a) Nematic order parameter (b) Hexatic order parameter

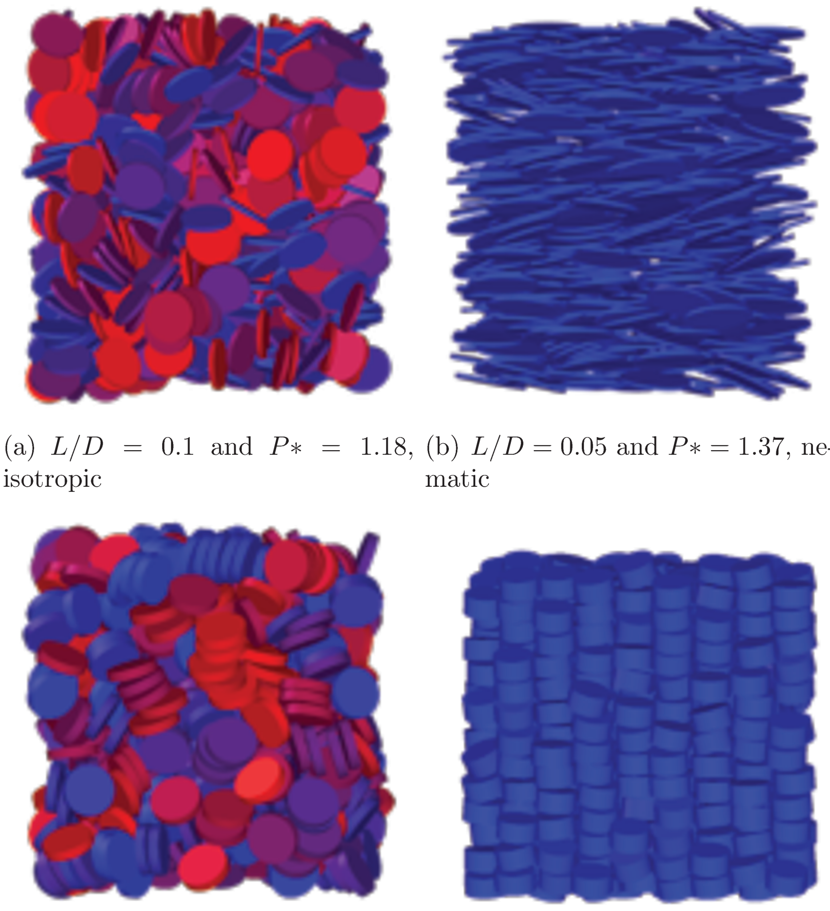
(c) Smectic order parameter

Figure 5.18: Order parameters for hard cylinders with $L/D = 0.05$ (closed symbols) and $L/D = 0.2$ (open symbols). Color code is outlined in Table 5.2.

phase, even though the order parameters are close to zero for both phases. An evidence of the formation of short stacks is the peak at short distances ($L/D < r/D < 2L/D$) in the radial distribution function (purple line in Figure 5.20) of the cubatic phase (Veerman and Frenkel, 1992), which is absent in the $g(r)$ of an isotropic phase (red line in Figure 5.20).

Duncan et al. (2009) simulated cut spheres of $L/D = 0.1$, 0.15, 0.2, 0.25, and 0.3 and, despite the differences in shape, the results are very similar to ours. The authors showed that there is a nematic but no cubatic phase at $L/D = 0.1$, and the opposite is true for $L/D \geq 0.15$. Figure 5.17 shows, however, that a cubatic phase is seen at $L/D = 0.11$, as the nematic phase vanishes. The cubatic phase is apparent until $L/D \approx 0.3$. For larger aspect ratios, only an isotropic and columnar phase are apparent.

Blaak et al. (1999) inquired whether a cubatic phase could be formed in a system


(a) $L/D = 0.1$ and $P* = 1.18$, isotropic (b) $L/D = 0.05$ and $P* = 1.37$, nematic

(c) $L/D = 0.2$ and $P* = 5.50$, cubatic (d) $L/D = 0.5$ and $P* = 9.82$, columnar

Figure 5.19: Snapshots of the simulations of hard cylindrical disks.

of hard cylinders. The authors simulated a system of HC with $L/D = 0.9$ and did not find such a phase. Our results support evidence that the HC fluid does exhibit a cubatic phase, the aspect ratio simulated by Blaak et al. (1999), however, is out of the range in which the phase appears.

5.7.3 Equation of State for the Isotropic phase

We have tested the Lee-Parsons approach - Equation 5.33 - with the excluded volume expressions derived by Onsager (1949) (Equation 5.32) and Ibarra-Avalos et al. (2007) (Equation 5.31). Figure 5.7.3 shows the theoretical predictions and simulation results for rods ($L/D = 2.5$, 5, 7.5 and 10) and disks ($L/D = 0.05$, 0.1, 0.2 and 0.5).

The equations of state are quite accurate for the isotropic phase of both rods and disks with different shape anisotropies. The differences between the EoS formulated with Γ_{ONS} and Γ_{IGR} are unnoticeable. Larger deviations are observed for higher pressures for

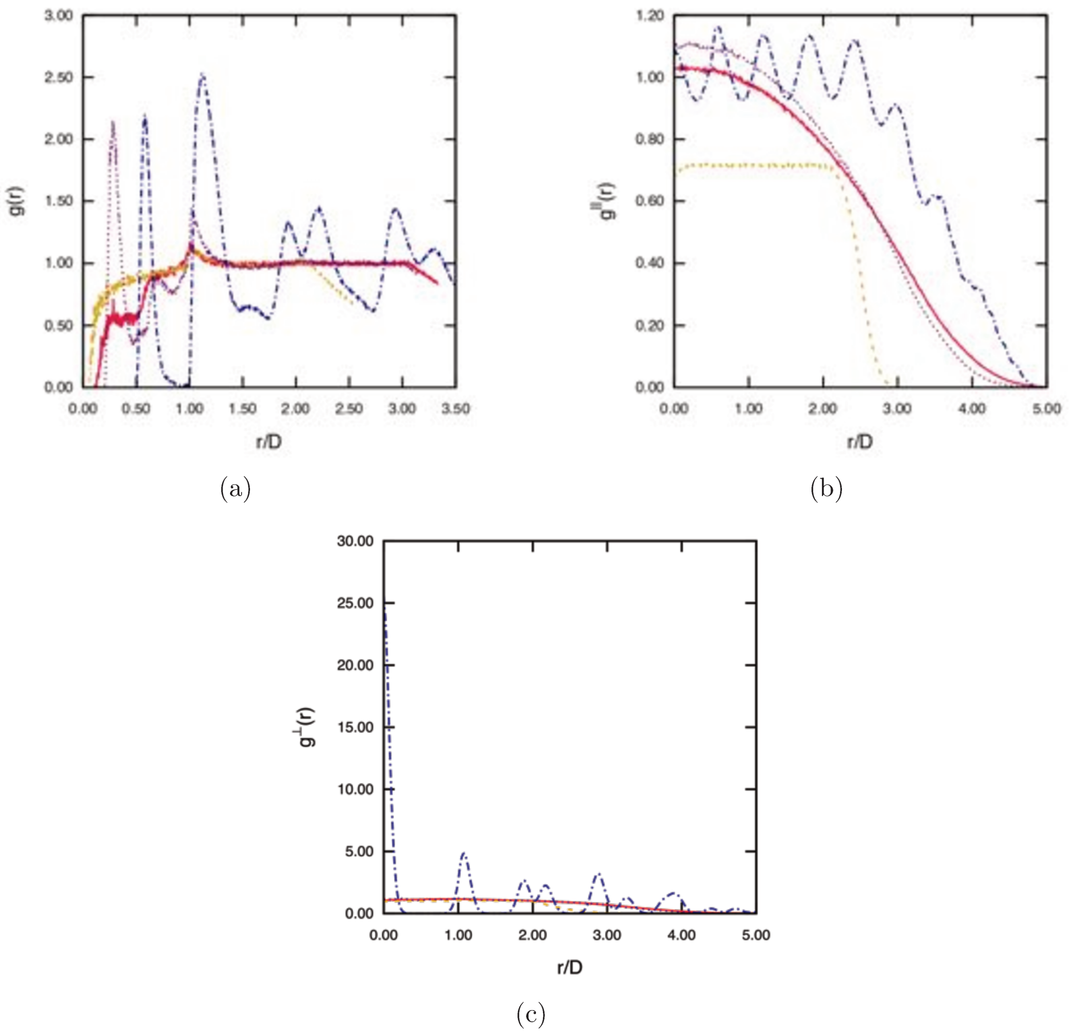
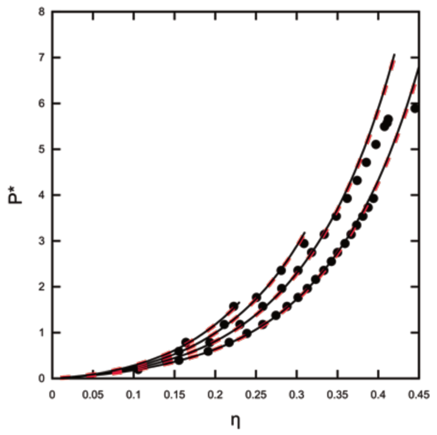


Figure 5.20: Distribution functions of hard cylindrical disks. $L/D = 0.1$ and $P* = 1.18$: continuous line, $L/D = 0.05$ and $P* = 1.37$: dashed line, $L/D = 0.2$ and $P* = 5.50$: dotted line, $L/D = 0.5$ and $P* = 9.82$: dash-dotted line. Color code is outlined in Table 5.2.

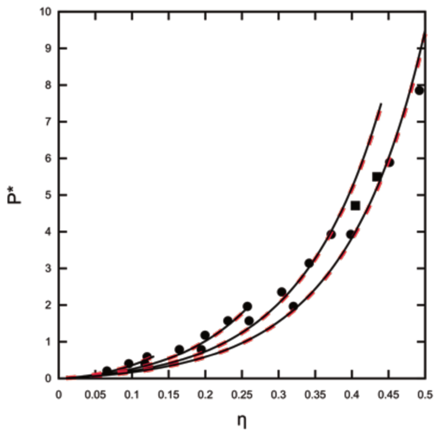
$L/D = 5$. The radial distribution functions (Figure 5.22) show that a peak is formed and becomes larger as the pressure increases, hence, the deviations might be attributed to the start of a transition to the smectic phase. Likewise, for $L/D = 0.2$, the deviations are larger for the cubatic phase represented by the square symbol.

5.8 Conclusions

Simulations of both rod and disk-like hard cylinders with a wide range of aspect ratios were carried out to investigate the boundaries of liquid crystalline phases. The rod-like system exhibited isotropic, nematic, smectic A, and crystalline phases, while the



(a) From left to right: $L/D = 10$, 7.5, 5 and 2.5.



(b) From left to right: $L/D = 0.05$, 0.1, 0.2 and 0.5.

Figure 5.21: Continuous line: Γ_{ONS}, dashed line:Γ_{IGR}, circles: simulations in the isotropic phase, squares:simulations in the cubatic phase.

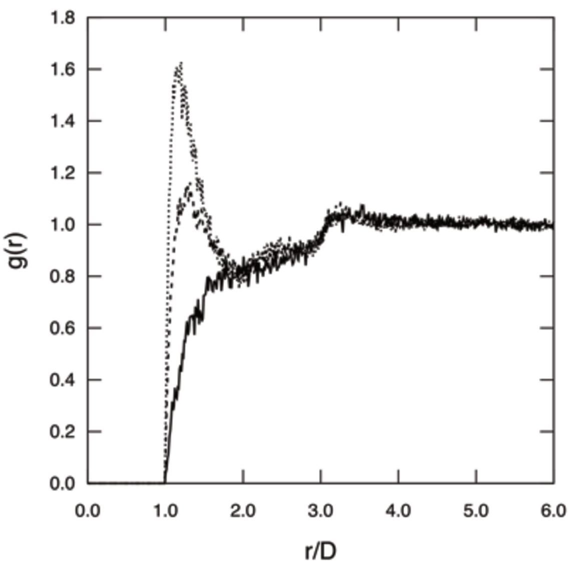


Figure 5.22: Radial distribution function of HC with $L/D = 5$. Continuous line: $P^* = 0.79$, dashed line: $P^* = 3.93$, dotted line: $P^* = 5.65$.

disk-like fluid showed isotropic, nematic, cubatic, and columnar phases. The isotropic phase is stable below a certain packing fraction for any aspect ratio; as the anisotropy increases ($L/D >>$ for rods, and $L/D <<$ for disks), the transition to an organized phase occurs at lower packing fractions. On the other hand, a nematic phase is only found when the particle is sufficiently anisotropic. The results suggest that there are two triple points for rod-like cylinders, an I-SmA-X at about $L/D = 3$ and an I-N-SmA around $L/D = 6.5$. The location of the I-SmA-X triple point is quite close to the one found in a hard spherocylinder fluid at $L/D = 3.1$. The location of the I-N-SmA, however, is considerably different in comparison to the HSC system, in which it occurs at $L/D = 3.7$. For $L/D < 1$, a cubatic phase between the isotropic and columnar phases was found in a small range of aspect ratios ($0.11 < L/D < 0.3$), appearing when the nematic phase vanishes. The results presented on this chapter provide a general mapping of the phase boundaries of the hard cylinder fluid. Further study applying Monte Simulations on the Grand-canonical or Gibbs ensemble could provide the exact locations of the triple points and asses the phase equilibrium of the system.

"We can only see a short distance ahead, but we can see plenty there that needs to be done."

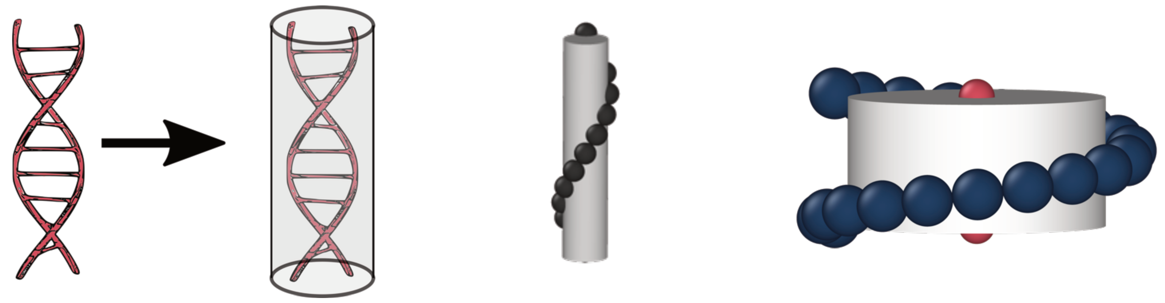
— Alan Turing

6 Exploratory models for cylindrical particles

6.1 Introduction

In chapters 2 and 3, molecules were modeled as ellipsoids in an attempt to capture the essential features of the system of interest, while simplifying theoretical and numerical calculations. On this chapter, we follow the same philosophy, but instead of focusing on properties prediction of industrial relevant fluids, we turn our attention to the investigation of liquid crystalline phase formation in biological systems.

Many biological nanounits adjust their intrinsic material properties due to changes in temperature, concentration, pH, and ionic strength. Charged viruses form different liquid crystalline phases depending on salt concentration (Grelet, 2014), and the self-assembly might be vital to their survival (Liu et al., 2015). Chromatin has to change its organization to achieve its genetic activity (Leforestier et al., 2008), and, although many aspects of their organization are still obscure, their building blocks, the nucleosomes, are known to have a rich diagram of liquid crystalline and crystalline phases (Livolant et al., 2006).



(a) Hard Cylinder. A simple model for DNA duplexes.

(b) A helical array of spheric beads wrapped around a cylinder. A model for helices.

(c) Short cylinder with two patches and a helical array of spherical beads wrapped around it. A simple model for nucleosomes.

Figure 6.1: Illustration of some of the models developed.

We aim at providing a set of models to be applied individually, or combined, to reproduce specific types of liquid crystalline structure in different systems. Spherocylinders have been widely used to simulate liquid crystalline phases (Avendaño et al., 2009), and a complete phase diagram of hard spherocylinders is available in the literature (Bolhuis and Frenkel, 1997). Nevertheless, some viruses (Grelet, 2014), DNA duplexes, and nucleosomes have geometries much closer to cylinders instead (Nakata et al., 2007; Leforestier et al., 2008; De Michele et al., 2012; Grelet, 2014; Wensink, 2014).

Therefore, we have chosen the hard cylinder model to underpin the development of all the other models, having the results from chapter 5 as a benchmark. Then, we decorate the cylinders to mimic specific features of biological systems: we add attractive patches to promote self-assembly behavior, a helical array of spherical beads to mimic repulsion between helices and add chirality, and a cylindrical electrostatic potential to effectively account for salt concentration (Figure 6.1).

6.2 Hard Cylinders decorated with Patches

Once that the code for the hard cylinders is implemented, the addition of the patches is straightforward. We have added a patch on the bottom and on the top of each cylinder to investigate the self-assembly of the system. The attractive part of the isotropic square-well potential was applied to model the interaction between two patches.

6.3 Hard Cylinders decorated with a helical array of beads

Many studies of liquid crystalline phases of hard helices are present in the literature (Frezza et al., 2011, 2013; Kolli et al., 2016). Wensink (2014) investigated the chirality of liquid crystals by developing a model of a cylinder enwrapped with a helical segment potential. Our model is a sort of a combination between the hard helices and the Wensink's model, following the author's suggestion to take into account the helical backbone steric interactions.

6.3.1 Building the particle

Calculation of helix parameters

The input variables are:

- The cylinder diameter D and length L.
- The beads diameter d_b.
- The percentage of fusion between the beads f.
- The number of pitches n_p (which has to be an integer to ensure that the helix is symmetric). A pitch is the height of one complete helix turn.

The radius of the helix is equal to $r_h = D/2 + d_b/2$ and the pitch is $l_p = l/n_p$. An illustration of a helix with three pitches is provided in Figure 6.3.1.

The total length of the helix l_h, and the number of beads n_b are calculated taking into consideration the fusion f, diameter of the beads d_b, helix radius r_h, and pitch l_p. The length $l_h = L_h/n_p$ of the same helix with only one pitch is obtained from Equation 6.1 (Figure 6.3.1).

$$l_h = \sqrt{4\pi^2 r_h^2 + l_p^2} \tag{6.1}$$

Taking into consideration the percentage of the fusion between the beads, the total length L_h can be written as (figure 6.3.1):

$$L_h = n_p l_h = (n_b - 1)(1 - f)d_b \tag{6.2}$$

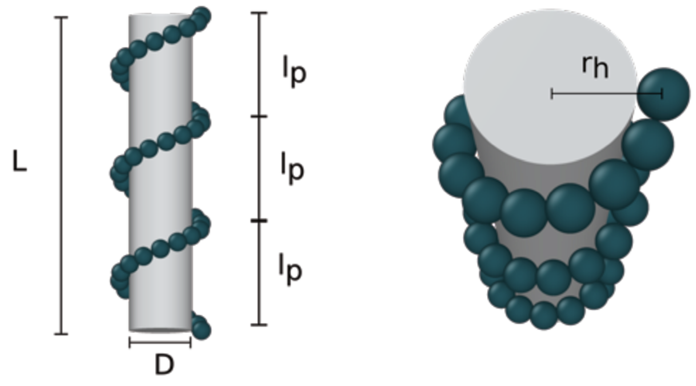


Figure 6.2: A helix wrapped around a cylinder with diameter D and length L. l_p and r_h are the pitch and radius of the helix, respectively.

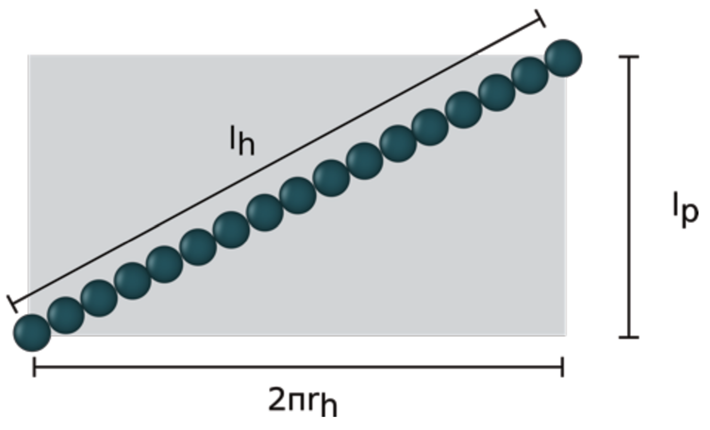


Figure 6.3: An illustration of an “unfolded” helix with one pitch.

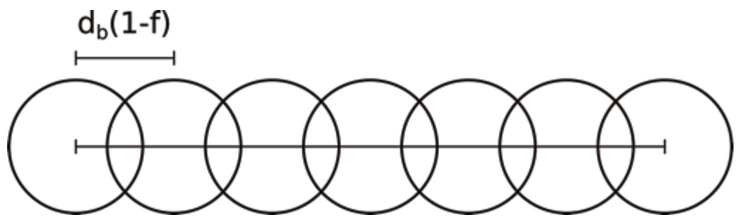


Figure 6.4: Illustration of the total length L_h of a helix considering that the beads are partially fused. d_b is the diameter of the beads and f is the percentage of fusion between them.

Equating the expressions in Eqs 6.1 and 6.2, an expression for the total number of beads n_b emerges:

$$n_b = \frac{n_p\sqrt{l_p^2 + 4\pi^2 r_h^2}}{d_b(1-f)} + 1 \tag{6.3}$$

We then approximate n_b to an integer and then calculate the f again, therefore, the input value f is only an approximate value.

Positions of each bead

The parametric equation of a helix is:

$$\begin{aligned}
x &= r_h \cos(\theta) \\
y &= r_h \sin(\theta) \\
z &= \frac{l_p}{2\pi}\theta \\
0 &\leq \theta \leq 2\pi
\end{aligned} \tag{6.4}$$

The angular increment of each bead is:

$$\theta = \frac{2\pi n_p}{(n_b - 1)} \tag{6.5}$$

Recalling the axis system fixed on the cylinder, w, v and u, the beads position are calculated as demostrated in the Fortran 90 pseudo-code below:

```fortran
do i =1,nb
       rbead(:,i) = rcylinder(:) - L/2*ez(:) & ! Bottom of the cylinder
                & + rh*ex(:)*cos(theta*dble(i - 1)) &
                & + rh*ey(:)*sin(theta*dble(i-1)) &
                & + lp*ez*theta*(i-1)/2/pi
end do
```

where the variables **rbead** and **rcylinder** store the x, y, and z positions of each bead and cylinder, respectively, and ex, ey and ez correspond to the axis w, v and u.

6.3.2 Checking the overlap

To check if there is an overlap between two particles, we proceed with the following tests:

1. Check if there is an overlap between two cylinders (section 5.3).
2. Check if the beads on each cylinder overlap with the beads on the other cylinder (the distance between two beads is less than the beads diameter).
3. Check if the cylinders overlap with the beads (the two conditions in Equation 6.6 has to be satisfied for an overlap to occur):

$$|\vec{r}_{bc}^{\,||}| < \frac{(L + d_b)}{2} \tag{6.6a}$$

$$|\vec{r}_{bc}^{\,\perp}| < \frac{(D + d_b)}{2} \tag{6.6b}$$

where $\vec{r}_{bc}^{\,||}$ and $\vec{r}_{bc}^{\,\perp}$ are the parallel and perpendicular components of the vector $\vec{r}_{bc} = \vec{r}_{bc}^{\,||} + \vec{r}_{bc}^{\,\perp}$ that joins the bead and cylinder center of mass.

6.4 Modified DLVO Potential for Cylinders

Charged groups on colloidal particles surfaces tend to dissociate into the dispersion medium, forming counterions (microions). As a consequence, the mesoscopic particles accumulate charges of the opposite sign at their surface, becoming electrically charged entities called macroions.

Even though the microions are strongly attracted to the opposite sign charges on the colloidal particles, they are not adsorbed on their surfaces, since the attraction is counteracted by the thermal motion of these ions. Therefore, at equilibrium, there will be a layer of microions surrounding the layer of opposite sign charges on the colloids surfaces, forming the electric double layer (Figure 6.5). The layer screening the macroion charges weakens the repulsion between the colloidal particles. In addition to that, due to thermal motion, the electric charge is carried by dissolved ions and extends over certain distance into the liquid phase (Verwey, 1947). To simplify the model, this charge is approximated by regarding it as a continuous space charge, and the layers are taken as a homogeneous surface charge.

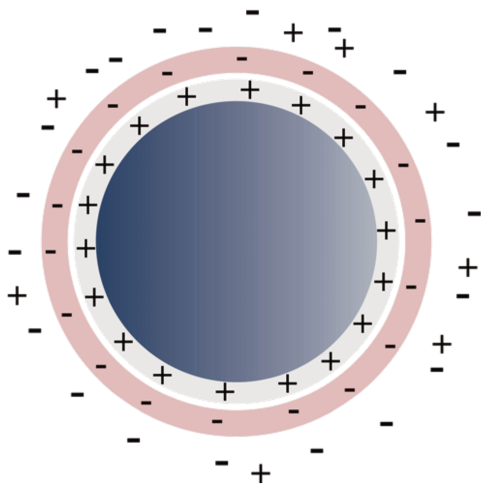


Figure 6.5: Illustration of an electric double layer on a spherical colloidal particle (blue).

As a mean-field theory, the **Poisson's equation** provides an expression for the electrostatic potential ϕ that each ion sees as a function of the charge density profile $\rho_e(r)$, assuming that (Andelman, 2006):

- The solution is modeled as a continuous media with dielectric constant ϵ_e.
- Only Coulombic interactions between charged bodies are considered.
- Induced and permanent dipole-dipole interactions are ignored.
- Charges are taken as point-like objects.
- ϕ is a continuous function that depends in a mean-field way of all other ions.

The Poisson's equation can be written as:

$$\nabla^2\phi(r) = -\frac{4\pi}{\epsilon_e}\rho_e(r) = -\frac{4\pi e}{\epsilon}(Z_+ n_+(r) + Z_- n_+(r)) \tag{6.7}$$

where e is the electron charge, Z_+ and Z_- are the valency of the cations and an anions, and $n_+(r)$ and $n_-(r)$ are the number density per unit volume.

Assuming that that the ions have a Boltzmann distribution at thermodynamic equilibrium:

$$n_\pm = n_\pm^0 \exp\left(-eZ_\pm \beta\phi\right) \tag{6.8}$$

where n^0 is the equilibrium distribution when $\phi = 0$.

By applying Equation 6.8 in Equation 6.7, the **Poisson-Boltzmann equation** is derived:

$$\nabla^2\phi(r) = \frac{4\pi e}{\epsilon_e}(Z_+ \exp(-eZ_+\beta\phi(r)) + Z_- \exp(-eZ_-\beta\phi(r))) \tag{6.9}$$

For added salt with symmetric monovalent ions (*e. g.*, NaCl, where $Z^{\pm} = \pm 1$ and $n^0_+ = n^0_- = n^0$), Equation 6.10 is simplified to:

$$\nabla^2\phi(r) = \frac{4\pi e^2}{\epsilon_e} n^0(\exp(-e\beta\phi(r)) - \exp(e\beta\phi(r))) \tag{6.10}$$

The **Debye-Hückel's theory** consists of linearizing Equation 6.10, which is valid for low electrostatic potentials. Truncating the Maclaurin series of the exponentials of Equation 6.10 at the first order term, the linearized equation becomes:

$$\nabla^2\phi(r) = -\frac{8\pi e^2}{\epsilon_e} n^0\beta\phi(r) = \lambda_D^{-2}\phi \tag{6.11}$$

where λ_D is the Debye-Hückel screening length, defined as:

$$\lambda_D = \frac{1}{k_D} = \sqrt{\frac{\epsilon_e k_B T}{8\pi e^2 n^0}} \tag{6.12}$$

where k_D is defined as the inverse of the Debye-Hückel length. The differential Equation 6.11 can be solved in spherical coordinates (for a spherical particle) by applying two boundary conditions:

1. The electric field and potential vanishes at infinity: $\phi = 0$ and $\frac{\mathrm{d}\phi}{\mathrm{d}r} = 0$ for $r \to +\infty$.
2. The electrostatic potential is assumed to be constant at the surface of the particles: $\left.\frac{\mathrm{d}\phi}{\mathrm{d}r}\right|_{r=\sigma} = -\frac{Ze}{\epsilon_e\sigma^2}$

where σ is the diameter of the sphere, and Z is the bare charge of the particle.

Solving the differential equation:

$$\phi(r) = \frac{Ze}{\epsilon_e r}\frac{\exp(-k_D(r-\sigma))}{(1+k_D\sigma)} \tag{6.13}$$

The expression in Equation 6.13 is for the electrostatic potential. The potential energy between two particles seeing one another is given by Equation 6.14.

$$\phi^Y(r) = \frac{Ze}{1+k_D\sigma}\phi(r) = \frac{Z^2e^2}{\epsilon(1+k_D\sigma)^2}\frac{\exp\left(-k_D(r-\sigma)\right)}{r} \tag{6.14}$$

To model colloidal particles in solutions containing counterions neutralizing the ions on the surface of the particles, and also microions originated from the addition of electrolytes, the DLVO (Derjaguin-Landau-Vervey-Overbeek) model (Verwey and Overbeek, 1948) is an usual strategy. The idea that grounds the model is that the van der Waals attraction is balanced with the screened electrostatic repulsion in Equation 6.14. Accordingly, the solvent and microions are accounted for as an effective interaction between the charged particles. In the spirit of the DLVO model, Giacometti et al. (2005) described the interaction between the monomers and dimers of a charged globular protein β-lactoglobulin in solution with the following potential:

$$u(r) = u^{\text{HS}}(r) + \phi^Y(r) \tag{6.15}$$

where $u(r)$ is the total intermolecular potential between two particles and u^{HS} is the hard-sphere potential.

In a similar fashion, we shall model the interactions between rod-like particles in solution as:

$$u(\vec{r}, \hat{\Omega}_1, \hat{\Omega}_2) = u^{\text{HC}}(\vec{r}, \hat{\Omega}_1, \hat{\Omega}_2) + \phi^C(\vec{r}, \hat{\Omega}_1, \hat{\Omega}_2) \tag{6.16}$$

where u^{HC} is the hard-cylinder potential, which in the code consists of merely checking the overlap, as presented in section 5.3), and ϕ^C is a modified electrostatic potential for cylinders.

6.4.1 Modified electrostatic potential for cylinders

We propose a simplified model for the electrostatic potential between charged cylinders:

$$\phi^C(\vec{r}, \hat{\Omega}_1, \hat{\Omega}_2) = \frac{Z^2e^2}{\epsilon_e(1+k_D\sigma_{sh}(\vec{r}, \hat{\Omega}_1, \hat{\Omega}_2))^2}\frac{\exp[-k_D(|\vec{r}| - \sigma_{sh}(\vec{r}, \hat{\Omega}_1, \hat{\Omega}_2))]}{|\vec{r}|} \tag{6.17}$$

where $\vec{r}$ is the vector joining the particles centers of mass and $\sigma_{sh}(\vec{r}, \hat{\Omega}_1, \hat{\Omega}_2)$ is the contact distance between the two cylindrical shells along $\vec{r}$. A more rigorous approach would be to take $\sigma_{sh}(\vec{r}, \hat{\Omega}_1, \hat{\Omega}_2)$ as the contact distance of the cylinders along the vector of closest approach between the two particles. Nonetheless, since the contact will be precluded by the cylinder overlap algorithm, we apply it as the contact distance along $\vec{r}$, for the sake of simplicity, as displayed in Equation 6.18.

$$\sigma_{sh}(\vec{r}, \hat{\Omega}_1, \hat{\Omega}_2) = x_1(\vec{r}, \hat{\Omega}_1) + x_2(\vec{r}, \hat{\Omega}_2) \tag{6.18}$$

where x_1 and x_2 are the distances between the center of mass and shell of each cylinder. $\vec{r}_{12}$ goes through either the rim or the disk of each cylinder. For instance, in Figure 6.6, $\vec{r}_{12}$ goes through the disk of cylinder 1 and through the rim of cylinder 2.

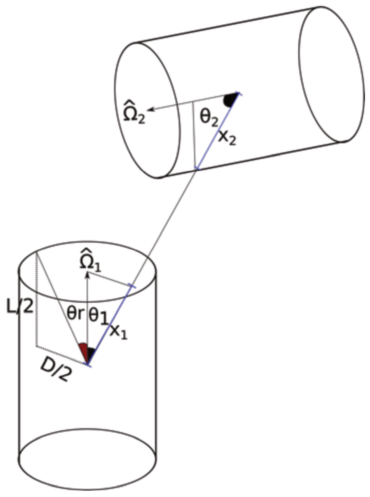


Figure 6.6:

If $\vec{r}_{12}$ goes through the disk of the cylinder, x is calculated as:

$$x = \frac{L}{2} \cdot \frac{1}{|\cos\theta|} = \frac{L}{2} \cdot \frac{1}{|\cos{(\hat{r} \cdot \hat{\Omega})}|} \tag{6.19}$$

On the other hand, if $\vec{r}_{12}$ goes through the rim, x is obtained from:

$$x = \frac{D}{2} \cdot \frac{1}{\sin\theta} \tag{6.20}$$

where θ is the angle between the orientation $\hat{\Omega}$ and $\vec{r}_{12}$, and $\hat{r}$ is the unit vector along $\vec{r}$.

Therefore, before calculating x, we have to check which is the case for each cylinder.

We define a limiting angle θ_r in such a way that if $|\cos\theta| < \cos\theta_r$, the vector goes through the rim and x is calculated with Equation 6.20. Otherwise, it goes through the disk and x is obtained from Equation 6.19. The cosine of the limiting angle is obtained from Equation 6.21.

$$\cos\theta_r = \frac{L}{\sqrt{D^2 + L^2}} \tag{6.21}$$

We have carried out the simulations with a slightly simpler model resembling an Yukawa potential, in which the potential strength is independent of σ_{sh}:

$$\phi^{Y*}(\vec{r}, \hat{\Omega}_1, \hat{\Omega}_2) = \frac{\exp[-k_D(|\vec{r}| - \sigma_{sh}(\vec{r}, \hat{\Omega}_1, \hat{\Omega}_2))]}{|\vec{r}|/D} \tag{6.22}$$

where $\phi^{Y*} = \phi^{\mathrm{Y}}/\varepsilon$, where ε is the potential strength.

6.5 Simulation Details

In this chapter, the number of cycles used is about twice the total number of particles. In each cycle, either the position and orientation of a particle or the total volume is chosen randomly to be changed. For this purpose, a random number between $1 \leq \zeta \leq N + 1$ is generated, if $\zeta \leq N$ a particle is moved, otherwise, the volume is changed. All the codes are provided in the appendix. There is a main code for each system and the subroutines that are common for multiple systems are provided in three different modules:

- MC NPT for Hard Cylinders + patches (D.1.1)
- MC NPT for Hard Cylinders + helical array of beads (D.1.2)
- MC NPT for Hard Cylinders + cylindrical Yukawa (D.1.3)
- Module to set global variables (C.2.2)
- Module to create initial configurations (C.2.3)
- Module with the main subroutines (including the overlap checks and calculation of the potentials) (C.2.4)

We used the OVITO visualization tool (Stukowski, 2010) to make the snapshots of the simulations. We added a python script to add color the particles according to the angle between its orientation and the phase director. The scripts are provided in the appendix C.2.5.

6.6 Results

6.6.1 Hard Cylinders with attractive patches

We have simulated hard cylinders with aspect ratio $L^* = 2$ and two attractive patches: one on the top and one at the bottom of each cylinder. To avoid kinetic trapping, we have started the simulation at $T^* = 1$ and then we have frozen the system until $T^* = 0.2$ ($T^* = 1$, 0.8, 0.6, 0.4 and 0.2), at the same pressure $p^* = 0.31$. The last configuration of a higher temperature is set as the initial configuration of a lower one. The range of the patches square-well potential was set to $\lambda = 1.5$.

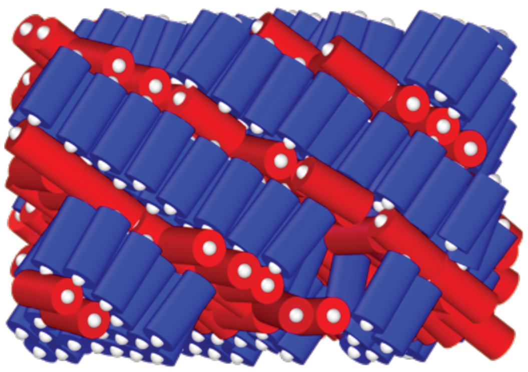
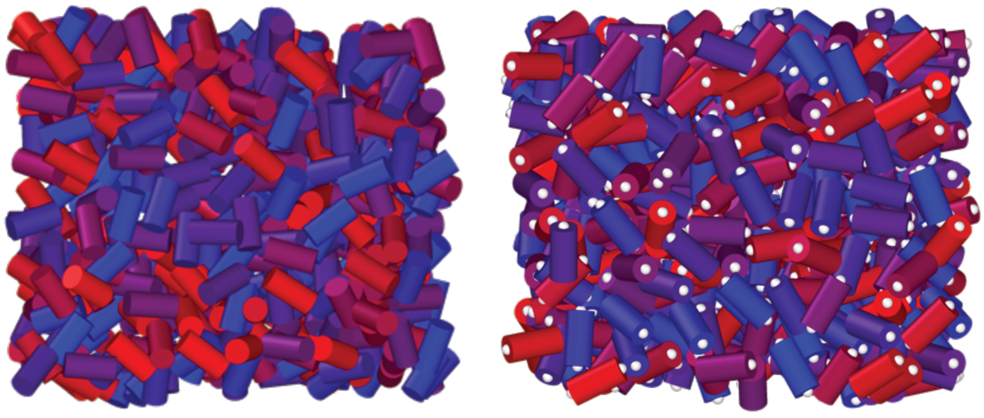


Figure 6.7: Snapshot of the simulation of hard cylinders with patches at the top and bottom. $p^* = 0.31$, $T^* = 0.2$ and $L^* = 2$. The coloring scale is such that the particle is blue if its orientation is aligned with the phase director $\vec{n}$, red if it is perpendicular to $\vec{n}$ and a combination of both colors depending on the angle between the orientation and director.

The particles self-assembled in directions parallel and perpendicular to the phase director (Figure 6.7). Although this phase is, as far as we know, incomparable to any real phases, it is interesting to notice that the patches location and/or potential range can be changed to promote some specific phase behavior. For instance, with a smaller range, Nguyen et al. (2014) observed the stacking of cylinders into longer units, reproducing the behaviour of short DNA duplexes observed by Nakata et al. (2007).

This elongation is insufficient for the cylinders to form organized phases only with steric interactions, as well-known from Onsager's theories (Onsager, 1949). On chapter 5 we have shown that at this aspect ratio no organized phase should be found. This is verified by a snapshot of a simulation without the patches (Figure 6.8a). At a higher temperature, the attractive interactions are less significant, and the fluid also remains in the isotropic phase (Figure 6.8b).



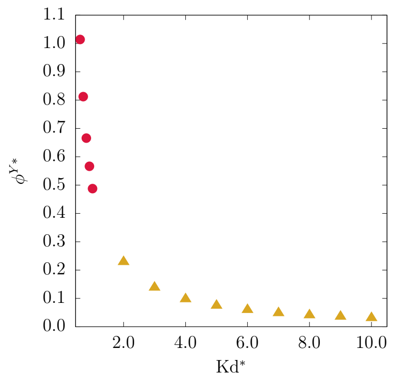
(a) Hard Cylinders without patches. $p^* = 0.31$ (b) Hard Cylinders with patches. $p^* = 0.31$, $T^* = 1.0$

Figure 6.8: Snapshot of the simulation of hard cylinders with and without patches at $p^* = 0.31$ and $L^* = 2$. The color scale is the same as in Figure 6.7.

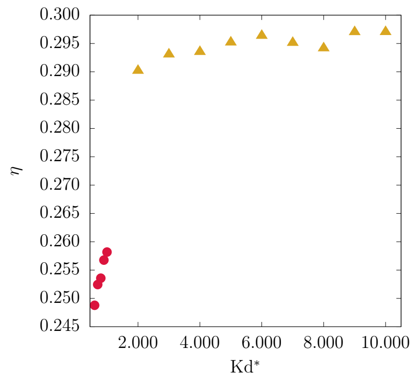
6.6.2 Hard Cylinders + cylindrical Yukawa

Simulations of hard cylinders with an anisotropic repulsive Yukawa potential (Equation 6.22) were carried out for cylinders with aspect ratio of $L^* = 10$, at $T^* = 1.0$, and p^* =2.36 and 6.28. The pressure and temperature were chosen to be equivalent to the condition at which a system of hard cylinders with this aspect ratio exhibits a nematic/smectic organization (chapter 5).

In this case, k_D is the screening parameter, and sets the range of the repulsion. Figure 6.9 and 6.10 show that the total intermolecular potential is larger for lower k_D, *i. e.*, when the screening is weak, the repulsion forces prevail, and the system occupies a larger volume. On the other hand, as the screening becomes stronger, the range of the repulsion decreases, and the system approaches the hard cylinder case (limit of $k_D \to \infty$ and zero intermolecular potential). The screening parameter also plays an important role in the phase organization. As k_D increases and the systems approach the hard cylinder case, a nematic phase is formed at $p^* = 0.3$. Conversely, for lower k_D, the system remains at the

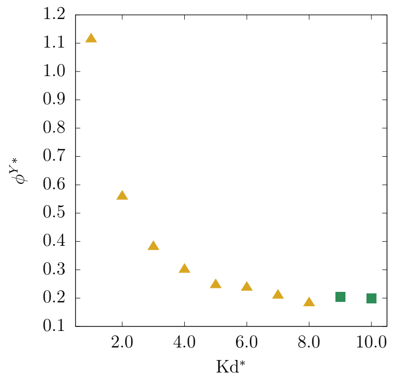


(a) Reduced intermolecular potential as a function of k_D^*.

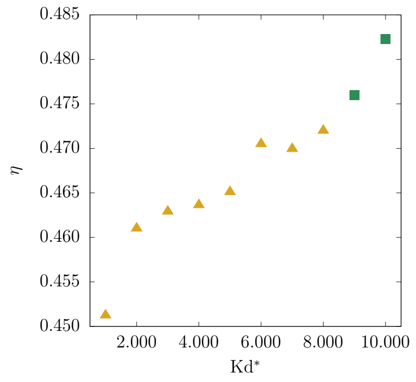


(b) Packing fraction as a function of k_D^*.

Figure 6.9: $p^* = 0.3$. Red circles are in the isotropic phase and yellow triangles in the nematic.



(a) Reduced intermolecular potential as a function of k_D^*.



(b) Packing fraction as a function of k_D^*.

Figure 6.10: $p^* = 0.8$. Yellow triangles in the nematic and green squares in the smectic.

isotropic phase, since by occupying a larger volume, the particles have enough space to maximize the orientational entropy.

Although at $p^* = 6.28$, the nematic order parameter approaches 1 for every value of k_D, a snapshot of the simulations reveals the formation of a smectic phase at higher k_Ds (Figure 6.11). It is worth noticing, however, that we have not used the floppy-box Monte Carlo to simulate these cases, hence, what seems like a smectic C phase is probably actually a smectic A. Since, as discussed on chapter 5, the pressure of a smectic is anisotropic Bolhuis and Frenkel (1997) and the floppy-box should be used to obtain an isotropic pressure.

Although the potential is purely repulsive, values above one of the radial distribution function and the negative values of the potential of mean force (Figure 6.12) reveal an effective attraction, probably due to the many-body interactions, which are more signifi-

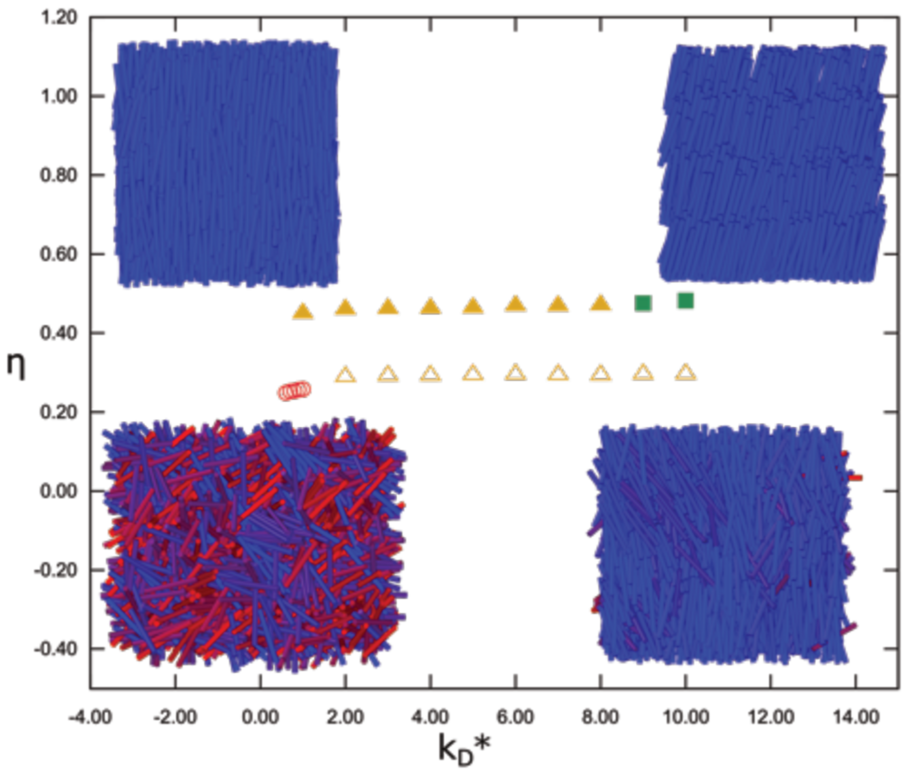
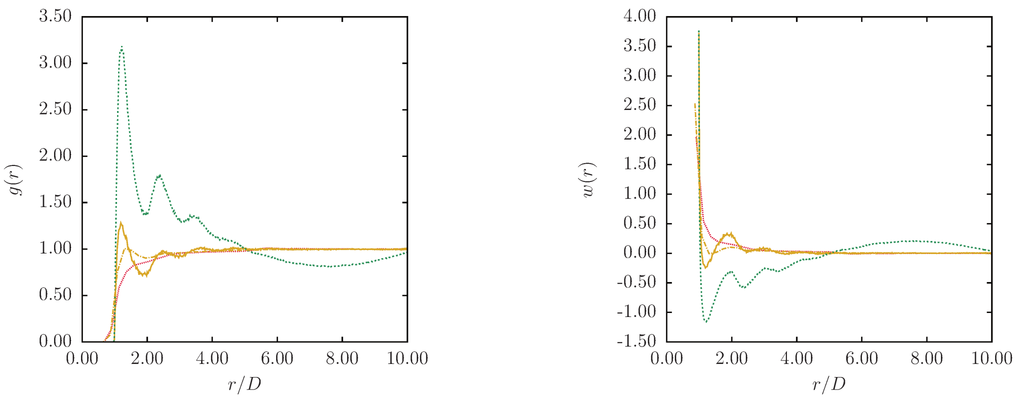


Figure 6.11: Comparison between $p^* = 2.36$ (open symbols) and $p^* = 6.28$ (closed symbols). Red circles: isotropic, yellow triangles: nematic, green squares: smectic.

cant at higher densities. Franco et al. (2015) showed that the many-body interactions are somehow attractive when compared to the two-body interactions.



(a) Radial Distribution function

(b) Potential of mean force

Figure 6.12: Dotted line: $p^* = 2.36$ and $kd^* = 1.0$, dash-dotted line: $p^* = 2.36$ and $kd^* = 10.0$, continuous line: $p^* = 6.28$ and $kd^* = 1.0$, dashed line: $p^* = 6.28$ and $kd^* = 10.0$

6.7 Conclusions

We have introduced three models to study different features in biological systems: attractive patches to mimic hydrophobicity and promote self-assembly, a helical array of hard beads to mimic helices repulsion and chirality, and a cylindrical Yukawa-like potential to account effectively for salt molality. Despite its exploratory nature, since only a limited number of tests have been carried out, this study offers some insight into how each model could be used to describe specific liquid crystalline phases. A systematic study of each model at different conditions could be usefully explored in further research, as they might be a good strategy to investigate liquid crystalline formation in biological systems.

"It is disastrous when instead of merely attending to a rose we are forced to think of ourselves looking at the rose, with a certain type of mind and a certain type of eyes. It is disastrous because, if you are not very careful, the color of the rose gets attributed to our optic nerves and its scent to our noses, and in the end there is no rose left."

—Clive Staples Lewis

7 Conclusions and suggestions for future works

7.1 Conclusions

The main goal of this work was to develop theoretical and numerical models for anisotropic particles as a strategy to capture the thermodynamic behavior of nonspherical molecules. The properties prediction of industrial relevant fluids is improved solely by modeling the molecule as a single nonspherical segment rather than a set of spherical segments; diverse liquid crystalline and self-assembly phases were obtained by applying cylindrical models in Monte Carlo molecular simulations, which can be useful to study biological systems. The general conclusion is that modeling molecules and nanounits as nonspherical particles to characterize their geometry might improve the description of thermodynamic systems, as it seems to provide more physically meaningful models.

Based on Barker-Henderson second order perturbation theory, a new approach to the development of equations of state for linear molecules and small chains was introduced, in which the Hard Gaussian Overlap model (HGO) is used as the reference fluid and the attractive part of the spherical square-well potential as the perturbation. In comparison to the SAFT-VR SW, in which the reference potential is the hard-sphere, our proposed approach predicts more accurately critical properties and generally provides better supercritical and derivative properties estimates for the fluids tested. In addition to that, the use of a single ellipsoidal segment described by the HGO potential eradicates two issues with the chain contribution calculations in the SAFT-VR SW original approach: the commonly fitted non-integer number of segments (which weakens the physical meaning) and the negative value on the zero density limit.

The trends of the isochoric heat capacity are not captured neither by SAFT-VR SW nor by the EoS proposed in this work. Therefore, we compared the predictions of the EoSs also with SAFT-VR Mie and two different coarse-grained force fields (SAFT-γ Mie and HGO + SW). The comparison of theoretical and Monte Carlo simulations results was revealing in several ways. First, it showed that the choice of a hard repulsive potential causes a significant loss of accuracy at higher densities. Furthermore, theoretical approximations were also found to have a substantial influence on the predictions: the truncation of the temperature expansion on the perturbed potential affects the predictions at low temperatures, and at high densities the macroscopic compressibility approximation applied in the calculation of the second-order perturbation term is inadequate to describe the behavior of the property.

A similar approach to the one introduced on the first chapter was applied to develop equations of state for spherocylindrical and cylindrical particles as well. The equations of state for ellipsoids, cylinders and spherocylinders were used to calculate the vapor-liquid equilibrium of several components, including longer chains and disk-like molecules. In general, the proposed equations of state have a better performance than SAFT-VR SW, raising the possibility that, at least for the molecules studied in this work, the chain contribution could be completely eliminated provided that the entire molecule is modeled as a nonspherical segment. Furthermore, the results showed that the overprediction of the critical point might be related to the fact that the reference potential is independent of the temperature.

Although studies on the liquid crystalline phases of ellipsoids and spherocylinders are abundant in the literature, much less was known about the hard cylinder. Therefore, we provided a general mapping of the phase boundaries of both rod and disk-like hard cylinders. Hard cylinders with any aspect ratio exhibit an isotropic phase at lower densities. The nematic phase was observed in sufficient anisotropic disks ($L/D <<$) and rods ($L/D >>$). Smectic A and crystalline phases formed at higher packing fractions in rod fluids, while cubatic and columnar phases formed in disk-like fluids.

Even though the numerical model for cylinders is considerably more complex to implement than the Hard Gaussian Overlap or the one for spherocylinders, it might be better suited to describe the geometry of some molecules and nanounits. Diverse self-assembly phases are obtained by decorating the cylinders with attractive patches that can mimic, for instance, terminal DNA bases hydrophobicity. An effective electrostatic potential was added in the fluid of cylinders and different phases were obtained solely by changing the screening parameter. The proposed model might be a good strategy to account for the effects of salt concentration on the liquid crystalline phase formation of biological nano-objects such as nucleosomes.

7.2 Suggestions for future works

We hope to provide with this work 'a small but genuine' contribution, and, since there is still 'plenty that needs to be done', we propose some suggestions for further research.

With regards to the equation of state proposed in this work, some extensions and modifications should be helpful to continue the work initiated:

- Extend the EoS for mixtures
- Apply an anisotropic potential as the perturbation.
- Use an effective diameter dependent on temperature to develop an equation of state for ellipsoids with a thermal reference potential mapped as a hard-sphere potential with an effective packing fraction.
- Include higher order perturbation terms in the development of the equation of state.

NPT Monte Carlo simulations of the HGO + SW potential might shed some light on the effects of the theoretical approximations and choice of the intermolecular potential also on other properties such as pressure, isobaric specific heat, isothermal compressibility and thermal expansion coefficient. The application of Monte Carlo optimization methods on the simulations proposed on Chapter 6 would be extremely helpful to to speed up equilibration. In addition to that, using the algorithm to calculate the shortest distance between two segments might be sufficient to know *a priori* the kind of overlap between two cylinders, which could speed up the overlap check.

References

Charlles R.A. Abreu, Frederico W. Tavares, and Marcelo Castier. Influence of particle shape on the packing and on the segregation of spherocylinders via Monte Carlo simulations. *Powder Technol.*, 134:167–180, 2003.

C. G. Aimoli, E. J. Maginn, and C. R. A. Abreu. Force field comparison and thermodynamic property calculation of supercritical CO_2 and CH_4 using molecular dynamics simulations. *Fluid Phase Equilibr.*, 368:80–90, 2014.

B. J. Alder, D. A. Young, and M. A. Mark. Studies in Molecular Dynamics. X. Corrections to the Augmented van der Waals Theory for the Square Well Fluid. *J. Chem. Phys.*, 56:3013–3029, 1972.

M. P. Allen and D. J. Tildesley. *Computer Simulation of Liquids*. Oxford University Press, 2 nd edition, 2017.

M. P. Allen, G. T. Evans, D. Frenkel, and B. M. Mulder. *Hard Convex Body Fluids*, pages 1–166. John Wiley & Sons, Ltd, 2007.

D. Andelman. Introduction to electrostatics in soft and biological matter. In *Soft Condens. Matter Phys. Mol. Cell Biol.*, pages 97–122. 2006.

I. S. Araújo and L. F. M. Franco. A model to predict adsorption of mixtures coupled with SAFT-VR Mie equation of state. *Fluid Phase Equilibr.*, 496:61–68, 2019.

T. Aslyamov, V. Pletneva, and A. Khlyupin. Random surface statistical associating fluid theory: adsorption of n-alkanes on rough surface. *J. Chem. Phys.*, 150:054703, 2019.

C. Avendaño, A. Gil-Villegas, and E. González-Tovar. A Monte Carlo simulation study of binary mixtures of charged hard spherocylinders and charged hard spheres. *Chem. Phys. Lett.*, 470:67–71, 2009.

C. Avendaño, T. Lafitte, A. Galindo, C. S. Adjiman, G. Jackson, and E. A. Müller. SAFT-γ force field for the simulation of molecular fluids. 1. A single-site coarse grained model of carbon dioxide. *J. Phys. Chem. B*, 115:11154–11169, 2011.

J. A. Barker and D. Henderson. Perturbation theory and equation of state for fluids: The square-well potential. *J. Chem. Phys.*, 47:2856–2861, 1967.

B. J. Berne and P. Pechukas. Gaussian Model Potentials for Molecular Interactions. *J. Chem. Phys.*, 56:4213–4216, 1972.

Ronald Blaak, Daan Frenkel, and Bela M. Mulder. Do cylinders exhibit a cubatic phase? *J. Chem. Phys.*, 110:11652–11659, 1999.

F. J. Blas and L. F. Vega. Thermodynamic behaviour of homonuclear and heteronuclear Lennard-Jones chains with association sites from simulation and theory. *Mol. Phys.*, 92:135–150, 1997.

P. Bolhuis and D. Frenkel. Tracing the phase boundaries of hard spherocylinders. *J. Chem. Phys.*, 106:666–687, 1997.

O. L. Boshkova and U. K. Deiters. Soft repulsion and the behavior of equations of state at high pressures. *Int. J. Thermophys.*, 31:227–252, 2010.

Tomas Boublik. Statistical thermodynamics of convex molecule fluids. *Mol. Phys.*, 27: 1415–1427, 1974.

E. H. Brown. On the thermodynamic properties of fluids. *Bull. Inst. Int. Froid*, 1:169, 1960.

M. Bzówka, K. Mitusińska, A. Raczyńska, A. Samol, J. A. Tuszyński, and A. Góra. Structural and Evolutionary Analysis Indicate That the SARS-CoV-2 Mpro Is a Challenging Target for Small-Molecule Inhibitor Design. *Int. J. Mol. Sci.*, 21:3099, 2020.

L. F. Cameretti, G. Sadowski, and J. M. Mollerup. Modeling aqueous electrolyte solutions with perturbed-chain statistical associated fluid theory. *Ind. Eng. Chem. Res.*, 44:3355–3362, 2005.

W. A. Cañas-Marín, B. A. Hoyos, and D. L. González. Role of the soft-core repulsion upon the prediction of characteristic curves with PC-SAFT. *Fluid Phase Equilibr.*, 499: 112247, 2019.

N. F. Carnahan and K. E. Starling. Equation of State for Nonattracting Rigid Spheres. *J. Chem. Phys.*, 51:635–636, 1969.

W. G. Chapman, K. E. Gubbins, and G. Jackson. SAFT: Equation-of-State Solution Model for Associating Fluids. *Fluid Phase Equilibr.*, 52:31–38, 1989.

J. Chen and J.-G. Mi. Equation of state extended from SAFT with improved results for non-polar fluids across the critical point. *Fluid Phase Equilibr.*, 186:165–184, 2001.

Stephen S Chen and Aleksander Kreglewski. Applications of the Augmented van der Waals Theory of Fluids.: I. Pure Fluids. *Berichte der Bunsengesellschaft für Phys. Chemie*, 81:1048–1052, 1977.

Maryam Dargahi and Elham Jafari. Prediction of the critical properties of n-alkanes and their mixtures with two versions of SAFT equation of state. *J. Iran. Chem. Soc.*, 12: 1493–1500, 2015.

G. Das, S. Hlushak, M. C. Ramos, and C. McCabe. Predicting the thermodynamic properties and dieletric behavior of electrolyte solutions using the SAFT-VR+DE equation of state. *AIChE J.*, 61:3053–3072, 2015.

C. De Michele. Theory of self-assembly-driven nematic liquid crystals revised. *Liq. Cryst.*, 46:2003–2012, 2019.

C. De Michele, T. Bellini, and F. Sciortino. Self-assembly of bifunctional patchy particles with anisotropic shape into polymers chains: Theory, simulations, and experiments. *Macromolecules*, 45:1090–1106, 2012.

E. De Miguel and E. M. Del Río. The isotropic-nematic transition in hard Gaussian overlap fluids. *J. Chem. Phys.*, 115:9072–9083, 2001.

S. Dufal, T. Lafitte, A. J. Haslam, A. Galindo, G. N. I. Clark, C. Vega, and G. Jackson. The A in SAFT: developing the contribution of association to the helmholtz free energy within a Wertheim TPT1 treatment of generic Mie fluids. *Mol. Phys.*, 113:948–984, 2015.

Peter D. Duncan, Matthew Dennison, Andrew J. Masters, and Mark R. Wilson. Theory and computer simulation for the cubatic phase of cut spheres. *Phys. Rev. E - Stat. Nonlinear, Soft Matter Phys.*, 79:1–11, 2009.

Simone Dussi, Massimiliano Chiappini, and Marjolein Dijkstra. On the stability and finite-size effects of a columnar phase in single-component systems of hard-rod-like particles. *Mol. Phys.*, 116:2792–2805, 2018.

I. G. Economou. Statistical Associating Fluid Theory: a successful model for the calculation of thermodynamic and phase equilibrium properties of complex fluid mixtures. *Ind. Eng. Chem. Res.*, 41:953–962, 2002.

R. Eppenga and D. Frenkel. Monte Carlo study of the isotropic and nematic phases of infinitely thin hard platelets. *Mol. Phys.*, 52:1303–1334, 1984.

D. K. Eriksen, G. Lazarou, A. Galindo, G. Jackson, C. S. Adjiman, and A. J. Haslam. Development of intermolecular potential models for electrolyte solutions using an electrolyte SAFT-VR Mie equation of state. *AIChE J.*, 61:3053–3072, 2015.

R. Espíndola-Heredia, F. Del Río, and A. Malijevsky. Optimized equation of the state of the square-well fluid of variable range based on a fourth-order free-energy expansion. *J. Chem. Phys.*, 130:024509, 2009.

L. F. M. Franco, M. Castier, and I. G. Economou. Two-body perturbation theory versus first order perturbation theory: A comparison based on the square-well fluid. *J. Chem. Phys.*, 147:214108, 2017a.

L. F. M. Franco, I. G. Economou, and M. Castier. Statistical mechanical model for adsorption coupled with SAFT-VR Mie equation of state. *Langmuir*, 33:11291–11298, 2017b.

Luís Fernando Mercier Franco, Cristiano Luis Pinto de Oliveira, and Pedro de Alcântara Pessôa Filho. Thermodynamics of protein aqueous solutions: From the structure factor to the osmotic pressure. *AIChE J.*, 61(9):2871–2880, 2015.

E. Frezza, F. Tombolato, and A. Ferrarini. Right- and left-handed liquid crystal assemblies of oligonucleotides: Phase chirality as a reporter of a change in non-chiral interactions? *Soft Matter*, 7:9291–9296, 2011.

E. Frezza, A. Ferrarini, H. B. Kolli, A. Giacometti, and G. Cinacchi. The isotropic-to-nematic phase transition in hard helices: Theory and simulation. *J. Chem. Phys.*, 138: 164906, 2013.

A. Galindo and F. J. Blas. Theoretical examination of the global fluid phase behavior and critical phenomena in carbon dioxide + n-alkane binary mixtures. *J. Phys. Chem. B*, 106:4503–4515, 2002.

E. García-Sánchez, A. Martínez-Richa, J. A. Villegas-Gasca, L. H. Mendoza-Huizar, and A. Gil-Villegas. Predicting the phase diagram of a liquid crystal using the convex peg model and the semiempirical PM3 method. *J. Phys. Chem. A*, 106:10342–10349, 2002.

J. G. Gay and B. J. Berne. Modification of the overlap potential to mimic a linear site–site potential. *J. Chem. Phys.*, 74:3316–3319, 1981.

Ahmadreza F. Ghobadi and J. Richard Elliott. A simple extrapolation of thermodynamic perturbation theory to infinite order. *J. Chem. Phys.*, 143, 2015.

A. Giacometti, D. Gazzillo, G. Pastore, and T. K. Das. Numerical study of a binary Yukawa model in regimes characteristic of globular proteins in solutions. *Phys. Rev. E*, 71:1–10, 2005.

J. W. Gibbs. *Elementary principles in statistical mechanics developed with especial reference to the rational foundation of thermodynamics.* Charles Scribner's Sons, 1902.

A. Gil-Villegas and A. L. Benavides. Equilibrium of the square-well fluid. *Fluid Phase Equilibr.*, 119:97–112, 1996.

A. Gil-Villegas, A. Galindo, P. J. Whitehead, S. J. Mills, G. Jackson, and A. N. Burgess. Statistical associating fluid theory for chain molecules with attractive potentials of variable range. *J. Chem. Phys.*, 106:4168–4186, 1997.

E. Grelet. Hard-Rod Behavior in Dense Mesophases of Semiflexible and Rigid Charged Viruses. *Phys. Rev. X*, 4:021053, 2014.

J. Gross and G. Sadowski. Perturbed-Chain SAFT: an equation of state based on a perturbation theory of chain molecules. *Ind. Eng. Chem. Res.*, 40:1244–1260, 2001.

Y. Han and P. Král. Computational Design of ACE2-Based Peptide Inhibitors of SARS-CoV-2. *ACS Nano*, 14:5143–5147, 2020.

J. P. Hansen and I. R. McDonald. *Theory of simple liquids*. Academic Press, Amsterdam, 3 rd edition, 2006.

N. Ibarra-Avalos, A. Gil-Villegas, and A. Martinez Richa. Excluded volume of hard cylinders of variable aspect ratio. *Mol. Simul.*, 33:505–515, 2007.

H. B. Kolli, G. Cinacchi, A. Ferrarini, and A. Giacometti. Chiral self-assembly of helical particles. *Faraday Discuss.*, 186:171–186, 2016.

T. Lafitte, D. Bessieres, M. M. Piñeiro, and J. L. Daridon. Simultaneous estimation of phase behavior and second-derivative properties using the statistical associating fluid theory with variable range approach. *J. Chem. Phys.*, 124:024509, 2006.

T. Lafitte, M. M. Piñeiro, J. L. Daridon, and D. Bessières. A comprehensive description of chemical association effects on second derivative properties of alcohols through a SAFT-VR approach. *J. Phys. Chem. B*, 111:3447–3461, 2007.

T. Lafitte, A. Apostolakou, C. Avendaño, A. Galindo, C. S. Adjiman, E. A. Müller, and G. Jackson. Accurate statistical associating fluid theory for chain molecules formed from Mie segments. *J. Chem. Phys.*, 139:154504, 2013.

S. D. Lee. A numerical investigation of nematic ordering based on a simple hard-rod model. *J. Chem. Phys.*, 87:4972–4974, 1987.

A. Leforestier, A. Bertin, J. Dubochet, K. Richter, N. Sartori Blanc, and F. Livolant. Expression of chirality in columnar hexagonal phases or DNA and nucleosomes. *Comptes Rendus Chim.*, 11:229–244, 2008.

P. J. Linstrom and W. G. Mallard.

Liming Liu and Shaozhou Chen. Correlation of the acentric factor for hydrocarbons. *Ind. Eng. Chem. Res.*, 35:2484–2486, 1996.

S. Liu, T. Zan, S. Chen, X. Pei, H. Li, and Z. Zhang. Thermoresponsive Chiral to Nonchiral Ordering Transformation in the Nematic Liquid-Crystal Phase of Rodlike Viruses: Turning the Survival Strategy of a Virus into Valuable Material Properties. *Langmuir*, 31:6995–7005, 2015.

F. Livolant, S. Mangenot, A. Leforestier, A. Bertin, M. de Frutos, E. Raspaud, and D. Durand. Are liquid crystalline properties of nucleosomes involved in chromosome structure and dynamics? *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.*, 364:2615–2633, 2006.

F. Llovell and L. F. Vega. Prediction of thermodynamic derivative properties of pure fluids through the soft-SAFT equation of state. *J. Phys. Chem. B*, 110:11427–11437, 2006.

H. C. Longuet-Higgins. The statistical thermodynamics of multicomponent systems. *Proc. Roy. Soc. Lond.*, 205:247–269, 1951.

J. T. Lopes and L. F. M. Franco. New Thermodynamic Approach for Nonspherical Molecules Based on a Perturbation Theory for Ellipsoids. *Ind. Eng. Chem. Res.*, 58: 6850–6859, 2019.

J. T. Lopes and L. F. M. Franco. Prediction of isochoric heat capacity: Discrete versus continuous potentials. *Fluid Phase Equilibr.*, 506:112380, 2020.

A. Maghari and M. S. Sadeghi. Prediction of sound velocity and heat capacities of n-alkanes from the modified SAFT-BACK equation of state. *Fluid Phase Equilibr.*, 252: 152–161, 2007.

M. G. Martin and J. I. Siepmann. Transferable potentials for phase equilibria. 1. United-atom description of n-alkanes. *J. Phys. Chem. B*, 102:2569–2577, 1998.

Simon C. McGrother, Dave C Williamson, and George Jackson. A re-examination of the phase diagram of hard spherocylinders. *J. Chem. Phys.*, 104:6755–6771, 1996.

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *J. Chem. Phys.*, 21:1087–1092, 1953.

N. J. Mottram and Christopher J. P. Newton. Introduction to Q-tensor theory. *arXiv e-prints*, page arXiv:1409.3542, 2014.

M. Nakata, G. Zanchetta, B. D. Chapman, C. D. Jones, J. O. Cross, R. Pindak, T. Bellini, and N. A. Clark. End-to-End Stacking and Liquid Crystal Condensation of 6- to 20-Base Pair DNA Duplexes. *Science*, 318:1276–1279, 2007.

K. T. Nguyen, F. Sciortino, and C. De Michele. Self-assembly-driven nematization. *Langmuir*, 30:4814–4819, 2014.

I. K. Nikolaidis, L. F. M. Franco, L. Vechot, and I. G. Economou. Modeling of physical properties and vapor-liquid equilibrium of ethylene and ethylene mixtures with equations of state. *Fluid Phase Equilibr.*, 470:149–163, 2018.

L. Onsager. The effects of shape on the interaction of colloidal particles. *Ann. N. Y. Acad. Sci.*, 51:627–659, 1949.

A. G. Orellana, E. Romani, and C. De Michele. Speeding up Monte Carlo simulation of patchy hard cylinders. *Eur. Phys. J. E*, 41:51, 2018.

M. M. Pakulska, I. Elliott Donaghue, J. M. Obermeyer, A. Tuladhar, C. K. McLaughlin, T. N. Shendruk, and M. S. Shoichet. Encapsulation-free controlled release: Electrostatic adsorption eliminates the need for protein encapsulation in PLGA nanoparticles. *Sci. Adv.*, 2:e1600519, 2016.

J. D. Parsons. Nematic ordering in a system of rods. *Phys. Rev. A*, 19:1225–1230, 1979.

C. A. Passut and R. P. Danner. Correlation of ideal gas enthalpy, heat capacity, and entropy. *Ind. Eng. Process. Des. Develop.*, 11:543–546, 1972.

A. Perera. Fluids of hard natural and Gaussian ellipsoids: a comparative study by integral equation theories. *J. Chem. Phys.*, 129:194504, 2008.

V. F. D. Peters, M. Vis, H. H. Wensink, and R. Tuinier. Algebraic equations of state for the liquid crystalline phase behavior of hard rods. *Phys. Rev. E*, 101:062707, 2020.

Oliver Pfohl and Gerd Brunner. 2. Use of BACK to modify SAFT in order to enable density and phase equilibrium calculations connected to gas-extraction processes. *Ind. Eng. Chem. Res.*, 37:2966–2976, 1998. doi: 10.1021/ie9705259.

Kenneth S. Pitzer. The Volumetric and Thermodynamic Properties of Fluids. I. Theoretical Basis and Virial Coefficients. *J. Am. Chem. Soc.*, 77:3427–3433, 1955.

J. J. Potoff and J. I. Siepmann. Vapor-liquid equilibria of mixtures containing alkanes, carbon dioxide, and nitrogen. *AIChE J.*, 47:1676–1682, 2001.

M. S. Santos, L. F. M. Franco, M. Castier, and I. G. Economou. Molecular dynamics simulation of n-alkanes and CO_2 confined by calcite nanopores. *Energy Fuels*, 32:1934–1941, 2018.

F. Sastre, E. Moreno-Hilario, M. G. Sotelo-Serna, and A. Gil-Villegas. Microcanonical-ensemble computer simulation of the high-temperature expansion coefficients of the Helmholtz free energy of a square-well fluid. *Mol. Phys.*, 116:351–360, 2018.

M. A. Selam, M. Castier, and I. G. Economou. A thermodynamic model for strong aqueous electrolytes based on the eSAFT-VR Mie equation of state. *Fluid Phase Equilibr.*, 464: 47–63, 2018.

O. K. Smith. Eigenvalues of a symmetric 3×3 matrix. *Commun. ACM*, 4:168, 1961.

A. Stukowski. Visualization and analysis of atomistic simulation data with OVITO-the Open Visualization Tool. *Model. Simul. Mater. Sci. Eng.*, 18, 2010.

T. van Westen and J. Gross. A critical evaluation of perturbation theories by monte carlo simulation of the first four perturbation terms in Helmholtz energy expansion for the lennard-jones fluid. *J. Chem. Phys.*, 147:014503, 2017.

J. A. C. Veerman and D. Frenkel. Phase behavior of disklike hard-core mesogens. *Phys. Rev. A*, 45:5632–5648, 1992.

C. Vega and S. Lago. A fast algorithm to evaluate the shortest distance between rods. *Comput. Chem.*, 18:55–59, 1994.

E. J. W. Verwey. Theory of the Stability of Lyophobic Colloids. *J. Phys. Colloid Chem.*, 51:631–636, 1947.

E. J. W Verwey and J. Th. G Overbeek. *Theory of the stability of lyophobic colloids.* Amsterdam, 1948.

J. Vieillard-Baron. Phase transitions of the classical hard-ellipse system. *J. Chem. Phys.*, 56:4729–4744, 1972.

J. J. Wendoloski, K. H. Gardner, J. Hirschinger, H. Miura, and A. D. English. Molecular dynamics in ordered structures: Computer simulation and experimental results for nylon 66 crystals. *Science*, 247:431–436, 1990.

H. H. Wensink. Spontaneous sense inversion in helical mesophases. *EPL*, 107:36001, 2014.

H.H. Wensink and H.N.W. Lekkerkerker. Phase diagram of hard colloidal platelets: a theoretical account. *Mol. Phys.*, 107:2111–2118, 2009.

M. S. Wertheim. Fluids with highly directional attractive forces. II. Thermodynamic perturbation theory and integral equations. *J. Stat. Phys.*, 35:35–47, 1984a.

M. S. Wertheim. Fluids with Highly Directional Attractive Forces. I. Statistical Thermodynamics. *J. Stat. Phys.*, 35, 1984b.

M. S. Wertheim. Fluids with highly directional attractive forces. III. Multiple attraction sites. *J. Stat. Phys.*, 42:459–476, 1986a.

M. S. Wertheim. Fluids with highly directional attractive forces. IV. Equilibrium polymerization. *J. Stat. Phys.*, 42:477–492, 1986b.

M. S. Wertheim. Fluids of dimerizing hard spheres, and fluid mixtures of hard spheres and dispheres. *J. Chem. Phys.*, 85:2929–2936, 1986c.

M. S. Wertheim. Thermodynamic perturbation theory of polymerization. *J. Chem. Phys.*, 87:7323–7331, 1987.

D. C. Williamson and F. Del Rio. The isotropic-nematic phase transition in a fluid of square well spherocylinders. *J. Chem. Phys.*, 109:4675–4686, 1998.

D. C. Williamson and Y. Guevara. Deviation from corresponding states for a fluid of square well spherocylinders. *J. Phys. Chem. B*, 103:7522–7530, 1999.

J. Wu and J. M Prausnitz. 110th Anniversary : Molecular Thermodynamics: An Endless Frontier. *Ind. Eng. Chem. Res.*, 58:9707–9708, 2019.

L. Wu, E. A. Müller, and G. Jackson. Understanding and describing the liquid-crystalline states of polypeptide solutions: A coarse-grained model of pblg in dmf. *Macromolecules*, 47:1482–1493, 2014.

C. Zannoni. Computer simulations. In G. R. Luckhurst and G. W. Gray, editors, *Mol. Phys. Liq. Cryst.*, pages Chapter 9, 191–220. Academic Press, London, 1979.

B. J. Zhang. Calculating thermodynamic properties from perturbation theory. I. An analytic representation of square-well potential hard-sphere perturbation theory. *Fluid Phase Equilib.*, 154:1–10, 1999.

R. W. Zwanzig. High-temperature equation of state by a perturbation method. I. Nonpolar gases. *J. Chem. Phys.*, 22:1420–1426, 1954.

Appendices

A
Chapter 2

A.1 Codes

A.1.1 HGO + SW Equation of state

An input file named *hgoinput.sci* is required to run the subroutine, which takes the value of the absolute temperature and the density in kg·m^{-3} and returns the pressure in MPa. An example of the input file is given below.

```
System=     HGO_SW_fluid
Fluid_formula=   CO2
Lambda=     ! range of the potential
Sigma_0_Angstrom=  ! diameter of a sphere with same volume
Molec_Weigth(kgmol)= ! molecular weight
Epsilon_Kb(K)=   ! Well-depth divided by Boltzmann constant
Elogantion_k=   ! Elongation of the ellipsoid
Cp_Correlation_coef= A B C D E ! cp coeff. Passut and Danner 1972
```

Listing A.1: Example of the input file hgoinput.sci

```
!**********************************************************************
!**********************************************************************
! Program to calculate thermodynamic properties using hgo-saft
!**********************************************************************
! Developer: Joyce Tavares Lopes
! Supervisor: Dr. Luis Fernando Mercier Franco
! School of chemical engineering (FEQ) - UNICAMP
!**********************************************************************
! Main reference: Gil-Villegas et al. 1997
!**********************************************************************
     subroutine hgosaft(t,rhoin,p)
     implicit none
     double precision,intent(in) :: t,rhoin
     double precision rho,rhoi,rhof,deltarho,pt(7),rhoaux
     double precision cv,cp,kt,sspeed,jtc,alfa,co(8)
     double precision lambda,lam(2),sig_0,mm,epskb,k
     integer i,j,points,tint
     double precision p,dpdrho,dump,potq,miid
     character (len=20) :: filename,fluid
     character(len=100) :: get,get2

     open(10,file='hgoinput.sci')
     read(10,*) get,get2
     read(10,*) get,fluid
     read(10,*) get,lambda
     read(10,*) get,sig_0
     read(10,*) get,mm
     read(10,*) get,epskb
     read(10,*) get,k
     read(10,*) get, co(1),co(2),co(3),co(4),co(5)
     close(10)

       rho = rhoin
         call eos(t,rho,pt(1),lambda,cv,cp,kt,alfa,sspeed,jtc,sig_0,&
             &mm,epskb,k,co,dpdrho,potq,miid)
        p = pt(1)
     end

     subroutine eos(t,rho,pt,lambda,cv,cp,kt,alfa,sspeed,jtc,sig_0,&
               &mm,epskb,k,co,dpdrho,potq,miid)
     implicit none

!***********************************************************************
!Constants *
!***********************************************************************
!Boltzmann constant (j/k) *
     double precision, parameter :: kbj = 1.3806485279d-23
     double precision kb
!Pi
     double precision, parameter :: pi = 4.d0*datan(1.d0)
!Avogadro number
     double precision, parameter :: na = 6.022140857d23
!***********************************************************************
!***********************************************************************


!***********************************************************************
!Input parameters *
!***********************************************************************
!Temperature
     double precision t,tt
```

```
!Volume v (angstrom^3)
     double precision v
!Molecular weigth (kg/mol)
     double precision mm
!Number of chain molecules
     double precision n
!Number density of chain molecules (kg/m^3) and (n/v)
     double precision rho
!***********************************************************************
!***********************************************************************

!***********************************************************************
!Energy input parameters *
!***********************************************************************
!Well depth/kb (k) *
     double precision epskb,eps
!Ellipsoid shape parameters
     double precision sig_0,sig_s,sig_e
!Ellipsoid elongation and anisotropy parameters
     double precision k,chi,gama
!Square-well parameter - range of attractive forces *
     double precision lambda
!Coefficients to calculate specific heat in ideal gas state *
     double precision co(8)
!***********************************************************************
!***********************************************************************


!***********************************************************************
!Variables
!***********************************************************************
!Thermal de broglie wavelength/mass of particle
     double precision bro,mp,hp
!Total pressure, specific heats, sound speed and joule-thompson coeff. *
       double precision pt(7), cv, cp, sspeed, jtc,potq
!Isothermal compressibility and coeff. of thermal expansion *
       double precision kt, alfa
!Packing fraction and effective packing fraction *
     double precision eta(10), etaef(12), detadv
!Ideal free energy, pressure and specific heat *
     double precision aid,pid(7),cvid,cpid,miid
!Monomer free energy, pressure, specific heat and entropy *
     double precision am,pm(7),cvm,cpm,sm,patr(7)
!Hard Gaussian overlap free energy and pressure *
     double precision ahgo,phgo(7)
!Percus-Yevick isothermal compressibility *
     double precision khs
!Khs*packing fraction *
```

```
     double precision keta(11)
!Numerator and denominator of keta(2) *
     double precision num(2),den(2)
!Effective radial distribution at contact- Carnahan-Starling eos *
     double precision gef(12)
!Baker-Henderson coefficients *
     double precision a1(13),a2(10)
!Derivative of gef with respect to etaef *
     double precision dgdef(10),d2gdef(3),d3gdef(3)
!Beta = 1/(kb*t)
     double precision beta

!***********************************************************************
!***********************************************************************
!Variables to calculate numeric derivative *
     double precision dam,am1,am2,beta2,dam2,am3,cvnum,snum,snum2

     double precision dpdrho,dpdrhonum
!***********************************************************************
!Parameterization variables to calculate eta
!***********************************************************************
     integer i,j
     double precision, dimension(3) :: c
     double precision, dimension(3) :: lambs
     double precision, dimension(3,3) :: matrix
              matrix(1,1) = 2.25855d0
              matrix(1,2) = -1.50349d0
              matrix(1,3) = 0.249434d0
              matrix(2,1) = -0.66927d0
              matrix(2,2) = 1.40049d0
              matrix(2,3) = -0.827739d0
              matrix(3,1) = 10.1576d0
              matrix(3,2) = -15.0427d0
              matrix(3,3) = 5.30827d0
!***********************************************************************
!Initialization *
!***********************************************************************
!Constants
       kb = kbj*1d20 !kg*angstrom^2(k*s^2)
!Parameters and properties
       eps = epskb*kb !kg*angstrom^2(s^2)
       sig_s = sig_0**3/k
       sig_s = sig_s**(1.d0/3.d0)
       chi = (-1+k**2)/(1+k**2)
       sig_e = k*sig_s
       lambs(1) = 1.d0
       lambs(2) = lambda
       lambs(3) = lambda**2.d0
       rho = rho*na/mm !(# molecules)/volume m^3
       rho = rho*1d-30 !to (# molecules )/angstrom^3
       n = 100.d0
       mp = mm/na !(mass of each particle)
       bro = sqrt(hp*hp/(2.d0*pi*mp*kbj*t))
       v = (n/rho) !angstrom^3
       eta(1) = (rho*pi*sig_0**3.d0)/6.d0
       eta(3) = -eta(1)/v
       khs = ((1.d0-eta(1))**4.d0)/(1.d0+4.d0*eta(1)+4.d0&
                 &*eta(1)**2.d0)
       keta(1) = khs*eta(1)
       beta = 1/(kb*t)
!************************************************************************
!Parameterization calculation and effectives values *
!************************************************************************
       c(1) = 0.d0
       c(2) = 0.d0
       c(3) = 0.d0
       do 11 i=1,3
          do 10 j=1,3
               c(i) = matrix(i,j)*lambs(j) + c(i)
   10 continue
   11 continue
       etaef(1) = c(1)*eta(1) + c(2)*eta(1)**2.d0 + c(3)*eta(1)**3.d0
       etaef(2) = c(1) + 2.d0*c(2)*eta(1) + 3.d0*c(3)*eta(1)**2.d0
       etaef(4) = 2.d0*c(2) + 6.d0*c(3)*eta(1)
       etaef(12) = 6.d0*c(3)
       etaef(3) = etaef(2)*eta(3)
       eta(5) = 2.d0*eta(1)/(v**2.d0)
       eta(6) = 0!-2.d0/v
       eta(10) = -2.d0/v
       gef(1) = (1.d0-0.5d0*etaef(1))/((1.d0-etaef(1))**3.d0)

!***********************************************************************
!Ideal contribution calculations
!***********************************************************************
      aid = (n*kb*t)*(log(rho*bro*bro*bro) - 1.d0)
      pid(1) = n*kb*t/v !kg/(angstrom*s^2)
      pid(3) = -kb*t*n/(v**2)
      pid(7) = kb*n/v
      miid = (aid+pid(1)*v)/n
      miid = miid*1d4*1d-30*na/mm !to mjoule/kg
!Specific heat - passut, c. a., & danner, r. p. (1972). correlation of
!Ideal gas enthalpy, heat capacity, and entropy. ind. eng. process. des.
!Develop., 11(4), -543546.
!Cp = b + 2ct+3ct**2+4et**3+5ft**4 --> btu/(lb ºr )
!Btu/(lb ºr ) to j/(kg k) --> *4186.8
!B = co(1), c=co(2), d=co(3), e=co(4),f=co(5)
```

```
         tt = 1.8d0*t
         cpid = co(1)+2.d0*co(2)*tt+3.d0*co(3)*tt**2.d0+4.d0*&
                &co(4)*tt**3.d0+5.d0*co(5)*tt**4.d0
         cpid = cpid*4186.800000009d0*mm !to j/mol k
         cvid = cpid - na*kbj !cv = cp - r !kb back to j
!**********************************************************************
!**********************************************************************


!**********************************************************************
!Monomer contribution calculations
!**********************************************************************
!Hard Gaussian overlap
      gama = 0.5d0*(1.d0+asin(chi)/(chi*sqrt(1.d0-chi*chi)))
!Ahgo/nkbt
      ahgo = gama*eta(1)*(4.d0-3.d0*eta(1))/((1.d0-eta(1))**2.d0)
      phgo(1) = n*kb*t*gama*eta(3)*((2.d0*eta(1)-4.d0)&
               &/((1.d0-eta(1))**3.d0))
      phgo(3) = n*kb*t*gama*(eta(5)*((2.d0*eta(1)-4.d0)/&
               &((1.d0-eta(1))**3.d0))&
               &+eta(3)*eta(3)*((4.d0*eta(1)-10.d0)/&
              &((1.d0-eta(1))**4.d0)))
      phgo(7) = phgo(1)/t

!Dispersion
       a1(1) = -4.d0*eta(1)*eps*(-1.d0+lambda**3)*gef(1)
       dgdef(1) = (2.5d0-etaef(1))/((1-etaef(1))**4.d0)
       gef(2) = dgdef(1)*etaef(2)
       a1(2) = gef(1)+gef(2)*eta(1)
       a1(2) = -4.d0*eps*(-1.d0+lambda**3.d0)*a1(2)

       a2(1) = 0.5d0*eps*eta(1)*khs*a1(2)
!Total --> hgo + attractive
       am = (ahgo + a1(1)*beta + a2(1)*beta**2)*n*kb*t
!Derivativess with respect to v to calculate pressure
      d2gdef(1) = (9.d0-3.d0*etaef(1))/(1.d0-etaef(1))**5.d0
      d3gdef(1) = (42.d0-12.d0*etaef(1))/(1.d0-etaef(1))**6.d0
      dgdef(2) = etaef(2)*d2gdef(1)
       gef(4) = etaef(2)*dgdef(2) + dgdef(1)*etaef(4)
       gef(3) = dgdef(1)*etaef(3)
       a1(4) = -4.d0*eps*(-1.d0+lambda**3)*(2.d0*gef(2)&
                &+eta(1)*gef(4))
       a1(6) = (-eta(1)/v)*a1(4)
      dgdef(3) = eta(3)*dgdef(2)
      etaef(6) = eta(3)*etaef(4)!+eta(6)*etaef(2)
      gef(6) = etaef(6)*dgdef(1) + etaef(2)*dgdef(3)
       num(1) = ((1.d0+4.d0*eta(1)+4.d0*eta(1)**2)*&
                &(((1.d0-eta(1))**4.d0)-&
                 &4.d0*eta(1)*(1.d0-eta(1))**3)-eta(1)&
                 &*(4.d0+8.d0*eta(1))&
                 &*(1.d0-eta(1))**4.d0)
      den(1) = ((1.d0+4.d0*eta(1)+4*eta(1)**2.d0)**2.d0)
      keta(2) = num(1)/den(1)
      keta(3) = keta(2)*(-eta(1)/v)
      a2(3) = 0.5d0*eps*((keta(3))*a1(2)+khs*eta(1)*a1(6))
      a1(3) = ((-eta(1))/v)*a1(2)
      pm(1) = (n*kb*t)*(phgo(1)/(n*kb*t) -&
                 & a1(3)*beta- a2(3)*beta**2)
!Entropy
      sm = (a2(1)*beta/t - kb*ahgo)*n


!Constant volume
      cvm = -2.d0*n*a2(1)*beta/t

!Derivatives to calculate kt, alfa and cp
!Calculation of d2a1/dv2
        etaef(5) = eta(3)*(etaef(4)*eta(3)+eta(10)*etaef(2))
        gef(5) = dgdef(3)*etaef(3)+etaef(5)*dgdef(1)
        a1(5) =-4.d0*eps*(-1.d0+lambda**3.d0)*(eta(3)*gef(2)+eta(10)*&
                 &gef(1)+gef(3)+eta(1)*(gef(5)/eta(3)))*eta(3)
        a1(5) = eta(3)*(a1(4)*eta(3)+eta(10)*a1(2))
        a1(5) = -4.d0*eps*(-1.d0+lambda**3.d0)*(2.d0*gef(3)*eta(3)+&
                 &eta(5)*gef(1)+gef(5)*eta(1))
!Calculation of d2a2/dv2
       num(2) = ((1.d0-eta(1))**4.d0)*(4.d0+8.d0*eta(1)) - 4.d0*((1.d0-
          eta(1))&
                 &**3)*(1.d0+4.d0*eta(1)+4.d0*eta(1)**2)+&
                 &3.d0*((1.d0-eta(1))**2&
                 &)*(4.d0*eta(1)+(16.d0*eta(1)**2)+16.d0*eta(1)**3)-((1.d0
                    -eta(1))&
                 &**3)*(4.d0+32.d0*eta(1)+48.d0*eta(1)**2.d0)-((1.d0-eta
                    (1))&
                 &**4.d0)*(4.d0+16.d0*eta(1))+4.d0*((1.d0-eta(1))**3)*(4.
                    d0*&
                 &eta(1) + 8.d0*eta(1)**2)
       den(2) = 2.d0*(1.d0+4.d0*eta(1)+4.d0*eta(1)**2.d0)*(4.d0+8.d0*eta
          (1))
       keta(4) = (num(2)*den(1) - den(2)*num(1))/(den(1)**2.d0)
       keta(10) = eta(3)*keta(4)+eta(10)*keta(2)
       d2gdef(2) = d3gdef(1)*etaef(2)
       gef(12) = etaef(12)*dgdef(1) + dgdef(2)*etaef(4)+&
                   &2.d0*etaef(2)*etaef(4)*d2gdef(1)+d2gdef(2)*etaef(2)
                      **2.d0
       a1(12) = -4.d0*eps*(-1.d0+lambda**3.d0)*(3.d0*gef(4)+eta(1)*gef
          (12))
       a1(11) = eta(10)*a1(4)+a1(12)*eta(3)
       a2(10) = 0.5d0*eps*(keta(10)*a1(2)+a1(4)*keta(3)&
                  &+a1(11)*keta(1)+keta(2)*a1(6))
       a2(5) = eta(3)*a2(10)

!Dp/dv
       patr(3) = n*kb*t*(-a1(5)*beta-a2(5)*beta**2)
       pm(3) = n*(phgo(3)+patr(3))/n
!Derivative of pressure with respect to t
       patr(7) = n*(a2(3)*beta/t)
       pm(7) = n/n*(phgo(7)+patr(7))


!***********************************************************************
!Total properties
!***********************************************************************
      pt(1) = pid(1) + pm(1) !+ pc(1)
      potq = (aid+am+pt(1)*v)/n
      potq = potq*1d4*1d-30*na/mm !to mpa*m^3/kg
      pt(1) = pt(1)*1d10*1d-6 !convert to mpa
      cv = cvid + cvm*1d-20*na/n !cvid is already in j/mol.k
!Derivatives of total pressure
        pt(3) = pm(3) + pid(3) !+ pc(3)
        kt = -1.d0/(v*pt(3))
        pt(7) = pm(7) + pid(7) !+ pc(7)
        alfa = kt*pt(7)
        cp = t*v*(alfa**2.d0)/kt
        cp = cp*1d-20*na/n + cv !convert to j/mol.k and add cv
        kt = kt*1d-10
!Rho is n/v(ang^3) but v is extensive. convert rho to extensive:
        rho = rho/n
!Pressure is kg*angs^-1*s^-2. so i have to divide for the system total
!Mass = n*mm/na
        sspeed = sqrt(-(cp/cv)*pt(3)/(rho**2)/(n*mm/na)) !angs/s
        sspeed = sspeed*1d-10
!To calculate the joule-thompson coefficient, cp has to be heat capacity
!= j/k. but cp is already in j/mol.k. so it has to be multiplied by
!N/na
        jtc = -(v-t*v*alfa)/(cp*n/na)
        jtc = jtc*1d-24 !k*mpa^-1
!**********************************************************************
!**********************************************************************
!Dp/drho
       pt(3) = pt(3)*1d34 !mpa/m^3
       pt(3) = pt(3)*n/na !mpa*mol/m^3
       pt(3) = pt(3)*mm !mpa*kg/m^3
       v = v*1d-30/n*na/mm !from ang^3 to m^3/kg
       rho = rho*1d30*n/na*mm !kg/m^3
       dpdrho = -pt(3)*v*v !mpa*m^3/kg
!***********************************************************************


end
```

B
Chapter 3

B.1 Quaternions

The quaternion number system was conceived by William Hamilton in the nineteenth century as an extension of the complex number system. This four-dimensional number system is vastly used to describe rotations of vectors in three dimensions. A quaternion q is defined as follows:

$$q = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} \tag{B.1}$$

where $i^2 = j^2 = k^2 = ijk = -1$, and q_0 is the scalar part of the number. The quaternion algebra is noncommutative and a unit quaternion, where $\sqrt{q_1^2 + q_2^2 + q_3^2} = 1$, represents a three-dimensional rotation. A rotation-quaternion that executes a rotation about an unit vector $\hat{a}$ by an angle θ is described as:

$$\begin{aligned} q_0 =& \cos\left(\frac{\theta}{2}\right) \\ q_1 =& a_1 \sin\left(\frac{\theta}{2}\right) \\ q_2 =& a_2 \sin\left(\frac{\theta}{2}\right) \\ q_3 =& a_3 \sin\left(\frac{\theta}{2}\right) \end{aligned} \tag{B.2}$$

Then, to rotate a vector $\mathbf{e}$ around $\hat{a}$ by θ, one should apply the following opetation:

$$\mathbf{e}' = q\mathbf{e}q^{-1} \tag{B.3}$$

where $\mathbf{e}'$ is the vector $\mathbf{e}$ after the rotation, $q^{-1} = q_0 - q_1 i - q_2 j - q_3 k$ is the inverse of q and represents the same rotation. The operation in Equation B.3 can be rewritten in another form as follows:

$$\mathbf{e}' = A^T \mathbf{e} \tag{B.4}$$

where A^T is the transpose of a rotation matrix A defined as:

$$A = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2) \\ 2(q_1q_2 - q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 + q_0q_1) \\ 2(q_1q_3 + q_0q_2) & 2(q_2q_3 - q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \tag{B.5}$$

B.2 Codes

B.2.1 NVT Monte Carlo code for HGO + SW potential

The code requires two input files:

- *conf0_ellip.xyz*: initial configuration file. The first line is the total number of particles, second line is the x, y and z dimensions of the simulation box, and from the third line on, we have the positions and orientations of each particle (type of the molecule, x position, y position, z position, scalar part of the quaternion, x component of the quaternion, y component of the quaternion, z component of the quaternion, σ_s, σ_s and σ_e).
- *hgo_isosw_input.sci*. An example of the file is given below.

```
Number_of_steps=
Print_every_x_steps= x !Print properties every x steps
Max_rotational_displacement_rad=
Max_translational_displacement=
Adjust_drmax_every_y_step= y !Adjust maximum displacement
Eps0= 1d0 ! Well-depth (reduced)
Ellipdois_sigs_sige= a b ! a is sigma_s and b is sigma_e
Range_parameter_lambda=
Reduce_Temperature=
Molecule_Type= 1
```

Listing B.1: Example of the input file hgo_isosw_input.sci

The output files are *conf_hgo_isosw.xyz* (file with the trajectory) and *prop.dat*: step, acceptance ratio, maximum translational displacement, elapsed time, total potential, and nematic order parameter.

```
!**********************************************************************
! NVT Monte simulation of HGO + SW potential
!**********************************************************************
```

```
! Developer: Joyce Tavares Lopes
! Supervisor: Dr. Luis Fernando Mercier Franco
! University of Campinas - School of Chemical Engineering
!*********************************************************************
! Matrix r(:,N) for N particles
! r(1,N) = x-axis
! r(2,N) = y-axis
! r(3,N) = z-axis
! Matrix q(:,N)
! q(0,N) = quaternion w (real number)
! q(1,N) = quaternion x (i imaginary number)
! q(2,N) = quaternion y (j imaginary number)
! q(3,N) = quaternion z (k imaginary number)
!*********************************************************************
! Energy variables are reduced
!*********************************************************************

Program HGOSW_MC_NVT
 Implicit None
 Integer :: seed = 349766914 !Input for the pseudo-random number
    algorithm
 Real(8) :: rnum
 Real(8), Allocatable :: r(:,:),q(:,:),e(:,:) !Position, quaternion and
    orientation
 Real(8) :: sige,sigs,boxl(3)
 Integer :: n,i,j,k
 Character :: get*100,moltype*1
 Real(8) :: sigma,sig,chi,chil,ke,kel,modrij,rcut,utotal
 Real(8), dimension(3) :: e1,e2,rij,ri,efixed,ei,urij
 Integer :: moves,nsteps,step,step2print,stepdrmax
 Real(8) :: max_angle, max_r,eps,uatrgb,urepgb,acc_mov
 Real(8) :: ni,mi,eps0
 Real(8), dimension(0:3) :: qi
 Real(8) :: deltap,new_pot,old_pot
 Real(8) :: lambda,tr,t !Reduced temperature and temperature
 Logical :: overlap
 Real :: start,finish,startstep,finishstep
 Real(8),dimension(0:3) :: qnew,qteste
 Real(8),Allocatable :: odf(:),theta(:)
 Real(8) :: s2 !Order Parameter
 Integer :: maxbinori
 Real :: dump
 Call cpu_time(start)

!Simulation Input Files

 !Monte Carlo simulation and Potential parameters
 Open(101,File= 'hgo_isosw_input.sci')
```

```
        Read(101,*) get,nsteps
        Read(101,*) get,step2print
        Read(101,*) get,max_angle
        Read(101,*) get,max_r
        Read(101,*) get,stepdrmax
        Read(101,*) get,eps0 !Well-depth
        Read(101,*) get,sigs,sige !Ellipsoid parameters
        Read(101,*) get,lambda
        Read(101,*) get,tr
        Read(101,*) get,moltype
 Close(101)


!Simulation Output Files
 Open(200,File='conf_hgo_isosw.xyz')
  Open(201,File='prop.dat')
 Write(201,*) ' Step ','acc_mov ','dr_max &
  & ',' time ',' U/N '&
        &,' Order Parameter'

!Read Initial Configuration File
 Open(102,FILE='conf0_ellip.xyz')
 Read(102,*) n
!Box length (1) = x-axis, (2) = y-axis, (3) = z-axis
 Read(102,*) boxl(1),boxl(2),boxl(3)
 Write(200,*) n
 Write(200,*) boxl(1),boxl(2),boxl(3)

!Allocate Matrix of particles positions and orientations
 Allocate(r(3,N),q(0:3,N),e(3,N))
    Do 99 i=1,n
        Read(102,*) moltype,r(1,i),r(2,i),r(3,i),q(0,i),q(1,i),&
                    &q(2,i),q(3,i),sigs,sigs,sige
        Write(200,*) moltype,r(1,i),r(2,i),r(3,i),q(0,i),q(1,i),&
                    &q(2,i),q(3,i),sigs,sigs,sige
 99 End Do
 Close(102)

!Calculates elongation 'ke' and anisotropy 'chi'
        ke = sige/sigs
        chi = (ke*ke - 1.d0)/(ke*ke + 1.d0)
        chil = (kel**(1.d0/mi) - 1.d0)/(kel**(1.d0/mi) + 1.d0)
        rcut = 4.d0*sigs
        max_r = max_r*sigs

!Calculate orientations from initial quaternion rotation
        efixed = (/0.d0,0.d0,1.d0/) !Ellipsoid orientation with no
           rotation
```

```
       Do 100 i=1,n !Loop over particles
              Call quat_to_ori(efixed,q(:,i),e(:,i))
   100 End Do

!Detect overlap in intial configuration
       Do 101 i=1,n-1
              Do 102 j=i+1,n

                     e1 = e(:,i)
                     e2 = e(:,j)
                     rij = r(1:3,i) - r(1:3,j)
                     modrij = sqrt(rij(1)*rij(1) + rij(2)*rij(2) + rij(3)
                        *rij(3))
                     urij = rij/modrij
                     sig = sigma(e1,e2,urij,sigs,chi)
                     If (modrij .lt. sig) then
                            print*,'Overlap detected in initial
                               configuration'
                            STOP
                     End if
          102 End Do
   101 End Do

 Call total_pot(boxl,n,sigs,sige,e,r,eps0,chi,chil,mi,ni,rcut,lambda,
    utotal)
 print*,'Initial Total Potential Energy per particle=',utotal/n

!Start trial moves
       Do 103 step=1,nsteps !Loop over steps
              Call cpu_time(startstep)
              moves = 0
              acc_mov = 0
              Do 104 i=1,n !Loop over particles
                     Call partial_pot(boxl,n,i,sigs,sige,e(:,i)&
                                  &,r(:,i),e,r,eps0,&
                                  &chi,chil,mi,ni,rcut,lambda,old_pot,
                                     overlap)
                     If (overlap) then
                            print*,'Error: overlap detected'
                            STOP
                     End If

                    !Translation Move for particle i
                     Call rand_numb(seed,rnum)
                     ri(1) = r(1,i) + (2.d0*rnum - 1.d0)*max_r
                     Call rand_numb(seed,rnum)
                     ri(2) = r(2,i) + (2.d0*rnum - 1.d0)*max_r
                     Call rand_numb(seed,rnum)
```

```
                ri(3) = r(3,i) + (2.d0*rnum - 1.d0)*max_r

               !Coordinates in central box after translation
                ri(1) = ri(1) - boxl(1)*Anint(ri(1)/boxl(1))
                ri(2) = ri(2) - boxl(2)*Anint(ri(2)/boxl(2))
                ri(3) = ri(3) - boxl(3)*Anint(ri(3)/boxl(3))

               !Rotational Move for particle i
               !Randomly rotate old quaternion
                Call random_rotate_quat(seed,q(:,i),max_angle,qi)
               !New orientation after rotation
                Call quat_to_ori(efixed,qi,ei)

                Call partial_pot(boxl,n,i,sigs,sige,ei,ri,e,r,eps0,
                    chi,chil,&
                                 &mi,ni,rcut,lambda,new_pot,overlap)

                If (.not. overlap) then
                       deltap = new_pot - old_pot
                       deltap = deltap/tr
                       If (deltap .lt. 0) then
                              q(:,i) = qi(:)
                              r(:,i) = ri(:)
                              e(:,i) = ei(:)
                              utotal = utotal + new_pot - old_pot
                              moves = moves + 1
                       Else if ((deltap/eps0) .lt. 75) then
                              Call rand_numb(seed,rnum)
                              If (exp(-(deltap/(eps0))) .gt. rnum)
                                 then
                                     q(:,i) = qi(:)
                                     r(:,i) = ri(:)
                                     e(:,i) = ei(:)
                                     utotal = utotal + new_pot -
                                        old_pot
                                     moves = moves + 1
                              End If
                       End if
                End if
104 End Do
    Call cpu_time(finishstep)
    !Adjust maximum displacement
    acc_mov = dble(moves)/dble(n)
    If (mod(step,stepdrmax) == 0) then
          If (acc_mov .gt. 0.5) then
                 max_r = 1.05*max_r
          Else
                 max_r = 0.95*max_r
```

```
                        End If
                End If
                If (Mod(step,step2print) == 0) then
                        Call orderparameter(n,e,s2)
                        Write(201,*) step,real(acc_mov),real(max_r),&
                          finishstep-startstep,utotal/n,real(s2)

                       !Write Output File (position and quaternions)
                        Write(200,*) n
                        Write(200,*) boxl(1),boxl(2),boxl(3)
                        Do 200 i=1,n
                                Write(200,*) moltype,r(1,i),r(2,i),r(3,i),&
                                             &q(0,i),q(1,i),q(2,i),q(3,i),sigs
                                                 ,sigs,sige
                        200 End DO
                End If
    103 End Do



 Call cpu_time(finish)
 Print*,'Final Total Potential Energy per particle=',utotal/n
 Print*,'Total run time=',finish-start,'s'

        Call orderparameter(n,e,s2)
        Print*,'Final Order Parameter=',s2

     Close(200)
     Close(201)
End

!**********************************************************************
!Subroutines used within the code
!**********************************************************************

!Calculates contact distance of ellipsoids given their orientaions and
   the orientation of the vector joining their centers of mass
Double Precision Function sigma(e1,e2,urij,sig0,chi)
 Implicit None
! Scalar Product of unit vector and orientations and between the two
   orientations
 Real(8) :: re1,re2,e1e2
! urij is the unit vector of vector rij joining the particles centers of
   mass
 Real(8), Dimension(3) :: urij,rij
! Particles orientation
 Real(8), Dimension(3) :: e1,e2
 Real(8) :: chi,sig0

 re1 = urij(1)*e1(1) + urij(2)*e1(2) + urij(3)*e1(3)
 re2 = urij(1)*e2(1) + urij(2)*e2(2) + urij(3)*e2(3)
 e1e2 = e1(1)*e2(1) + e1(2)*e2(2) + e1(3)*e2(3)
 sigma = sig0*(1.d0-0.5d0*chi*((re1+re2)**2.d0/(1.d0+chi*e1e2)+&
          &(re1-re2)**2.d0/(1.d0-chi*e1e2)))**(-0.5)

End

!Takes the old orientation e, the rotation quaternion q and returns the
!new orientation after rotation
Subroutine quat_to_ori(e,q,enew)
 Implicit None
 Real(8),Dimension(3),Intent(in) :: e
 Real(8),Dimension(3),Intent(out) :: enew
 Real(8),Dimension(0:3),Intent(in) :: q
 Real(8),Dimension(3,3) :: rotM, rotMT
 Integer :: i,j

 !Rotation Matrix rotM(3,3) - Allen and Tildesley, 2th edition page 110
 rotM(1,1) = q(0)*q(0) + q(1)*q(1) - q(2)*q(2) - q(3)*q(3)
 rotM(1,2) = 2.d0*(q(1)*q(2) + q(0)*q(3))
 rotM(1,3) = 2.d0*(q(1)*q(3) - q(0)*q(2))
 rotM(2,1) = 2.d0*(q(1)*q(2) - q(0)*q(3))
 rotM(2,2) = q(0)*q(0) - q(1)*q(1) + q(2)*q(2) - q(3)*q(3)
 rotM(2,3) = 2.d0*(q(2)*q(3) + q(0)*q(1))
 rotM(3,1) = 2.d0*(q(1)*q(3) + q(0)*q(2))
 rotM(3,2) = 2.d0*(q(2)*q(3) - q(0)*q(1))
 rotM(3,3) = q(0)*q(0) - q(1)*q(1) - q(2)*q(2) + q(3)*q(3)

 !Transpose of Rotation Matrix
 Do i=1,3
       Do j=1,3
              rotMT(i,j) = rotM(j,i)
       End Do
 End Do

 !New orientation
 enew(1) = e(1)*rotMT(1,1) + e(2)*rotMT(1,2) + e(3)*rotMT(1,3)
 enew(2) = e(1)*rotMT(2,1) + e(2)*rotMT(2,2) + e(3)*rotMT(2,3)
 enew(3) = e(1)*rotMT(3,1) + e(2)*rotMT(3,2) + e(3)*rotMT(3,3)
End


!Generates a random number between [0,1]
Subroutine Rand_numb(seed,rnum)
 Implicit None
 Integer :: seed
 Integer, Parameter :: a=1029, b=221591, M=1048576
 Real(8) :: rnum
 seed = mod((a*seed+b),M)
 rnum = dble(seed)/dble(M)
End

!Generates a vector on the surface of a unit sphere.
!Allen and Tildesley 2th edition, page 514 (routine:Marsaglia 1972).
Subroutine Rand_vector(seed,v)
 Implicit None
 Real(8), Intent(out) :: v(3)
 Real(8) :: n1,n2,nsq
 Integer, Intent(in) :: seed

 nsq = 2.d0
 Do while (nsq .gt. 1.d0)
        Call rand_numb(seed,n1)
        Call rand_numb(seed,n2)
        n1 = 2d0*n1 - 1.d0
        n2 = 2d0*n2 - 1.d0
        nsq = n1*n1 + n2*n2
 End Do
        v(1) = 2.d0*n1*sqrt(1.d0-nsq)
        v(2) = 2.d0*n2*sqrt(1.d0-nsq)
        v(3) = 1.d0-2.d0*nsq
End

!Randomly rotates a quartenion given a maximum angle
Subroutine random_rotate_quat(seed,qold,ang_max,qnew)
 Implicit None
 Real(8),Dimension(0:3),Intent(in) :: qold
 Real(8) :: ang_max
 Real(8),Dimension(0:3),Intent(out) :: qnew
 Real(8),Dimension(0:3) :: qrot
 Integer :: seed
 Call rotation_quat(seed,ang_max,qrot)
 Call multiply_quats(qrot,qold,qnew)
End Subroutine

!Generates a rotation quaternion
Subroutine rotation_quat(seed,max_ang,quat)
 Implicit None
 Integer,Intent(in) :: seed
 Real(8),Intent(in) :: max_ang
 Real(8),Dimension(0:3) :: quat
 Real(8) :: rn,ang
 Real(8),Dimension(3) :: axis
 Call rand_numb(seed,rn) !Random number 'rn' in range [0,1]
 rn = 2.d0*rn - 1d0 !Random number 'rn' now in range [-1,1]
 ang = max_ang*rn !Random angle
 quat(0) = cos(ang*0.5)
 !Random axis
 Call rand_vector(seed,axis)
 quat(1:3) = sin(ang*0.5)*axis(:)
End Subroutine

!Multiplies two quaternions
Subroutine multiply_quats(a,b,qab)
 Implicit None
 Real(8),Dimension(0:3),Intent(in) :: a,b
 Real(8),Dimension(0:3),Intent(out) :: qab

 qab(0) = a(0)*b(0) - a(1)*b(1) - a(2)*b(2) - a(3)*b(3)
 qab(1) = a(1)*b(0) + a(0)*b(1) - a(3)*b(2) + a(2)*b(3)
 qab(2) = a(2)*b(0) + a(3)*b(1) + a(0)*b(2) - a(1)*b(3)
 qab(3) = a(3)*b(0) - a(2)*b(1) + a(1)*b(2) + a(0)*b(3)
End Subroutine

!Calculates resulting epsilon of the Gay-Berne potential
 Subroutine eps_calc(e1,e2,eps0,chi,chil,mi,ni,urij,eps)
 Implicit None
 Real(8),Dimension(3),Intent(in) :: e1,e2,urij
 Real(8),Intent(in) :: eps0,chi,chil,mi,ni
 Real(8) :: eps1,eps2,e1e2,re1,re2
 Real(8), Intent(out) :: eps

 re1 = urij(1)*e1(1) + urij(2)*e1(2) + urij(3)*e1(3)
 re2 = urij(1)*e2(1) + urij(2)*e2(2) + urij(3)*e2(3)
 e1e2 = e1(1)*e2(1) + e1(2)*e2(2) + e1(3)*e2(3)
 eps1 = (1.d0 - chi*chi*e1e2*e1e2)**(-0.5d0)
 eps2 = 1.d0-0.5d0*chil*((re1+re2)**2.d0/(1.d0+chil*e1e2)+(re1-re2)**2.d0
    /(1.d0-chil*e1e2))
 eps = eps0*(eps1**ni)*(eps2**mi)
End Subroutine

!Calculates attractive part of the Gay-Berne potential
Double precision Function uatrgb(sigma,sigs,e1,e2,distij,eps0,chi,chil,mi
   ,ni,urij)
 Implicit None
 Real(8),Intent(in) :: sigma,sigs,distij,eps0,chi,chil,mi,ni
 Real(8),Dimension(3),Intent(in) :: e1,e2,urij
 Real(8) :: eps
 Call eps_calc(e1,e2,eps0,chi,chil,mi,ni,urij,eps)
 uatrgb = - 4.d0*eps*(sigs/(distij - sigma + sigs))**(6.d0)
End Function
```

```
!Calculates the repulsive part of the Gay-Berne potential
Double precision Function urepgb(sigma,sigs,e1,e2,distij,eps0,chi,chil,mi
   ,ni,urij)
 Implicit None
 Real(8),Intent(in) :: sigma,sigs,distij,eps0,chi,chil,mi,ni
 Real(8),Dimension(3),Intent(in) :: e1,e2,urij
 Real(8) :: eps
 Call eps_calc(e1,e2,eps0,chi,chil,mi,ni,urij,eps)
 urepgb = 4.d0*eps*(sigs/(distij - sigma + sigs))**(12.d0)
End Function

!Checks if the trial move of particle i causes any overlap.
!It goes over the other particles until an overlap is found.
!Then, if an overlap is found, it immediately returns the overlap as true
   .
!If it does not find any overlap, it returns as false.
Subroutine check_overlap(n,boxl,i,ei,ri,e,r,sigs,sige,chi,overlap)
 Implicit None
!Position and Orientation of moved particle
 Real(8),Dimension(3),Intent(in) :: ri,ei,boxl
!Position and Orientation of all particles
 Real(8),Dimension(3,n),Intent(in) :: r,e
 Real(8),Intent(in) :: sigs,sige,chi
 Logical,Intent(out) :: overlap
 Integer,Intent(in) :: i,n
 Integer :: j
 Real(8) :: sig,sigma,modrij,maxsig
 Real(8),Dimension(3) :: rij,urij
 maxsig = max(sige,sigs)
  Do j=1,n
        If (i .ne. j) then
                rij(:) = ri(:) - r(:,j)

                !Minimum Image
                rij(1) = rij(1) - boxl(1)*Anint(rij(1)/boxl(1))
                rij(2) = rij(2) - boxl(2)*Anint(rij(2)/boxl(2))
                rij(3) = rij(3) - boxl(3)*Anint(rij(3)/boxl(3))

                modrij = sqrt(rij(1)*rij(1)+rij(2)*rij(2)+rij(3)*rij(3))
                If (modrij .lt. maxsig) then
                        urij(:) = rij(:)/modrij
                        sig = sigma(ei,e(:,j),urij,sigs,chi)
                        If (modrij .le. sig) then
                                overlap = .true.
                                Return
                        End If
                End If
        End if
    End Do

    overlap = .false.
End Subroutine

Subroutine partial_pot(boxl,n,i,sigs,sige,ei,ri,e,r,eps0,chi,chil,mi,ni,&
                    &rcut,lambda,potij,overlap)
 Implicit None
 Integer,Intent(in) :: n,i
 Real(8),Intent(in) :: sigs,sige,eps0,chi,chil,mi,ni,rcut,lambda
 Real(8),Dimension(3),Intent(in) :: ei,ri, boxl
 Real(8),Dimension(3,n),Intent(in) :: e,r
 Real(8),Intent(out) :: potij
 Logical, Intent(out) :: overlap
 Real(8),Dimension(3) :: rij,urij
 Real(8) :: modrij != distij
 Integer :: j
 Real(8) :: uatr,urep,sig,sigma,urepgb,uatrgb
 Real(8) :: swrange,eps,sigsphere


 sigsphere = sigs*sigs*sige
 sigsphere = sigsphere**(1.d0/3.d0)
 swrange = lambda*sigsphere
 potij = 0.d0
 overlap = .false.
 Do j = 1,n
        If (j .ne. i) then
               rij(:) = ri(:) - r(:,j)
               !Minimum image
               rij(1) = rij(1) - boxl(1)*Anint(rij(1)/boxl(1))
               rij(2) = rij(2) - boxl(2)*Anint(rij(2)/boxl(2))
               rij(3) = rij(3) - boxl(3)*Anint(rij(3)/boxl(3))
               !Calculate versor of rij (urij)
               modrij = sqrt(rij(1)*rij(1)+rij(2)*rij(2)+rij(3)*rij(3))
               If (modrij .lt. rcut) then
                      urij(:) = rij(:)/modrij
                      sig = sigma(ei,e(:,j),urij,sigs,chi)
                      If (modrij .lt. sig) then
                             overlap = .true.
                             Return
                      Else if (modrij .lt. swrange) then
                             potij = potij - eps0
                      End If
               End If
        Else
        End if
End Do
End Subroutine

Subroutine total_pot(boxl,n,sigs,sige,e,r,eps0,chi,chil,mi,ni,rcut,lambda
    ,utotal)
Implicit None
Integer,Intent(in) :: n
Real(8),Intent(in) :: sigs,sige,eps0,chi,chil,mi,ni,rcut,lambda
Real(8),Dimension(3),Intent(in) :: boxl
Real(8),Dimension(3,n),Intent(in) :: e,r
Real(8),Intent(out) :: utotal
Real(8),Dimension(3) :: rij,urij
Real(8) :: modrij != distij
Integer :: i,j
Real(8) :: uatr,urep,sig,sigma,uatrgb,urepgb, swrange,eps
utotal = 0.d0
Do i = 1,n-1
        Do j = i + 1,n
               rij(:) = r(:,i) - r(:,j)
               !Minimum image
               rij(1) = rij(1) - boxl(1)*Anint(rij(1)/boxl(1))
               rij(2) = rij(2) - boxl(2)*Anint(rij(2)/boxl(2))
               rij(3) = rij(3) - boxl(3)*Anint(rij(3)/boxl(3))
               !Calculate versor of rij (urij)
               modrij = sqrt(rij(1)*rij(1)+rij(2)*rij(2)+rij(3)*rij(3))
               If (modrij .lt. rcut) then
                      urij(:) = rij(:)/modrij
                      sig = sigma(e(:,i),e(:,j),urij,sigs,chi)
                      swrange = lambda*sig
                      If (modrij .lt. sig) then
                             print*,'Overlap detected: not possible for
                                 this potential'
                             STOP
                      Else if (modrij .lt. swrange) then
                             uatr = uatrgb(sig,sigs,e(:,i),e(:,j),&
                                      &modrij,eps0,chi,chil,mi,ni,urij)
                             utotal = utotal - eps0
                      End If
               End If
        End Do
End Do
End Subroutine

Subroutine orderparameter(n,e,s)
Implicit None
Integer, Intent(in) :: n
Real(8), Intent(in),Dimension(3,n) :: e !Orientation
Real(8), Parameter,Dimension(3) :: efixed = (/0.d0,0.d0,1.d0/)
!Order tensor of particle i and average
Real(8), Dimension(3,3) :: Qabi,Qab
Real(8) :: dkro,m,dump,qq,eigenv(3),p,phi,det,pq
Real(8), Intent(out) :: s
Integer :: io,i,j,nstep,step,alpha,beta
Character :: infile*100, outfile*100

Qabi(:,:) = 0.d0
s = 0.d0
Qab(:,:) = 0.d0

!Construct order tensor Qab
Do i =1,n !Loop over particles
      Do alpha = 1,3
             Do beta = 1,3
                    If (alpha .eq. beta) then
                           dkro = 1.d0
                    Else
                           dkro = 0.d0
                    End If
                    Qabi(alpha,beta) = 1.5d0*e(alpha,i)*e(beta,i) - 0.5
                       d0*dkro + &
                                         & Qabi(alpha,beta)
             End Do
      End Do
End Do
Qab(:,:) = Qabi(:,:)/dble(n)

!Find Eigenvalues of Qab
!Smith, O. K.(1961). Eigenvalues of a symmetric 3 x 3 matrix.
!Communications of the ACM, 4(4), 168.
!https://doi.org/10.1145/355578.366316tps://doi.org
  /10.1145/355578.366316)
m = (Qab(1,1) + Qab(2,2) + Qab(3,3))/3.d0
det = (Qab(1,1)-m) *(Qab(2,2)-m)*(Qab(3,3)-m) + Qab(2,3)*Qab(1,2)*Qab
    (3,1) +&
        &Qab(1,3)*Qab(2,1)*Qab(3,2) - Qab(1,3)*(Qab(2,2)-m)*Qab(3,1)&
        & - (Qab(1,1)-m)*Qab(2,3)*Qab(3,2) - Qab(1,2)*Qab(2,1)*(Qab(3,3)-
           m)
qq = 0.5d0*det
p = (Qab(1,1)-m)*(Qab(1,1)-m)+(Qab(2,2)-m)*(Qab(2,2)-m)+(Qab(3,3)-m)*(
    Qab(3,3)-m)&
     & + Qab(1,2)*Qab(1,2) + Qab(1,3)*Qab(1,3) + Qab(2,1)*Qab(2,1) + Qab
        (2,3)*Qab(2,3)&
     & + Qab(3,1)*Qab(3,1) + Qab(3,2)*Qab(3,2)
p = p/6.d0
pq = p*p*p - qq*qq
If (pq .ge. 0) then
```

```
        phi = atan(sqrt(pq)/qq)/3.d0
 Else
        phi = 0.d0
 End If
 eigenv(1) = m + 2.d0*sqrt(p)*cos(phi)
 eigenv(2) = m - sqrt(p)*(cos(phi) + sqrt(3.d0)*sin(phi) )
 eigenv(3) = m - sqrt(p)*(cos(phi) - sqrt(3.d0)*sin(phi) )
 s = maxval(eigenv)
End Subroutine
```

C Chapter 5

C.1 MC-NPT simulations results

[a] Simulated in a box with constant shape.

Table C.1: $L/D = 2.50$

P^*	η	S	Phase
7.07	0.467 ± 0.003	0.029 ± 0.007	I
7.46	0.477 ± 0.002	0.034 ± 0.008	I
7.85	0.481 ± 0.004	0.030 ± 0.008	I
8.25	0.575 ± 0.005	0.931 ± 0.004	X
8.64	0.598 ± 0.004	0.979 ± 0.001	X
9.03	0.606 ± 0.002	0.981 ± 0.001	X
9.42	0.607 ± 0.005	0.980 ± 0.001	X
9.82	0.616 ± 0.005	0.983 ± 0.002	X
10.21	0.623 ± 0.004	0.984 ± 0.001	X
10.6	0.631 ± 0.004	0.986 ± 0.001	X

Table C.2: $L/D = 3.00$

P^*	η	S	Phase
7.07	0.460 ± 0.001	0.035 ± 0.017	I
7.3	0.465 ± 0.003	0.034 ± 0.013	I
7.54	0.467 ± 0.003	0.041 ± 0.019	I
7.78	0.542 ± 0.002	0.922 ± 0.002	SmA
8.01	0.550 ± 0.003	0.948 ± 0.003	SmA
8.25	0.556 ± 0.002	0.953 ± 0.005	SmA
8.48	0.590 ± 0.003	0.981 ± 0.001	X
8.72	0.594 ± 0.002	0.982 ± 0.002	X
8.95	0.602 ± 0.003	0.983 ± 0.001	X
9.19	0.605 ± 0.005	0.983 ± 0.001	X
9.42	0.609 ± 0.004	0.985 ± 0.001	X
9.66	0.616 ± 0.003	0.985 ± 0.002	X
9.9	0.621 ± 0.003	0.987 ± 0.001	X
10.13	0.624 ± 0.003	0.986 ± 0.001	X

Table C.3: $L/D = 3.25$

P^*	η	S	Phase
6.89	0.453 ± 0.002	0.028 ± 0.008	I
7.4	0.532 ± 0.002	0.926 ± 0.003	SmA
7.91	0.551 ± 0.002	0.958 ± 0.002	SmA
8.93	0.601 ± 0.005	0.984 ± 0.001	X
9.19	0.604 ± 0.002	0.984 ± 0.001	X
9.44	0.612 ± 0.005	0.987 ± 0.001	X
9.7	0.613 ± 0.002	0.987 ± 0.001	X
9.95	0.621 ± 0.002	0.988 ± 0.002	X
10.21	0.621 ± 0.003	0.987 ± 0.001	X
10.47	0.627 ± 0.003	0.988 ± 0.001	X

Table C.4: $L/D = 3.50$

P^*	η	S	Phase
6.87	0.451 ± 0.002	0.038 ± 0.017	I
7.15	0.524 ± 0.003	0.923 ± 0.004	SmA
7.42	0.537 ± 0.003	0.951 ± 0.005	SmA
7.7	0.544 ± 0.001	0.957 ± 0.001	SmA
7.97	0.550 ± 0.004	0.961 ± 0.001	SmA
8.25	0.559 ± 0.003	0.965 ± 0.001	SmA
8.52	0.588 ± 0.003	0.982 ± 0.001	X
8.8	0.595 ± 0.003	0.984 ± 0.001	X
9.07	0.604 ± 0.003	0.988 ± 0.001	X
9.35	0.607 ± 0.003	0.986 ± 0.001	X
9.62	0.616 ± 0.005	0.988 ± 0.001	X
9.9	0.621 ± 0.004	0.988 ± 0.001	X
10.17	0.625 ± 0.004	0.989 ± 0.001	X
10.45	0.626 ± 0.003	0.989 ± 0.001	X
10.72	0.631 ± 0.003	0.989 ± 0.001	X
11.0	0.637 ± 0.005	0.991 ± 0.001	X

Table C.5: $L/D = 5.00$

P^*	η	S	Phase
0.39	0.139 ± 0.001	0.028 ± 0.004	I
0.79	0.193 ± 0.003	0.029 ± 0.005	I
1.18	0.229 ± 0.004	0.030 ± 0.004	I
1.57	0.256 ± 0.004	0.029 ± 0.002	I
1.96	0.280 ± 0.002	0.034 ± 0.009	I
2.36	0.299 ± 0.003	0.034 ± 0.007	I
2.75	0.317 ± 0.004	0.033 ± 0.004	I
3.14	0.332 ± 0.003	0.032 ± 0.010	I
3.53	0.348 ± 0.002	0.041 ± 0.006	I
3.93	0.359 ± 0.002	0.040 ± 0.029	I
4.32	0.372 ± 0.001	0.045 ± 0.010	I
4.71	0.385 ± 0.002	0.042 ± 0.010	I
5.11	0.397 ± 0.002	0.059 ± 0.011	I
5.5	0.408 ± 0.003	0.044 ± 0.015	I
5.89	0.480 ± 0.004	0.927 ± 0.002	SmA
6.28	0.500 ± 0.002	0.951 ± 0.003	SmA
6.68	0.515 ± 0.002	0.963 ± 0.001	SmA
7.07	0.526 ± 0.001	0.966 ± 0.002	SmA
7.46	0.537 ± 0.003	0.971 ± 0.001	SmA
7.85	0.549 ± 0.002	0.974 ± 0.001	SmA
8.25	0.561 ± 0.004	0.978 ± 0.001	SmA
8.64	0.567 ± 0.002	0.979 ± 0.001	SmA
9.03	0.603 ± 0.004	0.992 ± 0.001	X
9.42	0.608 ± 0.005	0.991 ± 0.001	X
9.82	0.617 ± 0.003	0.992 ± 0.001	X
10.21	0.618 ± 0.003	0.991 ± 0.001	X
10.6	0.633 ± 0.006	0.993 ± 0.001	X
11.39	0.642 ± 0.004	0.995 ± 0.001	X
11.78	0.650 ± 0.001	0.995 ± 0.001	X
12.17	0.655 ± 0.001	0.995 ± 0.001	X
12.57	0.661 ± 0.002	0.996 ± 0.001	X
12.96	0.663 ± 0.003	0.995 ± 0.001	X
13.35	0.673 ± 0.002	0.996 ± 0.001	X
13.74	0.679 ± 0.001	0.996 ± 0.001	X
14.14	0.678 ± 0.002	0.996 ± 0.001	X
14.53	0.686 ± 0.001	0.997 ± 0.001	X
14.92	0.689 ± 0.003	0.997 ± 0.001	X
15.32	0.694 ± 0.002	0.997 ± 0.001	X
15.71	0.697 ± 0.001	0.997 ± 0.001	X

Table C.6: $L/D = 6.00$

P^*	η	S	Phase
4.71	0.381 ± 0.003	0.045 ± 0.030	I
5.65	0.475 ± 0.004	0.949 ± 0.003	SmA
6.13	0.498 ± 0.002	0.965 ± 0.002	SmA
6.6	0.513 ± 0.004	0.970 ± 0.003	SmA
7.07	0.526 ± 0.005	0.975 ± 0.002	SmA
7.54	0.542 ± 0.002	0.979 ± 0.003	SmA
8.01	0.555 ± 0.002	0.981 ± 0.001	SmA
8.48	0.564 ± 0.001	0.983 ± 0.001	SmA
8.95	0.576 ± 0.003	0.986 ± 0.001	SmA
9.42	0.604 ± 0.005	0.993 ± 0.001	X
9.9	0.620 ± 0.004	0.995 ± 0.001	X
10.37	0.627 ± 0.003	0.995 ± 0.001	X
10.84	0.636 ± 0.004	0.996 ± 0.001	X
11.31	0.641 ± 0.003	0.995 ± 0.001	X
11.78	0.652 ± 0.001	0.997 ± 0.001	X
12.25	0.655 ± 0.003	0.996 ± 0.001	X
12.72	0.665 ± 0.003	0.997 ± 0.001	X
13.19	0.670 ± 0.002	0.997 ± 0.001	X
13.67	0.682 ± 0.002	0.999 ± 0.001	X
14.14	0.682 ± 0.003	0.997 ± 0.001	X

Table C.7: $L/D = 6.25$

P^*	η	S	Phase
4.42	0.369 ± 0.002	0.101 ± 0.016	I
4.91	0.434 ± 0.002	0.896 ± 0.009	SmA
5.4	0.464 ± 0.004	0.947 ± 0.002	SmA
5.89	0.487 ± 0.002	0.963 ± 0.003	SmA
6.38	0.503 ± 0.008	0.969 ± 0.002	SmA
6.87	0.523 ± 0.003	0.975 ± 0.002	SmA
7.36	0.537 ± 0.001	0.979 ± 0.001	SmA
7.85	0.547 ± 0.003	0.981 ± 0.002	SmA
8.34	0.562 ± 0.004	0.984 ± 0.001	SmA
8.84	0.572 ± 0.005	0.986 ± 0.002	SmA
9.33	0.583 ± 0.001	0.987 ± 0.001	SmA
9.82	0.615 ± 0.005	0.995 ± 0.001	X
10.31	0.627 ± 0.002	0.995 ± 0.001	X
10.8	0.638 ± 0.004	0.996 ± 0.001	X
11.29	0.642 ± 0.006	0.996 ± 0.001	X
11.78	0.649 ± 0.004	0.996 ± 0.001	X
12.27	0.657 ± 0.002	0.997 ± 0.001	X
12.76	0.664 ± 0.003	0.997 ± 0.001	X

Table C.8: $L/D = 6.50$

P^*	η	S	Phase
4.19	0.363 ± 0.001	0.068 ± 0.004	I
4.29	0.366 ± 0.003	0.058 ± 0.018	I
4.39	0.370 ± 0.002	0.086 ± 0.033	I
4.49	0.395 ± 0.005	0.709 ± 0.029	N
4.59	0.409 ± 0.004	0.868 ± 0.018	SmA
4.7	0.415 ± 0.002	0.885 ± 0.012	SmA
5.11	0.447 ± 0.002	0.921 ± 0.004	SmA
5.62	0.473 ± 0.004	0.959 ± 0.003	SmA
6.13	0.497 ± 0.003	0.969 ± 0.002	SmA
6.64	0.513 ± 0.001	0.972 ± 0.004	SmA
7.15	0.529 ± 0.004	0.977 ± 0.004	SmA
7.66	0.543 ± 0.004	0.981 ± 0.003	SmA
8.17	0.556 ± 0.002	0.984 ± 0.001	SmA
8.68	0.570 ± 0.001	0.986 ± 0.001	SmA
9.19	0.608 ± 0.005	0.995 ± 0.001	X
9.7	0.624 ± 0.003	0.998 ± 0.001	X
10.21	0.623 ± 0.003	0.995 ± 0.001	X
10.72	0.642 ± 0.001	0.998 ± 0.001	X
11.23	0.642 ± 0.002	0.996 ± 0.001	X
11.74	0.654 ± 0.001	0.997 ± 0.001	X
12.25	0.655 ± 0.003	0.997 ± 0.001	X
12.76	0.666 ± 0.001	0.997 ± 0.001	X

Table C.9: $L/D = 7.00$

P^*	η	S	Phase
3.96	0.354 ± 0.002	0.097 ± 0.031	I
4.07	0.384 ± 0.003	0.823 ± 0.015	N
4.18	0.392 ± 0.003	0.863 ± 0.016	N
4.29	0.397 ± 0.001	0.876 ± 0.009	N
4.4	0.401 ± 0.002	0.880 ± 0.009	N
5.5	0.471 ± 0.003	0.963 ± 0.003	SmA
6.05	0.495 ± 0.004	0.971 ± 0.001	SmA
6.6	0.517 ± 0.002	0.977 ± 0.002	SmA
7.15	0.532 ± 0.004	0.981 ± 0.001	SmA
7.7	0.546 ± 0.002	0.984 ± 0.001	SmA
8.25	0.560 ± 0.003	0.986 ± 0.002	SmA
8.8	0.607 ± 0.004	0.998 ± 0.001	X
9.35	0.619 ± 0.002	0.998 ± 0.001	X
9.9	0.621 ± 0.003	0.996 ± 0.001	X
10.45	0.629 ± 0.004	0.996 ± 0.001	X
11.0	0.637 ± 0.001	0.996 ± 0.001	X

Table C.10: $L/D = 7.50$

P^*	η	S	Phase
2.95	0.305 ± 0.002	0.050 ± 0.030	I
3.53	0.335 ± 0.003	0.076 ± 0.026	I
4.12	0.390 ± 0.002	0.898 ± 0.013	N
5.3	0.464 ± 0.001	0.965 ± 0.002	SmA
5.89	0.491 ± 0.003	0.973 ± 0.002	SmA
6.48	0.509 ± 0.005	0.978 ± 0.002	SmA
7.07	0.530 ± 0.003	0.982 ± 0.001	SmA

Table C.11: $L/D = 10.00$

P^*	η	S	Phase
0.79	0.165 ± 0.001	0.024 ± 0.008	I
1.57	0.224 ± 0.001	0.043 ± 0.013	I
2.36	0.299 ± 0.042	0.804 ± 0.113	N
3.93	0.386 ± 0.002	0.953 ± 0.003	N
5.5	0.475 ± 0.001	0.982 ± 0.002	SmA
6.28	0.504 ± 0.002	0.986 ± 0.001	SmA
7.07	0.528 ± 0.002	0.989 ± 0.001	SmA
7.85	0.548 ± 0.002	0.991 ± 0.001	SmA
8.64	0.570 ± 0.002	0.993 ± 0.001	SmA
9.42	0.611 ± 0.001	0.998 ± 0.001	X
10.21	0.628 ± 0.002	0.998 ± 0.001	X
11.0	0.642 ± 0.001	0.998 ± 0.001	X
11.78	0.652 ± 0.003	0.999 ± 0.001	X

Table C.12: $L/D = 0.05$

P^*	η	S	Phase
0.2	0.066 ± 0.001	0.047 ± 0.008	I
0.39	0.096 ± 0.001	0.061 ± 0.002	I
0.59	0.121 ± 0.001	0.086 ± 0.027	I
0.98	0.187 ± 0.001	0.830 ± 0.019	N
1.18	0.212 ± 0.003	0.883 ± 0.014	N
1.37	0.231 ± 0.004	0.896 ± 0.031	N
1.57	0.251 ± 0.003	0.925 ± 0.008	N
1.77	0.267 ± 0.001	0.935 ± 0.009	N
1.96	0.283 ± 0.005	0.955 ± 0.008	N
2.16	0.296 ± 0.004	0.959 ± 0.005	N
2.36	0.310 ± 0.003	0.963 ± 0.009	N
2.55	0.322 ± 0.001	0.966 ± 0.006	N
2.75	0.335 ± 0.003	0.976 ± 0.005	N
2.95	0.346 ± 0.004	0.978 ± 0.006	N
3.14	0.357 ± 0.005	0.977 ± 0.007	N
3.34	0.369 ± 0.005	0.983 ± 0.004	N
3.53	0.376 ± 0.003	0.984 ± 0.002	N
3.73	0.388 ± 0.005	0.986 ± 0.001	N
3.93	0.396 ± 0.007	0.987 ± 0.003	N
4.32	0.445 ± 0.007	0.990 ± 0.001	C
4.52	0.455 ± 0.008	0.991 ± 0.001	C
4.71	0.467 ± 0.006	0.992 ± 0.001	C
4.91	0.473 ± 0.006	0.993 ± 0.001	C
5.11	0.481 ± 0.007	0.992 ± 0.002	C
5.3	0.489 ± 0.006	0.992 ± 0.002	C
5.5	0.502 ± 0.006	0.994 ± 0.001	C
5.69	0.511 ± 0.007	0.994 ± 0.001	C
7.85	0.575 ± 0.001	0.999 ± 0.001	C

Table C.13: $L/D = 0.10$

P^*	η	S	Phase
0.39	0.118 ± 0.001	0.043 ± 0.005	I
0.79	0.165 ± 0.002	0.053 ± 0.010	I
1.18	0.200 ± 0.001	0.058 ± 0.013	I
1.57	0.231 ± 0.003	0.071 ± 0.019	I
1.96	0.257 ± 0.004	0.078 ± 0.034	I
2.75	0.321 ± 0.002	0.742 ± 0.035	N
3.14	0.349 ± 0.001	0.887 ± 0.016	N
3.53	0.370 ± 0.004	0.912 ± 0.013	N
3.93	0.388 ± 0.007	0.929 ± 0.009	N
4.32	0.413 ± 0.001	0.956 ± 0.006	N
4.71	0.461 ± 0.004	0.972 ± 0.002	C
5.11	0.478 ± 0.003	0.972 ± 0.001	C
5.5	0.504 ± 0.007	0.981 ± 0.003	C
5.89	0.516 ± 0.004	0.982 ± 0.002	C
6.28	0.530 ± 0.001	0.983 ± 0.002	C
6.68	0.546 ± 0.003	0.982 ± 0.002	C
7.07	0.559 ± 0.004	0.984 ± 0.003	C
7.46	0.569 ± 0.005	0.986 ± 0.001	C
7.85	0.580 ± 0.003	0.987 ± 0.001	C

Table C.14: $L/D = 0.11$

P^*	η	S	Phase
2.16	0.275 ± 0.001	0.081 ± 0.017	I
2.59	0.302 ± 0.005	0.091 ± 0.043	Cub
3.02	0.327 ± 0.002	0.094 ± 0.041	Cub
3.46	0.354 ± 0.004	0.120 ± 0.028	Cub
4.75	0.462 ± 0.007	0.962 ± 0.008	C
5.18	0.489 ± 0.006	0.975 ± 0.003	C
5.62	0.506 ± 0.003	0.956 ± 0.002	C
6.05	0.527 ± 0.006	0.977 ± 0.002	C
6.48	0.543 ± 0.003	0.981 ± 0.002	C
6.91	0.556 ± 0.004	0.981 ± 0.002	C
7.34	0.566 ± 0.005	0.984 ± 0.003	C
7.78	0.578 ± 0.004	0.985 ± 0.002	C
8.21	0.587 ± 0.001	0.987 ± 0.001	C
8.64	0.597 ± 0.005	0.988 ± 0.001	C

Table C.15: $L/D = 0.12$

P^*	η	S	Phase
1.88	0.259 ± 0.005	0.072 ± 0.014	I
2.36	0.287 ± 0.002	0.081 ± 0.024	I
2.83	0.317 ± 0.003	0.088 ± 0.059	Cub
3.3	0.344 ± 0.003	0.112 ± 0.060	Cub
3.77	0.369 ± 0.006	0.114 ± 0.054	Cub
4.24	0.395 ± 0.005	0.092 ± 0.057	Cub
4.71	0.457 ± 0.005	0.950 ± 0.008	C
5.18	0.486 ± 0.004	0.969 ± 0.002	C
5.65	0.510 ± 0.003	0.978 ± 0.002	C
6.13	0.524 ± 0.007	0.982 ± 0.002	C
6.6	0.543 ± 0.005	0.978 ± 0.003	C
7.07	0.558 ± 0.004	0.981 ± 0.002	C
7.54	0.570 ± 0.002	0.966 ± 0.001	C
8.01	0.582 ± 0.007	0.984 ± 0.003	C
8.48	0.593 ± 0.003	0.986 ± 0.001	C
8.95	0.592 ± 0.004	0.964 ± 0.003	C

Table C.16: $L/D = 0.12$

P^*	η	S	Phase
4.52	0.441 ± 0.005	0.937 ± 0.014	C
4.62	0.460 ± 0.006	0.943 ± 0.010	C

Table C.17: $L/D = 0.15$

P^*	η	S	Phase
2.95	0.324 ± 0.002	0.054 ± 0.028	I
3.53	0.353 ± 0.003	0.058 ± 0.043	Cub
4.12	0.382 ± 0.003	0.068 ± 0.030	Cub
4.71	0.412 ± 0.003	0.080 ± 0.019	Cub
4.83	0.470 ± 0.004	0.888 ± 0.002	C
4.95	0.476 ± 0.006	0.905 ± 0.002	C
5.07	0.483 ± 0.002	0.918 ± 0.005	C
5.18	0.489 ± 0.005	0.930 ± 0.005	C
5.3	0.497 ± 0.002	0.938 ± 0.005	C
5.89	0.516 ± 0.005	0.946 ± 0.002	C

Table C.18: $L/D = 0.2$

P^*	η	S	Phase
0.79	0.194 ± 0.002	0.042 ± 0.004	I
1.57	0.260 ± 0.001	0.043 ± 0.006	I
2.36	0.304 ± 0.002	0.048 ± 0.011	I
3.14	0.342 ± 0.003	0.052 ± 0.011	I
3.93	0.372 ± 0.004	0.051 ± 0.012	I
4.71	0.405 ± 0.004	0.058 ± 0.028	Cub
5.5	0.434 ± 0.004	0.048 ± 0.016	Cub
5.65	0.508 ± 0.003	0.898 ± 0.006	C
5.81	0.512 ± 0.004	0.861 ± 0.007	C
5.97	0.515 ± 0.001	0.886 ± 0.008	C
6.13	0.525 ± 0.002	0.924 ± 0.006	C
6.28	0.532 ± 0.003	0.930 ± 0.006	C
7.07	0.551 ± 0.004	0.927 ± 0.007	C
7.85	0.560 ± 0.001	0.946 ± 0.004	C
8.64	0.583 ± 0.004	0.955 ± 0.006	C

Table C.19: $L/D = 0.25$

P^*	η	S	Phase
4.91	0.411 ± 0.004	0.032 ± 0.008	I
5.89	0.444 ± 0.004	0.042 ± 0.010	Cub
6.87	0.537 ± 0.002	0.838 ± 0.007	C
7.85	0.563 ± 0.003	0.908 ± 0.008	C
8.84	0.591 ± 0.002	0.959 ± 0.004	C

Table C.20: $L/D = 0.30$

P^*	η	S	Phase
5.89	0.443 ± 0.002	0.051 ± 0.024	I
6.48	0.460 ± 0.003	0.041 ± 0.016	Cub
7.07	0.536 ± 0.004	0.770 ± 0.006	C
7.66	0.561 ± 0.003	0.865 ± 0.003	C
8.25	0.580 ± 0.003	0.910 ± 0.003	C
8.84	0.606 ± 0.005	0.941 ± 0.003	C
9.42	0.589 ± 0.002	0.915 ± 0.004	C

Table C.21: $L/D = 0.35$

P^*	η	S	Phase
4.12	0.394 ± 0.003	0.038 ± 0.010	I
5.5	0.434 ± 0.006	0.044 ± 0.020	I
6.87	0.469 ± 0.004	0.041 ± 0.009	I
8.25	0.567 ± 0.003	0.822 ± 0.011	C

Table C.22: $L/D = 0.50$

P^*	η	S	Phase
1.96	0.320 ± 0.002	0.035 ± 0.002	I
3.93	0.398 ± 0.001	0.034 ± 0.005	I
5.89	0.451 ± 0.003	0.037 ± 0.006	I
7.85	0.492 ± 0.003	0.039 ± 0.015	I
8.25	0.573 ± 0.003	0.855 ± 0.007	C
8.64	0.594 ± 0.003	0.926 ± 0.005	C

C.2 Codes

The program requires an input file named *hc_input.sci*. An example is given below.

```
Number_of_particles=
Molecule_Type=      1
Cold_Configuration=     .true. ! If false it will require an initial
   configuration file
Production_run=      .false.
Reduced_pressure=     p_red ! p_red = p*D^3/kT
Number_of_steps=
Number_of_cycles_per_step=
Print_every_x_steps=    x ! Print hc.dat every x steps
Max_rotational_displacement_rad= 0.05d0 ! Maximum rotational displacement
    in radians
Max_translational_displacement=  0.05d0
Max_volume_scaling=     0.001d0
Adjust_drmax_every_x_step=   1000 ! Adjust maximum translational
   displacement
Adjust_dvmax_every_x_step=   1000 ! Adjust maximum volumetric
   displacement
Acceptance_Ratio=     0.4
Cylinder_D_and_L=     diameter length
Inputs_to_generate_initial_configuration_only_if_cold
Cubic_box_1_or_fcc_2_or_packed_3= 1 ! This example will generate a cubic
   box
Initial_Packing_Fraction=   0.01d0
Initial_Quaternion_axis=   0d0 1.d0 0.d0
Rotation_around_axis_degrees=  45.d0
```

Listing C.1: Example of the input file hc_input.sci

The main code needs three modules to run properly:

- module_global_global_variables.f90: contains the global variables
- module_initial_configuration.f90: contains the routines to generate the initial configuration
- module_mc.f90: contains the main routines
  The program will generate three output files:
  - conf.xyz: contains the trajectory. The file is organized as follows: the first line is the total number of particles, second line is the x, y and z dimensions of the simulation box, and from the third line on, we have the positions and orientations of each particle (type of the molecule, x position, y position, z position, scalar part of the quaternion, x component of the quaternion, y component of the quaternion, z component of the quaternion, $D/2$, $D/2$, L).
  - current_conf.xyz: contains the current configuration of the particles (same organization of file 'conf.xyz'.)

- hc.dat: The file is organized as follows: step, acceptance ratio (translational and rotational), acceptance ratio (volume moves), maximum translational displacement, maximum angular displacement, maximum change in volume, reduced pressure, packing fraction, nematic order parameter, x component of the phase director, y component of the phase director, z component of the phase director.

C.2.1 NPT Monte Carlo code for hard cylinders (main_hc_mc_npt.f90)

```
!************************************************************************
!************************************************************************
! NPT Monte Carlo simulation of hard cylinders
!************************************************************************
! Developer: Joyce Tavares Lopes
! Supervisor: Dr. Luis Fernando Mercier Franco
! School of chemical engineering (FEQ) - UNICAMP
!************************************************************************
! This code was developed during a research period at
! Universita Ca' Foscari Venezia under the supervision of
! Professor Achille Giacometi. (SEP 2019/FEB 2020)
!************************************************************************
Program HC_MC_NPT
 use monte_carlo
 use initial_configuration
 use global_variables
 use order_parameters
 Implicit None

 !-----------------------------------------------------------------------
 ! Local Variables
 !-----------------------------------------------------------------------
 ! Input for pseudo-random number generator
 !-----------------------------------------------------------------------
 Integer,Dimension(:),Allocatable :: vseed
 Integer :: seed_size,seed = 349766914
 Real(8) :: rnum
 !-----------------------------------------------------------------------
 ! Properties
 !-----------------------------------------------------------------------
 Real(8) :: p,vnew,eta0
 !-----------------------------------------------------------------------
 ! Characterization of Particle
 !-----------------------------------------------------------------------
 Real(8) :: d3
 !-----------------------------------------------------------------------
 ! Position, orientation and simulation box variables
 !-----------------------------------------------------------------------
```

```
Real(8), Allocatable :: rnew(:,:)
Real(8) :: boxlnew(3),rij(3),ri(3),ei(3)
Real(8) :: quat0_axis(3),quat0_angle
Logical :: cold_conf,production
Integer :: structure
Integer :: i,j,k,jlayer
Character :: get*100,moltype*1,outfile*100
Character :: inputfile1*100,inputfile2*100,outfile2*100
Integer :: moves_re,move_v
Integer :: step2print,stepdrmax,stepdvmax
Real(8) :: acc_mov,acc_mov_v,acc_ratio
Real(8), dimension(0:3) :: qi
Real(8) :: deltap,deltav,deltah,new_pot,old_pot
Logical :: overlap
Real :: start,finish,startstep,finishstep
Real(8),dimension(0:3) :: qnew
Real(8) :: s2,evec(3)
Real :: attempts_re
Character :: file0*1

!========================================================================
! Initialize Random_Number (Fortran intrinsic function)
!========================================================================

Call Random_seed(size=seed_size)
Allocate(vseed(seed_size))
vseed(:) = seed
Call Random_seed(put=vseed)

Call cpu_time(start)
!========================================================================
!Simulation Input File
!========================================================================
!Monte Carlo simulation and Potential parameters
Open(101,File='hc_input.sci')
Read(101,*) get,n
Read(101,*) get,moltype
Read(101,*) get,cold_conf
Read(101,*) get,production
Read(101,*) get,p
Read(101,*) get,nsteps
Read(101,*) get,ncycles
Read(101,*) get,step2print
Read(101,*) get,max_angle
Read(101,*) get,max_r
Read(101,*) get,max_v
Read(101,*) get,stepdrmax
Read(101,*) get,stepdvmax
Read(101,*) get,acc_ratio
! Cylinder parameters
Read(101,*) get,d,l
!========================================================================
!
!========================================================================
aspect_ratio = l/d
v_particle = 0.25d0*pi*d*d*l
halfl = 0.5d0*l
halfd = 0.5d0*d
!Variable 'space' is only used in packed box initial configuration
spacex = 0.d0
spacez = 0.d0
!========================================================================
! Initial Configuration - Warm or Cold
!========================================================================
Select Case (cold_conf)
Case(.true.)
  Read(101,*) get
  Read(101,*) get,structure
  Read(101,*) get,eta0
  Read(101,*) get,quat0_axis(1),quat0_axis(2),quat0_axis(3)
  Read(101,*) get,quat0_angle
  Select Case(structure)
  Case(1)
    Allocate(r(3,N),rnew(3,N),q(0:3,N),e(3,N))
    call cubic(eta0,quat0_axis,quat0_angle)
  Case(2)
    Allocate(r(3,N),rnew(3,N),q(0:3,N),e(3,N))
    call fcc(eta0,quat0_axis,quat0_angle)
  Case(3)
    call packed_box(eta0,quat0_axis,quat0_angle)
    Allocate(rnew(3,N))
  End Select
Case(.false.)
  inputfile2 = 'current_conf.xyz '
  Open(102,FILE=trim(adjustl(inputfile2)))
  Read(102,*) n
  Allocate(r(3,N),rnew(3,N),q(0:3,N),e(3,N))
  Read(102,*) boxl(1),boxl(2),boxl(3)
  Do 99 i=1,n
  Read(102,*) moltype,r(1,i),r(2,i),r(3,i),&
    &q(0,i),q(1,i),q(2,i),q(3,i),halfd,halfd,l
  99 End Do
  Close(102)
End Select
Close(101)
```

```
!======================================================================
! Simulation Output Files
!======================================================================
! Formats
1 Format(4x,A4,2x,A7,3x,a7,4x,a6,3x,a7,3x,a7,&
  &3x,a13,3x,a11,4x,a4,3x,a7,6x,a4,10x,a2,2x,a2)
2 Format(I7,f10.5,f10.5,f10.5,f10.5,f10.5,2x,&
  &f10.5,4x,f10.5,2x,f10.5,f10.3,f10.3,f10.3)
3 Format(A13,f10.5)
4 Format(A13,I5)
!----------------------------------------------------------------------
! Configuration File - Multiple Steps
!----------------------------------------------------------------------
!----------------------------------------------------------------------
Open(200,File='conf.xyz')
Write(200,*) n
Write(200,*) boxl(1),boxl(2),boxl(3)
Do i=1,n
Write(200,*) moltype,r(1,i),r(2,i),r(3,i),&
  &q(0,i),q(1,i),q(2,i),q(3,i),halfd,halfd,l
End Do
Call flush(200)
!----------------------------------------------------------------------
! Current Configuration File
!----------------------------------------------------------------------
Open(203,File='current_displacements')
Open(202,File='current_conf.xyz')
!----------------------------------------------------------------------
! Properties File
!----------------------------------------------------------------------

Write(outfile2,'("PropP",f3.1,".dat")') p
Open(201,File='hc.dat',status='unknown')
!----------------------------------------------------------------------
! Simulation Parameters File
!----------------------------------------------------------------------
Open(204,file = 'parameters.dat')

!======================================================================
! Initialization
!======================================================================

v = boxl(1)*boxl(2)*boxl(3)
rho = dble(n)/v
d3 = d*d*d
max_v = max_v*d3
rcut = 4.d0*d
max_r = max_r*d
```

```
!=======================================================================
! Calculate orientations from initial quaternion rotation
!=======================================================================

Do 100 i=1,n !Loop over particles
Call quat_to_ori(efixed,q(:,i),e(:,i))
100 End Do

!=======================================================================
! Check Overlap in Initial Configuration
!=======================================================================

overlap = .false.
Do i=1,n-1
ei(:) = e(:,i)
ri(:) = r(:,i)
qi(:) = q(:,i)
Call check_overlap_cylinder(boxl,i,qi,ei,ri,r,overlap)
If (overlap) then
  Print*, 'Overlap detected in initial configuration!'
  STOP
End If
End Do

!=======================================================================
! Print Initial Conditions and Parameters
!=======================================================================

Call orderparameter(s2,evec)

Write(*,4) 'N =',N
Write(*,3) 'L/D =',aspect_ratio
Write(*,3) 'P* =',p
Write(*,3) 'eta_i =',rho*v_particle
Write(*,3) 'rho =',rho
Write(*,3) '<P2> =',s2

Write(204,4) 'N =',N
Write(204,3) 'L/D =',aspect_ratio
Write(204,3) 'P* =',p
Write(204,3) 'eta_i =',rho*v_particle
Write(204,3) 'rho =',rho
Write(204,3) '<P2> =',s2
flush(204)

step = 0
acc_mov = 0.d0
```

```
acc_mov_v = 0.d0
Write(201,2) step,acc_mov,acc_mov_v,max_r/d,max_angle,max_V/d3,&
  &p,rho*v_particle,s2,evec(1),evec(2),evec(3)
Call flush(201)


!========================================================================
! Start Trial moves
!========================================================================

Do 103 step=1,nsteps !Loop over steps
overlap = .false.
attempts_re = 0
moves_re = 0
move_v = 0
acc_mov = 0
Do 104 cyclei=1,ncycles !Loop over cycles
deltah = 0.d0
Call random_number(rnum)
i = Idint(rnum*dble((n+1))) + 1

!------------------------------------------------------------------------
If (i .le. n) then
  attempts_re = attempts_re + 1
  !----------------------------------------------------------------------
  ! Translational and Rotational moves
  !----------------------------------------------------------------------
  ! Translation move for particle i

  Call new_position(i,boxl,ri)

  ! Rotational move for particle i

  ! Randomly rotate old quaternion
  Call random_rotate_quat(q(:,i),qi)
  ! New orientation after rotation
  Call quat_to_ori(efixed,qi,ei)

  ! Check Overlap
  Call check_overlap_cylinder(boxl,i,qi,ei,ri,r,overlap)



  If (.not. overlap) then
    q(:,i) = qi(:)
    r(:,i) = ri(:)
    e(:,i) = ei(:)
    moves_re = moves_re + 1
  End if
  !----------------------------------------------------------------------
Else
  !----------------------------------------------------------------------
  ! Volume Move
  !----------------------------------------------------------------------

  Call new_volume(rnew,boxlnew,vnew)
  deltav = vnew - v
  !Check Overlap after volume scaling
  Do i=1,n-1
  ei = e(:,i)
  ri = rnew(:,i)
  qi = q(:,i)
  Call check_overlap_cylinder(boxlnew,i,qi,ei,ri,rnew,overlap)
  If (overlap) exit
  End Do
  If (.not. overlap) then
    deltah = p*deltav/d3
    deltah = deltah - (dble(n)+1d0)*dlog(vnew/v)
    If (deltah .lt. 0.0) then
      r(:,:) = rnew(:,:)
      rho = dble(n)/vnew
      boxl(:) = boxlnew(:)
      v = boxl(1)*boxl(2)*boxl(3)
      move_v = 1 + move_v
    Else If ((deltah) .lt. 75) then
      Call random_number(rnum)
      If (dexp(-deltah) .gt. rnum) then
        r(:,:) = rnew(:,:)
        rho = dble(n)/vnew
        boxl(:) = boxlnew(:)
        v = boxl(1)*boxl(2)*boxl(3)
        move_v = 1 + move_v
      End If
    End If
  End If
End If
104 End Do


acc_mov = dble(moves_re)/dble(attempts_re)
if (attempts_re .lt. ncycles) &
  & acc_mov_v = dble(move_v)/dble(ncycles - attempts_re)

!========================================================================
! Write properties and configuration every step2print steps
!========================================================================
if (mod(step,step2print) == 0) then
  !--------------------------------------------------------------------
  ! Properties to print
  !--------------------------------------------------------------------
  eta = rho*v_particle
  !--------------------------------------------------------------------
  ! Order parameters
  !--------------------------------------------------------------------

  Call orderparameter(s2,evec)
  !--------------------------------------------------------------------
  ! Write Properties and Configuration Files
  !--------------------------------------------------------------------
  ! Properties File
  Write(201,2) step,acc_mov,acc_mov_v,max_r/d,&
    &max_angle,max_V/d3,&
    &p,eta,s2,evec(1),evec(2),evec(3)
  Call flush(201)
  ! Last Configuration File (position and quaternions)
  Write(203,*) 'Max Translational displacement',max_r
  Write(203,*) 'Max Rotational displacement',max_angle
  Write(203,*) 'Max Volume',max_v
  Rewind(203)
  Write(202,*) n
  Write(202,*) boxl(1),boxl(2),boxl(3)
  Do 200 i=1,n
  Write(202,*) moltype,r(1,i),r(2,i),r(3,i),q(0,i),&
    &q(1,i),q(2,i),q(3,i),halfd,halfd,l
  200 End DO
  Call flush(202)
  Call flush(203)
  Rewind(202)
  Rewind(203)
  ! If (Mod(step,10*step2print) == 0) then
  ! Configuration File (position and quaternions)
  Write(200,*) n
  Write(200,*) boxl(1),boxl(2),boxl(3)
  Do 205 i=1,n
  Write(200,*) moltype,r(1,i),r(2,i),r(3,i),q(0,i),&
    &q(1,i),q(2,i),q(3,i),halfd,halfd,l

  205 End DO
  Call flush(200)
  ! End If

End If
!======================================================================
! Adjust Maximum Displacement
!=====================================================================

! Adjust max_r every stepdrmax steps
If (mod(step,stepdrmax) == 0) then
  If (acc_mov .gt. acc_ratio) then
    max_r = 1.05*max_r
    max_angle = 1.05*max_angle
  Else
    max_r = 0.95*max_r
    max_angle = 0.95*max_angle
  End If
End If

! Adjust max_v every stepdvmax steps
If (attempts_re .lt. ncycles) then
  If (mod(step,stepdvmax) == 0) then
    If (acc_mov_v .gt. acc_ratio) then
      max_v = 1.05*max_v
    Else
      max_v = 0.95*max_v
    End If
  End If
End If
103 End Do



Call cpu_time(finish)

Call orderparameter(s2,evec)

Write(*,*) '======= Final ======='
Write(*,3) 'eta_f =',eta
Write(*,3) '<P2> =',s2
Write(*,3,advance = "no") 'Run Time =',(finish-start)/60.d0
Write(*,*) 'min'

Write(204,*) '======= Final ======='
Write(204,3) 'eta_f =',eta
Write(204,3) '<P2> =',s2
Write(204,3,advance = "no") 'Run Time =',(finish-start)/3600.d0
Write(204,*) 'h'


Close(200)
Close(201)
Close(202)
```

```
 Close(203)
 Close(204)
 End
```

C.2.2 Module to set global variables (module_global_global_variables.f90)

```
module global_variables

implicit none

!=========================================================================
! Variables Declaration
!=========================================================================
Integer,Public :: n
Integer,Public :: n_ave
Integer,Public :: nsmectic_layers,nlayers
!-------------------------------------------------------------------------
! Properties
!-------------------------------------------------------------------------
 Real(8) :: rho,eta
!-------------------------------------------------------------------------
! Simulation Values
!-------------------------------------------------------------------------
 Integer,Public :: nsteps,step,ncycles,cyclei
!-------------------------------------------------------------------------
! Characterization of Particle
!-------------------------------------------------------------------------
 Real(8),Public :: d,dsq,l,lsq,aspect_ratio
 Real(8),Public :: halfl,halfd,v_particle
 Real(8),Public :: spacex,spacez
 Real(8),Public :: dist_cutoff
 Real(8),Public :: DL_spacing,hlayer
!-------------------------------------------------------------------------
! Nematic director values
!-------------------------------------------------------------------------
 Real(8),Public :: EV1(3)
 Real(8),Public :: p2, pd(3) ! Nematic order parameter and Phase director
!-------------------------------------------------------------------------
! Position, orientation and simulation box variables
!-------------------------------------------------------------------------
 Real(8),Allocatable,Public :: r(:,:),q(:,:),e(:,:)
 Real(8),Public :: v,boxl(3)
 Real(8),Public :: Lbox
 Real(8),Parameter,Public :: efixed(3) = (/0.d0,0.d0,1.d0/)
 Real(8),Parameter,Public :: eyfixed(3) = (/0.d0,1d0,0d0/)
 Real(8),Parameter,Public :: exfixed(3) = (/1.d0,0d0,0d0/)
!-------------------------------------------------------------------------
! Potential - Patches Paramters - Helix parameter
```

```
!-----------------------------------------------------------------------
 Real(8),Public :: t,eps,lambda,rcut,rcutsq,swrange
 Integer,Public :: n_patches
 Real(8),Public :: pitch,l_h,r_h !helices radius
 Real(8),Public :: d_bead,r_bead,fusion,sqd_bead
 Integer,Public :: n_beads,np
 !Reduced temperature
 Real(8),Public :: reduced_temperature !kbT/eps
!-----------------------------------------------------------------------
! Yukawa Parameters
!-----------------------------------------------------------------------
 Real(8),Public :: Z1,Z2 ! charge
 Real(8),Public :: eps_elec !dielectric constant of solvent
 Real(8),Public :: kd !inverse Debye screening length [cm^-1]
 Real(8),Public :: Ic,Is !ionic strength
 Real(8),Public :: lb !Bjerrum length (LB=e^2/epsilon*KB*T) [cm]
 !Other Reduced temperature
 Real(8),Public :: T_star !diameter/Bjerrum length
 Real(8),Public :: P_star
 Real(8),Public :: zeta !independent reduced screening parameter
 !Renormalization
 Real(8),Public :: Z_big
 Real(8),Public :: yuk_cte,y_energy
 Real(8),Public :: d_cm,d3_cm3 !Diameter in cm
 Real(8),Public :: v_cm3
!-----------------------------------------------------------------------
! Post-processing variables
!-----------------------------------------------------------------------
 Real(8),Public :: dspacing(20),smectic_R(20),smectic_I(20),smectic(20)
 Real(8),Public :: nz(20),nz_ave(20),sinterm(20),costerm(20),Psi6l_ave
    (20),zl(20)
 Real(8),Public :: dsp,theta,smR,smI
 ! RDF
 Real(8),Public :: delr,delr_par,delr_per
 Integer,Public :: nbins
 Real(8),Public,allocatable :: hist(:), hist_per(:), hist_par(:), gr(:),
    gr_par(:), gr_per(:)
!-----------------------------------------------------------------------
! Moves
!-----------------------------------------------------------------------
 Real(8),Public :: max_angle,max_r,max_v
!-----------------------------------------------------------------------
! Parameters
!-----------------------------------------------------------------------
 Real(8),Parameter,Public :: na = 6.0221409d23
 Real(8),Parameter,Public :: pi = 4.d0*datan(1.d0)
 Real(8),Parameter,Public :: kb = 1.38064852d-23
 Real(8),Parameter,Public :: kb_cgs= 1.38d-16 ! [erg/K]
 Real(8),Parameter,Public :: e_charge = 4.803d-10 !electronic charge [cm
    ^3/2*g^1/2*sec^-1]
!-------------------------------------------------------------------------
! Unit Conversion
!-------------------------------------------------------------------------
 Real(8),Parameter,Public :: mpa2pa = 1d6
 Real(8),Parameter,Public :: m2ang = 1d10
 Real(8),Parameter,Public :: ang3tocm3 = 1d-24
 Real(8),Parameter,Public :: bartoba = 1d6


end module
```

C.2.3 Module to generate initial configurations (module_initial_configuration.f90)

```
!**********************************************************************
! Module to generate initial configurations
!**********************************************************************
! Developer: Joyce Tavares Lopes
! Supervisor: Dr. Luis Fernando Mercier Franco
! School of chemical engineering (FEQ) - UNICAMP
!**********************************************************************
! This code was developed during a research period at
! Universita Ca' Foscari Venezia under the supervision of
! Professor Achille Giacometi.
!**********************************************************************
module initial_configuration
use global_variables
use monte_carlo
 implicit none
contains

Subroutine cubic(eta0,quat0_axis,quat0_angle)
       Implicit None
       Integer :: i,j,k,id,nl
       Real(8) :: a,eta0,quat0_axis(3),quat0_angle
       Real(8),Allocatable :: rx(:),ry(:),rz(:)
       Character :: moltype*1

       ! Convert to radians
       quat0_angle = quat0_angle*pi/180.d0
       quat0_angle = quat0_angle/2.d0
       q(0,:) = dcos(quat0_angle)
       q(1,:) = dsin(quat0_angle)*quat0_axis(1)
       q(2,:) = dsin(quat0_angle)*quat0_axis(2)
       q(3,:) = dsin(quat0_angle)*quat0_axis(3)
! Calculation of Volume
       V = N*v_particle/eta0 !^3Angstrom
```

```
        boxl(:) = V**(1.d0/3.d0) !Box length
        nl = nint(dble(n)**(1.d0/3.d0))
        a = boxl(1)/nl
! Output File

! Start positioning particles
        id = 1
        Do 101 i = 1,nl
               Do 102 j = 1,nl
                      Do 103 k = 1,nl
                      ! Particles on the right vertice of unit cell
                             r(1,id) = (dble(i)-1.d0)*a - 0.5*boxl(1)
                             r(2,id) = (dble(j)-1.d0)*a - 0.5*boxl(2)
                             r(3,id) = (dble(k)-1.d0)*a - 0.5*boxl(3)
                             id = id + 1
                  103 End DO
           102 End Do
    101 End Do
End

Subroutine fcc(eta0,quat0_axis,quat0_angle)
        Implicit None
        Integer :: i,j,k,uc,id
        Real(8) :: lx,ly,lz,ucl,lbox
        Real(8) :: quat0_axis(3),quat0_angle,quatw,quatx,quaty,quatz
        Real(8) :: eta0
        Character :: get*100,moltype*1

        ! Convert to radians
        quat0_angle = quat0_angle*pi/180.d0
        quat0_angle = quat0_angle/2.d0
        q(0,:) = dcos(quat0_angle)
        q(1,:) = dsin(quat0_angle)*quat0_axis(1)
        q(2,:) = dsin(quat0_angle)*quat0_axis(2)
        q(3,:) = dsin(quat0_angle)*quat0_axis(3)


! Calculation of Volume
        V = N*v_particle/eta0 !³Angstrom
        lbox = V**(1.d0/3.d0) ! Cubic Box length
        uc = nint((dble(n)*0.25)**(1.d0/3.d0)) ! Number of unit cells per
           axis
        ucl = lbox/uc ! Length of unit cell
! Make the unit cell proportional to the ellipsoid: lz/lx = l/d, ³ucl =
    lx*ly*lz, ly=lx --> ³ucl = ³lx*l/d
        lx = (ucl**3.d0*d/l)**(1.d0/3.d0)
        lz = lx*l/d
        ly = lx
        boxl(1) = lx*uc
        boxl(2) = ly*uc
        boxl(3) = lz*uc
        print*,boxl(1),boxl(2),boxl(3)

! Start positioning particles
        id = 1
        Do 101 i = 1,uc
               Do 102 j = 1,uc
                      Do 103 k = 1,uc
                      ! Particles on the right vertice of unit cell
                             r(1,id) = (dble(i)-1.d0)*lx - 0.5*boxl(1)
                             r(2,id) = (dble(j)-1.d0)*ly - 0.5*boxl(2)
                             r(3,id) = (dble(k)-1.d0)*lz - 0.5*boxl(3)
                      ! Particles on the front face of unit cell
                             r(1,id+1) = 0.5d0*lx + (dble(i)-1.d0)*lx -
                                0.5*boxl(1)
                             r(2,id+1) = (dble(j)-1.d0)*ly - 0.5*boxl(2)
                             r(3,id+1) = 0.5d0*lz + (dble(k)-1.d0)*lz -
                                0.5*boxl(3)
                      ! Particles on the left face of unit cell
                             r(1,id+2) = (dble(i)-1.d0)*lx - 0.5*boxl(1)
                             r(2,id+2) = 0.5d0*ly + (dble(j)-1.d0)*ly -
                                0.5*boxl(2)
                             r(3,id+2) = 0.5d0*lz + (dble(k)-1.d0)*lz -
                                0.5*boxl(3)
                      ! Particles on the down face of unit cell
                             r(1,id+3) = 0.5d0*lx + (dble(i)-1.d0)*lx -
                                0.5*boxl(1)
                             r(2,id+3) = 0.5d0*ly + (dble(j)-1.d0)*ly -
                                0.5*boxl(2)
                             r(3,id+3) = (dble(k)-1.d0)*lz - 0.5*boxl(3)
                             id = id + 4
                  103 End DO
           102 End Do
    101 End Do
End
Subroutine packed_box(eta0,quat0_axis,quat0_angle)
        Implicit None
        Integer :: i,j,k,id,nl,nxx,nzz
        Real(8) :: az,ax,eta0,quat0_axis(3),quat0_angle
        Real(8),Allocatable :: rx(:),ry(:),rz(:)
        Character :: moltype*1

! Calculation of Volume
      ! V = N*0.25d0*pi*d*d*l/eta0 !^3Angstrom
        az = l + 0.01d0 + spacez
        ax = d + 0.01d0 + spacex
```

```
       !nl = nint(dble(n)**(1.d0/3.d0))
       nxx = dnint(dble(n)*l/d)**(1.d0/3.d0)
       nzz = dnint(dble(nxx)*d/l)
       boxl(1) = dble(nxx)*ax
       boxl(2) = boxl(1)
       boxl(3) = dble(nzz)*az
       V = boxl(1)*boxl(2)*boxl(3)
       N = nxx*nxx*nzz
! Output File

       Allocate(r(3,N),q(0:3,N),e(3,N))
       ! Convert to radians
       quat0_angle = quat0_angle*pi/180.d0
       quat0_angle = quat0_angle/2.d0
       q(0,:) = dcos(quat0_angle)
       q(1,:) = dsin(quat0_angle)*quat0_axis(1)
       q(2,:) = dsin(quat0_angle)*quat0_axis(2)
       q(3,:) = dsin(quat0_angle)*quat0_axis(3)
! Start positioning particles
       id = 1
       Do 101 i = 1,nxx
              Do 102 j = 1,nxx
                     Do 103 k = 1,nzz
                     ! Particles on the right vertice of unit cell
                            r(1,id) = (dble(i)-1.d0)*ax - 0.5d0*boxl(1)
                            r(2,id) = (dble(j)-1.d0)*ax - 0.5d0*boxl(2)
                            r(3,id) = (dble(k)-1.d0)*az - 0.5d0*boxl(3)
                            id = id + 1
                 103 End DO
          102 End Do
    101 End Do
End
end module
```

C.2.4 Module with main subroutines (module_mc.f90)

```
!**********************************************************************
! Module with important routines to Monte Carlo simulations
! (including overlap check and the calculation of potentials
! between several different nonspherical particles).
!**********************************************************************
! Developer: Joyce Tavares Lopes
! Supervisor: Dr. Luis Fernando Mercier Franco
! School of chemical engineering (FEQ) - UNICAMP
!**********************************************************************
! Some subroutines were developed during a research period at
! Universita Ca' Foscari Venezia under the supervision of
! Professor Achille Giacometi.
```

```
!**********************************************************************
module monte_carlo
use global_variables
implicit none
!-------------------------------------------------------------------------
! Internal Variables
!-------------------------------------------------------------------------
 Real(8),Dimension(3),Private :: e1,e2,r1,r2,r12
 Real(8),Dimension(0:3),Private :: q1,q2
 Real(8),Dimension(3),Private :: mule2,lanle1
 Real(8),Private :: cc,e1e2,r12e1,r12e2,r12sq
 Real(8),Private :: dlvo_pot,sphere_dlvo_pot
 Real(8),Private :: yukawa_pot
contains

Subroutine check_overlap_sc(boxlc,i,ei,ri,rc,overlap)
 Implicit None
 Real(8),Dimension(3),Intent(in) :: ri,ei,boxlc
 ! Matrix of Positions to be checked
 Real(8),Intent(in) :: rc(3,n)
 Logical,Intent(out) :: overlap
 Integer,Intent(in) :: i
 Integer :: j
 Real(8) :: rijsq,dsph,dsph2,sd2
 Real(8),Dimension(3) :: rij,urij
 Logical :: parallel
 dsq = d*d
 dsph = l + d
 dsph2 = dsph*dsph
 r1(:) = ri(:)
 e1(:) = ei(:)
  overlap = .false.
  Do j=1,n
       If (i .ne. j) then
              r2(:) = rc(:,j)
              r12(:) = r2(:) - r1(:)
              e2(:) = e(:,j)

              !Minimum Image
              r12(1) = r12(1) - boxlc(1)*Dnint(r12(1)/boxlc(1))
              r12(2) = r12(2) - boxlc(2)*Dnint(r12(2)/boxlc(2))
              r12(3) = r12(3) - boxlc(3)*Dnint(r12(3)/boxlc(3))
              r12sq = r12(1)*r12(1)+r12(2)*r12(2)+r12(3)*r12(3)
              If (r12sq .le. dsph2) then
                     Call shortest_distance(sd2,parallel)
                     If (sd2 .le. dsq) then
                            overlap = .true.
                            Return
                     End If
              End If
       End if
   End Do
   Return
End Subroutine


Subroutine multiply_quats(a,b,qab)
 Implicit None
 Real(8),Dimension(0:3),Intent(in) :: a,b
 Real(8),Dimension(0:3),Intent(out) :: qab

 qab(0) = a(0)*b(0) - a(1)*b(1) - a(2)*b(2) - a(3)*b(3)
 qab(1) = a(1)*b(0) + a(0)*b(1) - a(3)*b(2) + a(2)*b(3)
 qab(2) = a(2)*b(0) + a(3)*b(1) + a(0)*b(2) - a(1)*b(3)
 qab(3) = a(3)*b(0) - a(2)*b(1) + a(1)*b(2) + a(0)*b(3)
End Subroutine


Subroutine new_position(i,boxlc,ri)
       Implicit None
       Integer, Intent(in) :: i
       Real(8), Intent(in) :: boxlc(3)
       Real(8), Intent(out) :: ri(3)
       Real(8) :: rnum

       Call random_number(rnum)
       ri(1) = r(1,i) + (2.d0*rnum - 1.d0)*max_r
       Call random_number(rnum)
       ri(2) = r(2,i) + (2.d0*rnum - 1.d0)*max_r
       Call random_number(rnum)
       ri(3) = r(3,i) + (2.d0*rnum - 1.d0)*max_r
  ! Cooral box after translation
       ri(1) = ri(1) - boxlc(1)*Dnint(ri(1)/boxlc(1))
       ri(2) = ri(2) - boxlc(2)*Dnint(ri(2)/boxlc(2))
       ri(3) = ri(3) - boxlc(3)*Dnint(ri(3)/boxlc(3))

End Subroutine

Subroutine new_volume(rnew,boxlnew,vnew)
       Implicit None
       Integer :: j
       Real(8), Intent(out) :: boxlnew(3),rnew(3,n),vnew
       Real(8) :: rnum,lnvnew,boxr,vcheck

       Call random_number(rnum)
```

```
        lnvnew = dlog(v) + (2.d0*rnum - 1.d0)*max_v
        vnew = dexp(lnvnew)
        boxr = (vnew/v)**(1.d0/3.d0)
        boxlnew(:) = boxr*boxl(:)

        Do j=1,n
                rnew(1,j) = boxr*r(1,j)
                rnew(2,j) = boxr*r(2,j)
                rnew(3,j) = boxr*r(3,j)
        End Do

End Subroutine

Subroutine floppy_new_volume(rnew,boxlnew,vnew)
        Implicit None
        Integer :: j,axis
        Real(8), Intent(out) :: boxlnew(3),rnew(3,n),vnew
        Real(8) :: rnum,boxr,lmax

        boxlnew = boxl
        rnew = r
        lmax = max_v**(1.d0/3.d0)


        Call random_number(rnum)

        if (rnum .lt. 0.33d0) then
                axis = 1
        else if (rnum .lt. 0.66d0) then
                axis = 2
        else
                axis = 3
        end if

        Call random_number(rnum)

        boxlnew(axis) = boxl(axis) + (2.d0*rnum - 1.d0)*lmax
        boxr = boxlnew(axis)/boxl(axis)


        Do j=1,n
                rnew(axis,j) = boxr*r(axis,j)
        End Do

        vnew = boxlnew(1)*boxlnew(2)*boxlnew(3)


End Subroutine
```

```
Subroutine quat_to_ori(eold,qc,enew)
 Implicit None
 Real(8),Dimension(3),Intent(in) :: eold
 Real(8),Dimension(3),Intent(out) :: enew
 Real(8),Dimension(0:3),Intent(in) :: qc
 Real(8),Dimension(3,3) :: rotM,rotMT
 Integer :: i,j

 !Rotation Matrix rotM(3,3) - Allen and TilVdesley, 2th edition page 110,
     equation 3.40
 rotM(1,1) = qc(0)*qc(0) + qc(1)*qc(1) - qc(2)*qc(2) - qc(3)*qc(3)
 rotM(1,2) = 2.d0*(qc(1)*qc(2) + qc(0)*qc(3))
 rotM(1,3) = 2.d0*(qc(1)*qc(3) - qc(0)*qc(2))
 rotM(2,1) = 2.d0*(qc(1)*qc(2) - qc(0)*qc(3))
 rotM(2,2) = qc(0)*qc(0) - qc(1)*qc(1) + qc(2)*qc(2) - qc(3)*qc(3)
 rotM(2,3) = 2.d0*(qc(2)*qc(3) + qc(0)*qc(1))
 rotM(3,1) = 2.d0*(qc(1)*qc(3) + qc(0)*qc(2))
 rotM(3,2) = 2.d0*(qc(2)*qc(3) - qc(0)*qc(1))
 rotM(3,3) = qc(0)*qc(0) - qc(1)*qc(1) - qc(2)*qc(2) + qc(3)*qc(3)

 !Transpose of Rotation Matrix
 Do i=1,3
       Do j=1,3
              rotMT(i,j) = rotM(j,i)
       End Do
 End Do

 !New orientation
 enew(1) = eold(1)*rotMT(1,1) + eold(2)*rotMT(1,2) + eold(3)*rotMT(1,3)
 enew(2) = eold(1)*rotMT(2,1) + eold(2)*rotMT(2,2) + eold(3)*rotMT(2,3)
 enew(3) = eold(1)*rotMT(3,1) + eold(2)*rotMT(3,2) + eold(3)*rotMT(3,3)
End


Subroutine random_rotate_quat(qold,qnew)
 Implicit None
 Real(8),Dimension(0:3),Intent(in) :: qold
 Real(8),Dimension(0:3),Intent(out) :: qnew
 Real(8),Dimension(0:3) :: qrot
 Call rotation_quat(qrot)
 Call multiply_quats(qrot,qold,qnew)
End Subroutine


Subroutine rotation_quat(quat)
 Implicit None
 Real(8),Dimension(0:3) :: quat
 Real(8) :: rn,ang
 Real(8),Dimension(3) :: axis
 Call random_number(rn) !Random number 'rn' in range [0,1]
 rn = 2.d0*rn - 1d0 !Random number 'rn' now in range [-1,1]
 ang = max_angle*rn !Random angle
 quat(0) = dcos(ang*0.5)
 !Random axis
 Call rand_vector(axis)
 quat(1:3) = dsin(ang*0.5)*axis(:)
End Subroutine


!***********************************************************************
! Code to calculate shortest distance between segments
! Outputs the square of the shortest distance
!-----------------------------------------------------------------------
! Reference:
! "A Fast Algorithm To Evaluate The Shortest Distance Between Rods"
! Carlos Vega and Santiago Lago
! Computers Chem. Vol. 18, No.1i, pp. 55-59, 1994
!***********************************************************************
 Subroutine shortest_distance(sd2,parallel)
  Implicit None
  Real(8),Intent(out) :: sd2
  Real(8) :: mul,lanl
  Real(8) :: deltamu,deltalan
  Real(8) :: halfl1,halfl2
  Real :: start, finish
  Logical,Intent(out) :: parallel
! Initialization
  halfl1 = 0.5d0*l
  halfl2 = 0.5d0*l
  r12sq = r12(1)*r12(1) + r12(2)*r12(2) + r12(3)*r12(3)
  r12e1 = r12(1)*e1(1) + r12(2)*e1(2) + r12(3)*e1(3)
  r12e2 = r12(1)*e2(1) + r12(2)*e2(2) + r12(3)*e2(3)
  e1e2 = e1(1)*e2(1) + e1(2)*e2(2) + e1(3)*e2(3)
  cc = 1.d0 - e1e2*e1e2
  parallel = .false.

! Check if the segments are parallel to each other
  If (cc .lt. 1d-10) then
         parallel = .true.
! Check if the segments are not (almost) perpendicular to rij
       If (dabs(r12e1) .gt. 1d-10) then
! Take the extreme closer to the other particle
             lanl = dsign(halfl1,r12e1)
! Calculate closest point between segment 1 and line 2
```

```
              mul = lanl*e1e2 -r12e2
              If (dabs(mul) .gt. halfl2) then
                     mul = dsign(halfl2,mul)
              End If
! Case in which segments are perperpendicular to rij. sd = rij
       Else
              mul = 0.d0
              lanl = 0.d0
       End if

  Else
! Calculate values of mu' and lambda'
         mul = (r12e1*e1e2 - r12e2)/cc
         lanl = (r12e1 - r12e2*e1e2)/cc

         If ((dabs(mul) .le. halfl2) .and. (dabs(lanl) .le. halfl1)) then
         Else
                deltalan = dabs(lanl) - halfl1
                deltamu = dabs(mul) - halfl2

! Check if it is in Regions 3 or 1
                If (deltalan .gt. deltamu) then
                     lanl = dsign(halfl1,lanl)
                     mul = lanl*e1e2 - r12e2
                     If (dabs(mul) .gt. halfl2) mul = dsign(halfl2,mul)

! Regions 2 or 4
                Else
                     mul = dsign(halfl2,mul)
                     lanl = mul*e1e2 + r12e1
                     If (dabs(lanl) .gt. halfl1) lanl = dsign(halfl1,lanl
                         )

                End if
         End If


  End If

! Vectors to be used in the cylinder overlap check
 lanle1(:) = lanl*e1(:)
 mule2(:) = mul*e2(:)
! Calculates shortest distance sd
 sd2 = r12sq + lanl*lanl + mul*mul + 2.d0*mul*r12e2 -&
     &2.d0*lanl*r12e1 - 2.d0*mul*lanl*e1e2
 Return

 End Subroutine
```

```
Subroutine Rand_vector(vec)
 Implicit None
 Real(8), Intent(out) :: vec(3)
 Real(8) :: n1,n2,nsq

 nsq = 2.d0
 Do while (nsq .gt. 1.d0)
       Call random_number(n1)
       Call random_number(n2)
       n1 = 2d0*n1 - 1.d0
       n2 = 2d0*n2 - 1.d0
       nsq = n1*n1 + n2*n2
 End Do
       vec(1) = 2.d0*n1*dsqrt(1.d0-nsq)
       vec(2) = 2.d0*n2*dsqrt(1.d0-nsq)
       vec(3) = 1.d0-2.d0*nsq
End

Subroutine check_overlap_cylinder(boxlc,i,qi,ei,ri,rc,overlap)
 Implicit None
 Real(8),Dimension(3),Intent(in) :: ri,ei,boxlc
 Real(8),Dimension(0:3),Intent(in) :: qi
 ! Matrix of Positions to be checked
 Real(8),Intent(in) :: rc(3,n)
 Logical,Intent(out) :: overlap
 Integer,Intent(in) :: i
 Integer :: j
 Real(8) :: dsph,dsph2,sd2
 Logical :: parallel

 dsq = d*d
 lsq = l*l
 dsph = l + d
 dsph2 = dsph*dsph
 r1(:) = ri(:)
 e1(:) = ei(:)
 q1(:) = qi(:)
  Do j=1,n
       If (i .ne. j) then
              r2(:) = rc(:,j)
              r12(:) = r2(:) - r1(:)
              e2(:) = e(:,j)
              q2(:) = q(:,j)
              !Minimum Image
              r12(1) = r12(1) - boxlc(1)*Dnint(r12(1)/boxlc(1))
              r12(2) = r12(2) - boxlc(2)*Dnint(r12(2)/boxlc(2))
```

```fortran
              r12(3) = r12(3) - boxlc(3)*Dnint(r12(3)/boxlc(3))
              r12sq = r12(1)*r12(1)+r12(2)*r12(2)+r12(3)*r12(3)

 ! Check if the spheres overlap
              If (r12sq .le. dsph2) then
 ! Check overlap of spherocylinders
                     Call shortest_distance(sd2,parallel)
                     If (sd2 .le. dsq) then
                            r2(:) = r1(:) + r12(:)
                            Call overlap_cylinder(sd2,parallel,overlap)
                            If(overlap) return
                     End If

              End If
       End if
   End Do
   overlap = .false.
End Subroutine

Subroutine overlap_cylinder(sd2,parallel,overlap)
 Implicit None
 ! Matrix of Positions to be checked
 Logical,Intent(out) :: overlap
 Logical,Intent(in) :: parallel
 ! Square of Shortest Distance betweem segments
 Real(8),Intent(in) :: sd2
 Real(8) :: r12_parallel(3)
 Real(8) :: r12_perpendicular(3)
 !Square of the parallel r12psq(1) and perpendicular r12psq(2)
 !components of r12
 Real(8) :: r12psq(2)
 ! Position of Disks of Cylinders 1 and 2
 Real(8) :: d1(2,3),d2(2,3),di(3),dj(3)
 Real(8) :: ei(3),ej(3),ri(3),qi(0:3),qj(0:3)
 Integer :: i,j
 halfl = 0.5d0*l
 halfd = 0.5d0*d

 ! Check overlap if the cylinders are parallel
  If (parallel) then
        r12_parallel(:) = e1(:)*r12e1
        r12_perpendicular(:) = r12(:) - r12_parallel(:)
        r12psq(1) = r12_parallel(1)*r12_parallel(1)&
                    &+ r12_parallel(2)*r12_parallel(2)&
                    &+ r12_parallel(3)*r12_parallel(3)
        r12psq(2) = r12_perpendicular(1)*r12_perpendicular(1)&
                    &+ r12_perpendicular(2)*r12_perpendicular(2)&
                    &+ r12_perpendicular(3)*r12_perpendicular(3)
        If (r12psq(1) .le. l*l .and. r12psq(2) .le. d*d) then
             overlap = .true.
        Else
             overlap = .false.
        End If

 Else
       Call rim_rim(sd2,overlap)
       !If (overlap) print*,'rim-rim overlap'
       If (overlap) return
! Check overlap between all disks
        d1(1,:) = r1(:) + e1(:)*halfl
        d1(2,:) = r1(:) - e1(:)*halfl
        d2(1,:) = r2(:) + e2(:)*halfl
        d2(2,:) = r2(:) - e2(:)*halfl
        ei(:) = e1(:)
        ej(:) = e2(:)
        Do i=1,2
              di(:) = d1(i,:) !Disks on cylinder 1
              Do j=1,2
                     dj(:) = d2(j,:)
                     Call disk_disk(di,dj,ei,ej,overlap)
                     !If (overlap) print*,'disk-disk overlap'
                     If (overlap) return
              End Do
        End Do


 ! Rim of Cylinder 1 versus Disks on Cylinder 2
        ri(:) = r1(:)
        ei(:) = e1(:)
        qi(:) = q1(:)
        ej(:) = e2(:)
        qj(:) = q2(:)

        Do j=1,2
                     dj(:) = d2(j,:)
                     Call disk_rim(dj,ej,qj,ri,ei,qi,overlap)
                     !If (overlap) print*,'disk-rim overlap'
                     If (overlap) return
        End Do
 ! Rim of Cylinder 2 versus Disks on Cylinder 1
        ri(:) = r2(:)
        ei(:) = e2(:)
        qi(:) = q2(:)
        ej(:) = e1(:)
        qj(:) = q1(:)
```

```
         Do j=1,2
                      dj(:) = d1(j,:)
                      Call disk_rim(dj,ej,qj,ri,ei,qi,overlap)
                      !If (overlap) print*,'disk-rim overlap'
                      If (overlap) return
         End Do

   End If
End Subroutine

Subroutine rim_rim(sd2,overlap)
 Implicit None
 Logical,Intent(out) :: overlap
 Real(8),Intent(in) :: sd2
 Real(8) :: proj1,proj2

! mu and lambda are from the shortest_distance algorithm
! Calculate the projection of r12 + mue2 on e1
 proj1 = (r12(1) +mule2(1))*e1(1)
 proj1 = proj1 + (r12(2) +mule2(2))*e1(2)
 proj1 = proj1 + (r12(3) +mule2(3))*e1(3)
 proj1 = dabs(proj1)
! Calculate the projection of r12 + lambdae1 on e2
 proj2 = (-r12(1) +lanle1(1))*e2(1)
 proj2 = proj2 + (-r12(2) +lanle1(2))*e2(2)
 proj2 = proj2 + (-r12(3) +lanle1(3))*e2(3)
 proj2 = dabs(proj2)
 ! If the shortest distance between the two cylinders pass through
 ! both rims, the overlap occurs since it is the same as in the
 ! spherocylinders overlap, which has already been tested and found
 ! to be true
 If ( proj1 .lt. halfl .and. proj2 .lt. halfl) then
        overlap = .true.
 Else
        overlap = .false.
 End If

! If (sd2 .le. D) overlap = .true.


End Subroutine

Subroutine disk_disk(di,dj,ei,ej,overlap)
 Implicit None
 Logical,Intent(out) :: overlap
 Real(8),Intent(in) :: di(3),dj(3),ei(3),ej(3)
 Real(8) :: idistsq,jdistsq,dij(3),dijsq
 Real(8) :: segi,segj,radiussq
 Real(8) :: eij(3),modeij,pipj

! Vector joining disks centers of mass

 dij(:) = dj(:) - di(:)
 dijsq = dij(1)*dij(1) + dij(2)*dij(2) + dij(3)*dij(3)
 ! Check if the distance between disks is less than D. If it is larger
    than D,
 ! there is no overlap
 If (dijsq .gt. dsq) then
        overlap = .false.
        return
 End If
! Square of Shortest Distance between disk i and
! interesection line between plans of the disks
 idistsq = dij(1)*ej(1) + dij(2)*ej(2) + dij(3)*ej(3)
 idistsq = idistsq*idistsq
 idistsq = idistsq/cc

! Square of Shortest Distance between disk j and
! interesection line between plans of the disks
 jdistsq = dij(1)*ei(1) + dij(2)*ei(2) + dij(3)*ei(3)
 jdistsq = jdistsq*jdistsq
 jdistsq = jdistsq/cc


! Test the necessary but not sufficient condition for the
! overlap
 radiussq = halfd*halfd
 If (idistsq .lt. radiussq .and. jdistsq .lt. radiussq) then
! Test overlap
       segi = dsqrt(radiussq - idistsq)
       segj = dsqrt(radiussq - jdistsq)

! Calculate the projection of dij in the
! direction of the instersection line

! Direction of intersection line between the plans of the
! two disks --> eij
       Call cross_product(ei,ej,eij)
! Normalize orientation of intersection line eij
       modeij = eij(1)*eij(1) + eij(2)*eij(2) + eij(3)*eij(3)
       modeij = dsqrt(modeij)
       eij(:) = eij(:)/modeij
! Projection of dij in the eij
       pipj = dij(1)*eij(1) + dij(2)*eij(2) + dij(3)*eij(3)
       pipj = dabs(pipj)
       If (pipj .le. (segi + segj)) then
```

```
                overlap = .true.
            Else
                overlap = .false.
            End If

      Else
            overlap = .false.
      End If

     End Subroutine

     Subroutine disk_rim(dj,ej,qj,ri,ei,qi,overlap)
      Logical,Intent(out) :: overlap
      Logical :: do_bisec
      Real(8),Intent(in) :: dj(3),ri(3),ei(3),ej(3)
      Real(8),Intent(in) :: qi(0:3),qj(0:3)
      ! ei = eiz -- > eiy and eix are the other axis on the particle
      Real(8),Dimension(3):: ejy(3),ejx(3)
      Real(8),Dimension(3):: ui,djri,djui,pc,pd,T
      Real(8) :: Tpsq(2),Tsq,djri_ei
      Real(8) :: djuisq,dhyp,fact
      Real(8) :: ejx_ei,ejy_ei,djri_ejy,djri_ejx
      Real(8) :: lambda,num,den,hyp,cosphi,sinphi
      Real(8) :: lambda1,lambda2,lambdai,fi
      Real(8) :: tol,w,f1,f2,f,df,ddhyp,dump
      Integer :: cont,contb
      djri(:) = dj(:) - ri(:)
      djri_ei = djri(1)*ei(1) + djri(2)*ei(2) + djri(3)*ei(3)

     ! uj is the closest point to disk i on cylinder j (or on the cylinder
        axis line)
      ui(:) = ri(:) + ei(:)*djri_ei
      djui(:) = dj(:) - ui(:)
      djuisq = djui(1)*djui(1) + djui(2)*djui(2) + djui(3)*djui(3)

      If (djuisq .gt. dsq) then
             overlap = .false.
      Else If (djuisq .lt. halfd*halfd .and. dabs(djri_ei) .gt. halfl) then
             ! In this case, the overlap check is a disk-disk check, so, if it
             ! the disk-disk overlap has already been tested, there is no
             ! overlap. Otherwise, this test should not be done here, but in
             ! disk-disk check.
             overlap = .false.
             return

      Else If (djuisq .le. halfd*halfd .and. dabs(djri_ei) .le. halfl) then
            ! In this case, there is overlap
             !print*,'here'
            overlap = .true.
            !print*,'disk-rim center overlap'
            return

      Else
      ! The overlap might happen between another point on the circle of disk i
      ! and cylinder j. Here a iterative process is necessary to find the
         closest
      ! approach between any point on the circle of the disk and the rim.
      ! Axis on cylinders i
            Call quat_to_ori(eyfixed,qj,ejy)
            Call quat_to_ori(exfixed,qj,ejx)
            ejx_ei = ejx(1)*ei(1) + ejx(2)*ei(2) + ejx(3)*ei(3)
            ejy_ei = ejy(1)*ei(1) + ejy(2)*ei(2) + ejy(3)*ei(3)
            djri_ejx = djri(1)*ejx(1) + djri(2)*ejx(2) + djri(3)*ejx(3)
            djri_ejy = djri(1)*ejy(1) + djri(2)*ejy(2) + djri(3)*ejy(3)

      ! Point on disk --> Pd = Dj + Rcos(phi)ejx + Rsin(phi)ejx
      ! Point on cylinder i --> Pc = ri + lambdaei
      ! Function that minimizes the distance between disk and cylinder =
      ! lambda - R(•ejxei)cos(phi) - Rsin(phi)(ejy*ei) - (Djri*ei) = 0
      ! lambda - R(ejx*ei)den/hyp - R(ejy*ei)num/hyp - (Djri*ei) = 0

      ! Calculate the function with f1 = f(lambda = L/2) and f2 = f(lambda = -
         L/2)
            lambda = halfl
            Call func_lambda(ejy_ei,djri_ejy,ejx_ei,djri_ejx,djri_ei,lambda,f1
               ,dump)

            lambda = -halfl
            Call func_lambda(ejy_ei,djri_ejy,ejx_ei,djri_ejx,djri_ei,lambda,f2
               ,dump)

            ! if f1*f2 > 0, the point of closest of the disk j is outside the
               rim
            ! of the cylinder i (root of f is out the limits of the cylinder
               segment)
            ! and so, it is checked in the other possible configurations
            If (f1*f2 .gt. 0 ) then
                   overlap = .false.
                   return
            End If


            ! Newton-Raphson (max of 20 iterations)
            lambda = 0.d0
            w = 0.95d0
            cont = 0
```

```
tol = 1d-6
do_bisec = .false.

Call func_lambda(ejy_ei,djri_ejy,ejx_ei,djri_ejx,djri_ei,lambda,f,
   df)

Do while (dabs(f) .gt. tol .and. .not. do_bisec)
       cont = cont + 1
       If (cont .gt. 20 ) do_bisec = .true.
       If (cont .gt. 0 .and. mod(cont,10) .eq. 0) w = 0.99d0*w
       fact = -f/df
       ! If it fact gives a value outside the length of the
          cylinder,
       ! do this to put it back in
       If (dabs(fact) .gt. halfl) fact = dsign(halfl,fact)
       lambda = lambda + w*fact
       Call func_lambda(ejy_ei,djri_ejy,ejx_ei,djri_ejx,djri_ei,
          lambda,f,df)
End Do

! Bisection (if over 20 Newton-Raphson Iterations)
If (do_bisec) then
contb = 0
lambda1 = -halfl
lambda2 = halfl
Call func_lambda(ejy_ei,djri_ejy,ejx_ei,djri_ejx,djri_ei,lambda1,
   f1,dump)
Call func_lambda(ejy_ei,djri_ejy,ejx_ei,djri_ejx,djri_ei,lambda2,
   f2,dump)
lambdai = 0.5d0*(lambda1 + lambda2)
Call func_lambda(ejy_ei,djri_ejy,ejx_ei,djri_ejx,djri_ei,lambdai,
   fi,dump)
Do while (dabs(fi) .gt. tol)
       If (fi*f1 .gt. 0.d0) then
              lambda1 = lambdai
              f1 = fi
       Else
              lambda2 = lambdai
              f2 = fi
       End If
       lambdai = 0.5d0*(lambda1 + lambda2)
       Call func_lambda(ejy_ei,djri_ejy,ejx_ei,djri_ejx,djri_ei,&
                     &lambdai,fi,dump)
       contb = contb + 1
       If (contb .gt. 100000) print*,'Too many bisection
          iterations=',contb

End Do
```

```
        lambda = lambdai

        End If

 ! Point on Disk (Pd) and on Cylinder (Pc)
        num = lambda*ejy_ei - djri_ejy
        den = lambda*ejx_ei - djri_ejx
        hyp = dsqrt(num*num + den*den)
        cosphi = den/hyp
        sinphi = num/hyp
        Pd(:) = dj(:) + halfd*cosphi*ejx(:) + halfd*sinphi*ejy(:)
        Pc(:) = ri(:) + lambda*ei(:)
        !T(:) = Pd(:) - Pc(:)
        T(:) = Pd(:) - ri(:)
        Tsq = T(1)*T(1) + T(2)*T(2) + T(3)*T(3)
 ! Square of Component of T parallel to ei
        Tpsq(1) = ei(1)*T(1) + ei(2)*T(2) + ei(3)*T(3)
        Tpsq(1) = Tpsq(1)*Tpsq(1)
 ! Square of Component of T perpendicular to ei
        Tpsq(2) = Tsq - Tpsq(1)

        If (Tpsq(1) .lt. halfl*halfl .and. Tpsq(2) .lt. halfd*halfd) then
               overlap = .true.
        Else
               overlap = .false.
        End If
 End If

End Subroutine

Subroutine func_lambda(ejy_ei,djri_ejy,ejx_ei,djri_ejx,djri_ei,lambda,f,
   df)
 Implicit None
 Real(8),Intent(in) :: ejy_ei,djri_ejy,ejx_ei,djri_ejx,djri_ei,lambda
 Real(8),Intent(out) :: f,df
 Real(8) :: num,den,hyp,dhyp,ddhyp
  num = lambda*ejy_ei - djri_ejy
  den = lambda*ejx_ei - djri_ejx
  hyp = dsqrt(num*num + den*den)
  dhyp = (num*ejy_ei + den*ejx_ei)/hyp
  f = lambda - halfd*dhyp - djri_ei
  ddhyp = ((ejy_ei*ejy_ei + ejx_ei*ejx_ei) - dhyp*dhyp)/hyp
  df = 1.d0 - halfd*ddhyp

End Subroutine


Subroutine cross_product(v1,v2,vout)
```

```
 Implicit none
 Real(8),Intent(in) :: v1(3),v2(3)
 Real(8),Intent(out) :: vout(3)
 vout(1) = v1(2)*v2(3) - v1(3)*v2(2)
 vout(2) = v1(3)*v2(1) - v1(1)*v2(3)
 vout(3) = v1(1)*v2(2) - v1(2)*v2(1)

End Subroutine

Subroutine partial_potential_patchy_cylinder(boxlc,i,qi,ei,ri,rc,overlap,
   potij)
 Implicit None
 Real(8),Dimension(3),Intent(in) :: ri,ei,boxlc
 Real(8),Dimension(0:3),Intent(in) :: qi
 ! Matrix of Positions to be checked
 Real(8),Intent(in) :: rc(3,n)
 Logical,Intent(out) :: overlap
 Real(8),Intent(out) :: potij
 Real(8) :: r12_patch(3),sq_r12_patch
 Integer,Intent(in) :: i
 Integer :: j,k,z,n_inter
 Real(8) :: dsph,dsph2,sd2
 Real(8) :: patches_c1(3,n_patches)
 Real(8) :: patches_c2(3,n_patches)
 Logical :: parallel
 overlap = .false.
 potij = 0.d0
 dsq = d*d
 lsq = l*l
 dsph = l + d
 dsph2 = dsph*dsph
 r1(:) = ri(:)
 e1(:) = ei(:)
 q1(:) = qi(:)
 halfl = 0.5d0*l
  Do j=1,n
        If (i .ne. j) then
               r2(:) = rc(:,j)
               r12(:) = r2(:) - r1(:)
               e2(:) = e(:,j)
               q2(:) = q(:,j)
               !Minimum Image
               r12(1) = r12(1) - boxlc(1)*Dnint(r12(1)/boxlc(1))
               r12(2) = r12(2) - boxlc(2)*Dnint(r12(2)/boxlc(2))
               r12(3) = r12(3) - boxlc(3)*Dnint(r12(3)/boxlc(3))
               r12sq = r12(1)*r12(1)+r12(2)*r12(2)+r12(3)*r12(3)
               r2(:) = r1(:) + r12(:)
 ! Check if the spheres overlap
              If (r12sq .le. dsph2) then
 ! Check overlap of spherocylinders
                     Call shortest_distance(sd2,parallel)
                     If (sd2 .le. dsq) then
                            Call overlap_cylinder(sd2,parallel,overlap)
                            If(overlap) return
                     End If

              End If

              If (.not. overlap .and. r12sq .lt. rcutsq) then
                     Call patches_allocation(r1,e1,patches_c1)
                     Call patches_allocation(r2,e2,patches_c2)
                     Call patches_interaction(boxlc,patches_c1,patches_c2
                        ,n_inter)
                     potij = potij - dble(n_inter)*eps
              End If
       End if
  End Do

End Subroutine


Subroutine total_potential_patchy_cylinder(boxlc,rc,overlap,potij)
 Implicit None
 Real(8),Dimension(3),Intent(in) :: boxlc
 ! Matrix of Positions to be checked
 Real(8),Intent(in) :: rc(3,n)
 Logical,Intent(out) :: overlap
 Real(8),Intent(out) :: potij
 Real(8) :: r12_patch(3),sq_r12_patch
 Integer :: i
 Integer :: j,k,z,n_inter
 Real(8) :: dsph,dsph2,sd2
 Real(8) :: patches_c1(3,n_patches)
 Real(8) :: patches_c2(3,n_patches)
 Logical :: parallel
 overlap = .false.
 potij = 0.d0
 dsq = d*d
 lsq = l*l
 dsph = l + d
 dsph2 = dsph*dsph
 halfl = 0.5d0*l
 Do i =1,n-1
        r1(:) = rc(:,i)
        e1(:) = e(:,i)
        q1(:) = q(:,i)
       Do j=i+1,n
              r2(:) = rc(:,j)
              r12(:) = r2(:) - r1(:)
              e2(:) = e(:,j)
              q2(:) = q(:,j)
              !Minimum Image
              r12(1) = r12(1) - boxlc(1)*Dnint(r12(1)/boxlc(1))
              r12(2) = r12(2) - boxlc(2)*Dnint(r12(2)/boxlc(2))
              r12(3) = r12(3) - boxlc(3)*Dnint(r12(3)/boxlc(3))
              r12sq = r12(1)*r12(1)+r12(2)*r12(2)+r12(3)*r12(3)
              r2(:) = r1(:) + r12(:)

 ! Check if the spheres overlap
              If (r12sq .le. dsph2) then
 ! Check overlap of spherocylinders
                     Call shortest_distance(sd2,parallel)
                     If (sd2 .le. dsq) then
                            Call overlap_cylinder(sd2,parallel,overlap)
                            If(overlap) return
                     End If

              End If

              If (.not. overlap .and. r12sq .lt. rcutsq) then
                     Call patches_allocation(r1,e1,patches_c1)
                     Call patches_allocation(r2,e2,patches_c2)
                     Call patches_interaction(boxlc,patches_c1,patches_c2
                        ,n_inter)
                     potij = potij - dble(n_inter)*eps
              End If
   End Do
   End Do

End Subroutine


Subroutine patches_allocation(rc,ec,patches)
 implicit none
 real(8),intent(out) :: patches(3,n_patches)
 real(8),intent(in) :: rc(3),ec(3)
 integer :: i,j
 patches(:,1) = rc(:) + ec(:)*halfl
 patches(:,2) = rc(:) - ec(:)*halfl
End Subroutine

Subroutine helices_allocation(rc,ec,qc,helix)
 implicit none
 real(8),intent(in) :: rc(3),ec(3),qc(0:3)
 real(8),intent(out) :: helix(3,n_beads)
 real(8) :: ang,bottom(3),ex(3),ey(3)
 integer :: i

 Call quat_to_ori(exfixed,qc,ex)
 Call quat_to_ori(eyfixed,qc,ey)
 bottom(:) = rc(:) - ec(:)*halfl
 ang = 2.d0*pi*dble(np)/dble(n_beads - 1)
 Do i=1,n_beads
      helix(:,i) = bottom(:) + r_h*dcos((i-1)*ang)*ex(:)&
              &+ r_h*dsin((i-1)*ang)*ey(:) + ((i-1)*ang*pitch/(2.d0*pi))*
                 ec(:)
 End Do

End Subroutine

Subroutine helices_parameters()
 implicit none
 real(8) :: aux,nb,f,a,b,c,x(2)
 d_bead = d*d_bead
 sqd_bead = d_bead*d_bead
 f = 1.d0 - fusion
 halfd = 0.5d0*d
 r_bead = 0.5d0*d_bead
 r_h = halfd + r_bead
 aux = dble(np)*dsqrt((l/dble(np))**2.d0 + pi*pi*(d+d_bead)**2.d0)/d_bead
 n_beads = floor(aux/f +1.d0)
 print*,'fusion before',1.d0 - f
 f = aux/dble(n_beads-1)
 print*,'fusion after',1.d0 - f
 pitch = l/dble(np)
 l_h = dsqrt(pitch*pitch + 4.d0*pi*pi*r_h*r_h)
End Subroutine




Subroutine patches_interaction(boxlc,patches_c1,patches_c2,n_inter)
 implicit none
 real(8),intent(in) :: boxlc(3)
 real(8),intent(in) :: patches_c1(3,n_patches)
 real(8),intent(in) :: patches_c2(3,n_patches)
 real(8) :: r12_patch(3),sq_r12_patch
 integer :: i,j
 integer,intent(out) :: n_inter
 n_inter = 0
 Do i=1,n_patches !Loop over patches on cylinder 1
        Do j=1,n_patches !Loop over patches on cylinder 2
               r12_patch(:) = patches_c1(:,i) - patches_c2(:,j)

           ! ! Minimum Image Convention

               sq_r12_patch = r12_patch(1)*r12_patch(1) + r12_patch(2)*
                  r12_patch(2) + &
                            & r12_patch(3)*r12_patch(3)
               if (sq_r12_patch .le. swrange) then
                      n_inter = n_inter + 1
               end if
        End do
 End do

End Subroutine
Subroutine partial_potential_helical_patchy_cylinder(boxlc,i,qi,ei,ri,rc,
   overlap,potij)
 Implicit None
 Real(8),Dimension(3),Intent(in) :: ri,ei,boxlc
 Real(8),Dimension(0:3),Intent(in) :: qi
 ! Matrix of Positions to be checked
 Real(8),Intent(in) :: rc(3,n)
 Logical,Intent(out) :: overlap
 Real(8),Intent(out) :: potij
 Real(8) :: r12_patch(3),sq_r12_patch
 Integer,Intent(in) :: i
 Integer :: j,k,z,n_inter
 Real(8) :: dsph,dsph2,sd2
 Real(8) :: patches_c1(3,n_patches),beads_c1(3,n_beads)
 Real(8) :: patches_c2(3,n_patches),beads_c2(3,n_beads)
 Logical :: parallel
 overlap = .false.
 potij = 0.d0
 dsq = d*d
 lsq = l*l
 dsph = l + d
 dsph2 = dsph*dsph
 r1(:) = ri(:)
 e1(:) = ei(:)
 q1(:) = qi(:)
 halfl = 0.5d0*l
  Do j=1,n
        If (i .ne. j) then
               r2(:) = rc(:,j)
               r12(:) = r2(:) - r1(:)
               e2(:) = e(:,j)
               q2(:) = q(:,j)
```

```
              !Minimum Image
              r12(1) = r12(1) - boxlc(1)*Dnint(r12(1)/boxlc(1))
              r12(2) = r12(2) - boxlc(2)*Dnint(r12(2)/boxlc(2))
              r12(3) = r12(3) - boxlc(3)*Dnint(r12(3)/boxlc(3))
              r12sq = r12(1)*r12(1)+r12(2)*r12(2)+r12(3)*r12(3)
              r2(:) = r1(:) + r12(:)

 ! Check if the spheres overlap
              If (r12sq .le. dsph2) then
 ! Check overlap of spherocylinders
                     Call shortest_distance(sd2,parallel)
                     If (sd2 .le. dsq) then
                            Call overlap_cylinder(sd2,parallel,overlap)
                            If(overlap) return
                     End If

              End If

 ! Check beads overlap
              if (.not. overlap) then
                     Call helices_allocation(r1,e1,q1,beads_c1)
                     Call helices_allocation(rc(:,j),e2,q2,beads_c2)
                     Call beads_overlap(boxlc,beads_c1,beads_c2,overlap)
                     if(overlap) return
              end if

              If (.not. overlap .and. r12sq .lt. rcutsq) then
                     Call patches_allocation(r1,e1,patches_c1)
                     Call patches_allocation(r2,e2,patches_c2)
                     Call patches_interaction(boxlc,patches_c1,patches_c2
                        ,n_inter)
                     potij = potij - dble(n_inter)*eps
              End If
       End if
   End Do

End Subroutine

Subroutine total_potential_helical_patchy_cylinder(boxlc,rc,overlap,potij
   )
 Implicit None
 Real(8),Dimension(3),Intent(in) :: boxlc
 ! Matrix of Positions to be checked
 Real(8),Intent(in) :: rc(3,n)
 Logical,Intent(out) :: overlap
 Real(8),Intent(out) :: potij
 Real(8) :: r12_patch(3),sq_r12_patch
 Integer :: i
 Integer :: j,k,z,n_inter
 Real(8) :: dsph,dsph2,sd2
 Real(8) :: patches_c1(3,n_patches),beads_c1(3,n_beads)
 Real(8) :: patches_c2(3,n_patches),beads_c2(3,n_beads)
 Logical :: parallel
 overlap = .false.
 potij = 0.d0
 dsq = d*d
 lsq = l*l
 dsph = l + d
 dsph2 = dsph*dsph
 halfl = 0.5d0*l
 Do i =1,n-1
        r1(:) = rc(:,i)
        e1(:) = e(:,i)
        q1(:) = q(:,i)
       Do j=i+1,n
              r2(:) = rc(:,j)
              r12(:) = r2(:) - r1(:)
              e2(:) = e(:,j)
              q2(:) = q(:,j)
              !Minimum Image
              r12(1) = r12(1) - boxlc(1)*Dnint(r12(1)/boxlc(1))
              r12(2) = r12(2) - boxlc(2)*Dnint(r12(2)/boxlc(2))
              r12(3) = r12(3) - boxlc(3)*Dnint(r12(3)/boxlc(3))
              r12sq = r12(1)*r12(1)+r12(2)*r12(2)+r12(3)*r12(3)
              r2(:) = r1(:) + r12(:)

 ! Check if the spheres overlap
              If (r12sq .le. dsph2) then
 ! Check overlap of spherocylinders
                     Call shortest_distance(sd2,parallel)
                     If (sd2 .le. dsq) then
                             Call overlap_cylinder(sd2,parallel,overlap)
                             If(overlap) return
                     End If

              End If

 ! Check beads overlap
              if (.not. overlap) then
                     Call helices_allocation(r1,e1,q1,beads_c1)
                     Call helices_allocation(rc(:,j),e2,q2,beads_c2)
                     Call beads_overlap(boxlc,beads_c1,beads_c2,overlap)
                     if(overlap) return
              end if
```

```
              If (.not. overlap .and. r12sq .lt. rcutsq) then
                     Call patches_allocation(r1,e1,patches_c1)
                     Call patches_allocation(r2,e2,patches_c2)
                     Call patches_interaction(boxlc,patches_c1,patches_c2
                        ,n_inter)
                     potij = potij - dble(n_inter)*eps
              End If
   End Do
   End Do

End Subroutine

Subroutine beads_overlap(boxlc,beads_1,beads_2,overlap)
 implicit none
 real(8),intent(in) :: boxlc(3),beads_1(3,n_beads),beads_2(3,n_beads)
 logical,intent(out) :: overlap
 real(8) :: bij(3),bijsq
 integer :: i,j
 overlap = .false.
 Do i =1,n_beads !Loop over beads on cylinder 1
       Do j=1,n_beads !Loop over beads on cylinder 2
              bij(:) = beads_1(:,i) - beads_2(:,j)

             ! Minimum image

              bijsq = bij(1)*bij(1) + bij(2)*bij(2) + bij(3)*bij(3)
              if (bijsq .le. sqd_bead) then
                     overlap = .true.
                     return
              end if
       End Do
 End Do
End Subroutine

Subroutine bead_cylinder_overlap(boxlc,beads_1,beads_2,overlap)
 implicit none
 real(8),intent(in) :: boxlc(3),beads_1(3,n_beads),beads_2(3,n_beads)
 logical,intent(out) :: overlap
 integer :: i,j
 real(8) :: rcb(3),rcb_pll(3),rcb_per(3),rcb_ori
 real(8) :: mod_per,mod_pll
 real(8) :: dist_per,dist_pll

 dist_pll = halfl + r_bead
 dist_pll = dist_pll*dist_pll
 dist_per = halfd + r_bead
 dist_per = dist_per*dist_per
 overlap = .false.
 ! Cylinder 1 x Beads on cylinder 2
 Do i=1,n_beads
       rcb(:) = r1(:) - beads_2(:,i)

       ! Minimum Image

       rcb_ori = e1(1)*rcb(1) + e1(2)*rcb(2) + e1(3)*rcb(3)
       rcb_pll(:) = e1(:)*rcb_ori
       rcb_per(:) = rcb(:) - rcb_pll(:)
       mod_per = rcb_per(1)*rcb_per(1) + rcb_per(2)*rcb_per(2) + rcb_per
           (3)*rcb_per(3)
       mod_pll = rcb_pll(1)*rcb_pll(1) + rcb_pll(2)*rcb_pll(2) + rcb_pll
           (3)*rcb_pll(3)
       if(mod_per .lt. dist_per .and. mod_pll .lt. dist_pll) then
               overlap = .true.
               return
       end if
 End Do

 ! Cylinder 2 x Beads on cylinder 1

 Do i=1,n_beads
       rcb(:) = r2(:) - beads_1(:,i)

       ! Minimum Image
       ! rcb(:) = rcb(:) - boxlc(:)*dnint(rcb(:)/boxlc(:))

       rcb_ori = e2(1)*rcb(1) + e2(2)*rcb(2) + e2(3)*rcb(3)
       rcb_pll(:) = e2(:)*rcb_ori
       rcb_per(:) = rcb(:) - rcb_pll(:)
       mod_per = rcb_per(1)*rcb_per(1) + rcb_per(2)*rcb_per(2) + rcb_per
           (3)*rcb_per(3)
       mod_pll = rcb_pll(1)*rcb_pll(1) + rcb_pll(2)*rcb_pll(2) + rcb_pll
           (3)*rcb_pll(3)
       if(mod_per .lt. dist_per .and. mod_pll .lt. dist_pll) then
               overlap = .true.
               return
       end if
 End Do

End subroutine

Subroutine check_overlap_helices_cylinder(boxlc,i,qi,ei,ri,rc,overlap)
 Implicit None
 Real(8),Dimension(3),Intent(in) :: ri,ei,boxlc
 Real(8),Dimension(0:3),Intent(in) :: qi
 ! Matrix of Positions to be checked
```

```
Real(8),Intent(in) :: rc(3,n)
Logical,Intent(out) :: overlap
Integer,Intent(in) :: i
Integer :: j
Real(8) :: dsph,dsph2,sd2
Logical :: parallel
Real(8) :: beads_c1(3,n_beads)
Real(8) :: beads_c2(3,n_beads)

overlap = .false.
dsq = d*d
lsq = l*l
dsph = l + maxval((/d,d_bead/))
dsph2 = dsph*dsph
r1(:) = ri(:)
e1(:) = ei(:)
q1(:) = qi(:)
 Do j=1,n
      If (i .ne. j) then
              r2(:) = rc(:,j)
              r12(:) = r2(:) - r1(:)
              e2(:) = e(:,j)
              q2(:) = q(:,j)
              !Minimum Image
              r12(1) = r12(1) - boxlc(1)*Dnint(r12(1)/boxlc(1))
              r12(2) = r12(2) - boxlc(2)*Dnint(r12(2)/boxlc(2))
              r12(3) = r12(3) - boxlc(3)*Dnint(r12(3)/boxlc(3))
              r12sq = r12(1)*r12(1)+r12(2)*r12(2)+r12(3)*r12(3)

! Check if the spheres overlap
              If (r12sq .le. dsph2) then
                     r2(:) = r1(:) + r12(:)
! Check overlap of spherocylinders
                     Call shortest_distance(sd2,parallel)
                     If (sd2 .le. dsq) then
! Check overlap of cylinders
                            Call overlap_cylinder(sd2,parallel,overlap)
                            If(overlap) return
                     End If
! Check Overlap overlap between beads and then between bead-cylinder
! Allocate helical beads around cylinders
                     Call helices_allocation(r1,e1,q1,beads_c1)
                     Call helices_allocation(r2,e2,q2,beads_c2)
                     ! Check bead-bead overlap
                     Call beads_overlap(boxlc,beads_c1,beads_c2,overlap)
                     If (overlap) return
                     ! Check bead-cylinder overlap
                     Call bead_cylinder_overlap(boxlc,beads_c1,beads_c2,
                              overlap)
                          If (overlap) return

                    End If
              End if
          End Do
        End Subroutine
        Subroutine partial_potential_patchy_sphere(boxlc,i,qi,ei,ri,rc,overlap,
           potij)
         Implicit None
         Real(8),Dimension(3),Intent(in) :: ri,ei,boxlc
         Real(8),Dimension(0:3),Intent(in) :: qi
         ! Matrix of Positions to be checked
         Real(8),Intent(in) :: rc(3,n)
         Logical,Intent(out) :: overlap
         Real(8),Intent(out) :: potij
         Real(8) :: r12_patch(3),sq_r12_patch
         Integer,Intent(in) :: i
         Integer :: j,k,z,n_inter
         Real(8) :: dsph,dsph2,sd2
         Real(8) :: patches_c1(3,n_patches)
         Real(8) :: patches_c2(3,n_patches)
         Logical :: parallel
         overlap = .false.
         potij = 0.d0
         dsq = d*d
         lsq = l*l
         dsph = l + d
         dsph2 = dsph*dsph
         r1(:) = ri(:)
         e1(:) = ei(:)
         q1(:) = qi(:)
         halfl = 0.5d0*l
          Do j=1,n
               If (i .ne. j) then
                     r2(:) = rc(:,j)
                     r12(:) = r2(:) - r1(:)
                     e2(:) = e(:,j)
                     q2(:) = q(:,j)
                     !Minimum Image
                     r12(1) = r12(1) - boxlc(1)*Dnint(r12(1)/boxlc(1))
                     r12(2) = r12(2) - boxlc(2)*Dnint(r12(2)/boxlc(2))
                     r12(3) = r12(3) - boxlc(3)*Dnint(r12(3)/boxlc(3))
                     r12sq = r12(1)*r12(1)+r12(2)*r12(2)+r12(3)*r12(3)
                     r2(:) = r1(:) + r12(:)

         ! Check if the spheres overlap
                     If (r12sq .le. dsq) then
                    overlap = .true.
                    return

            Else If (r12sq .lt. rcutsq) then
                    Call patches_allocation(r1,e1,patches_c1)
                    Call patches_allocation(r2,e2,patches_c2)
                    Call patches_interaction(boxlc,patches_c1,patches_c2
                       ,n_inter)
                    potij = potij - dble(n_inter)*eps
            End If
        End if
   End Do

End Subroutine


Subroutine total_potential_patchy_sphere(boxlc,rc,overlap,potij)
 Implicit None
 Real(8),Dimension(3),Intent(in) :: boxlc
 ! Matrix of Positions to be checked
 Real(8),Intent(in) :: rc(3,n)
 Logical,Intent(out) :: overlap
 Real(8),Intent(out) :: potij
 Real(8) :: r12_patch(3),sq_r12_patch
 Integer :: i
 Integer :: j,k,z,n_inter
 Real(8) :: dsph,dsph2,sd2
 Real(8) :: patches_c1(3,n_patches)
 Real(8) :: patches_c2(3,n_patches)
 Logical :: parallel
 overlap = .false.
 potij = 0.d0
 dsq = d*d
 lsq = l*l
 dsph = l + d
 dsph2 = dsph*dsph
 halfl = 0.5d0*l
 Do i =1,n-1
        r1(:) = rc(:,i)
        e1(:) = e(:,i)
        q1(:) = q(:,i)
       Do j=i+1,n
              r2(:) = rc(:,j)
              r12(:) = r2(:) - r1(:)
              e2(:) = e(:,j)
              q2(:) = q(:,j)
              !Minimum Image
              r12(1) = r12(1) - boxlc(1)*Dnint(r12(1)/boxlc(1))
```

```
                r12(2) = r12(2) - boxlc(2)*Dnint(r12(2)/boxlc(2))
                r12(3) = r12(3) - boxlc(3)*Dnint(r12(3)/boxlc(3))
                r12sq = r12(1)*r12(1)+r12(2)*r12(2)+r12(3)*r12(3)
                r2(:) = r1(:) + r12(:)

 ! Check if the spheres overlap
                If (r12sq .le. dsq) then
                       overlap = .true.
                       return

                Else if (r12sq .lt. rcutsq) then
                       Call patches_allocation(r1,e1,patches_c1)
                       Call patches_allocation(r2,e2,patches_c2)
                       Call patches_interaction(boxlc,patches_c1,patches_c2
                          ,n_inter)
                       potij = potij - dble(n_inter)*eps
                End If
  End Do
  End Do

End Subroutine


subroutine cylinder_shell_shortest_distance(sd_shell)
real(8) :: unit_r12(3),cos1,cos2,sin1,sin2
real(8) :: cosr,x1,x2,absr12
real(8),intent(out) :: sd_shell

absr12 = dsqrt(r12sq)
!Limiting angle
cosr = aspect_ratio/(dsqrt(1.d0 + aspect_ratio*aspect_ratio))
unit_r12(:) = r12(:)/absr12
cos1 = unit_r12(1)*e1(1) + unit_r12(2)*e1(2) + unit_r12(3)*e1(3)
cos2 = unit_r12(1)*e2(1) + unit_r12(2)*e2(2) + unit_r12(3)*e2(3)
cos1 = dabs(cos1)
cos2 = dabs(cos2)

if (cos1 .gt. cosr) then
       x1 = halfl/cos1
else
       sin1 = dsqrt(1.d0 - cos1*cos1)
       x1 = halfd/sin1
end if

if (cos2 .gt. cosr) then
       x2 = halfl/cos2
else
       sin2 = dsqrt(1.d0 - cos2*cos2)
       x2 = halfd/sin2
end if

 sd_shell = x1 + x2

end subroutine

! Reduced potential (u/kb_cgs*T)
function dlvo_pot(sd_shell,dist)
 implicit none
 real(8) :: dist,sd_shell
 sd_shell = sd_shell/d
 dist = dsqrt(dist)
 dist = dist/d
 dlvo_pot = (z1*z1/((1.d0+0.5d0*zeta*sd_shell)**2.d0))*dexp(-zeta*(dist -
     sd_shell))/dist/T_star
end function

function yukawa_pot(sd_shell,dist)
 implicit none
 real(8) :: dist,sd_shell
 sd_shell = sd_shell/d
 dist = dsqrt(dist)
 dist = dist/d
 yukawa_pot = dexp(-zeta*(dist - sd_shell))/dist/T_star
end function


Subroutine total_potential_dlvo_hc(boxlc,rc,overlap,potij)
 Implicit None
 Real(8),Dimension(3),Intent(in) :: boxlc
 ! Matrix of Positions to be checked
 Real(8),Intent(in) :: rc(3,n)
 Logical,Intent(out) :: overlap
 Real(8),Intent(out) :: potij
 Real(8) :: sd_shell,r12_patch(3),sq_r12_patch
 Integer :: i
 Integer :: j,k,z,n_inter
 Real(8) :: dsph,dsph2,sd2
 Logical :: parallel
 overlap = .false.
 potij = 0.d0
 dsq = d*d
 lsq = l*l
 dsph = l + d
 dsph2 = dsph*dsph
 halfl = 0.5d0*l
 Do i =1,n-1
```

```
        r1(:) = rc(:,i)
        e1(:) = e(:,i)
        q1(:) = q(:,i)
       Do j=i+1,n
              r2(:) = rc(:,j)
              r12(:) = r2(:) - r1(:)
              e2(:) = e(:,j)
              q2(:) = q(:,j)
              !Minimum Image
              r12(1) = r12(1) - boxlc(1)*Dnint(r12(1)/boxlc(1))
              r12(2) = r12(2) - boxlc(2)*Dnint(r12(2)/boxlc(2))
              r12(3) = r12(3) - boxlc(3)*Dnint(r12(3)/boxlc(3))
              r12sq = r12(1)*r12(1)+r12(2)*r12(2)+r12(3)*r12(3)
              r2(:) = r1(:) + r12(:)

 ! Check if the spheres overlap
              If (r12sq .le. dsph2) then
 ! Check overlap of spherocylinders
                     Call shortest_distance(sd2,parallel)
                     If (sd2 .le. dsq) then
                            Call overlap_cylinder(sd2,parallel,overlap)
                            If(overlap) return
                     End If

              End If

              If (.not. overlap) then
                     Call cylinder_shell_shortest_distance(sd_shell)
                     potij = potij + dlvo_pot(sd_shell,r12sq)
              End If
  End Do
  End Do

End Subroutine

Subroutine partial_potential_dlvo_hc(boxlc,i,qi,ei,ri,rc,overlap,potij)
 Implicit None
 Real(8),Dimension(3),Intent(in) :: ri,ei,boxlc
 Real(8),Dimension(0:3),Intent(in) :: qi
 ! Matrix of Positions to be checked
 Real(8),Intent(in) :: rc(3,n)
 Logical,Intent(out) :: overlap
 Real(8),Intent(out) :: potij
 Integer,Intent(in) :: i
 Integer :: j,k,z,n_inter
 Real(8) :: sd_shell,dsph,dsph2,sd2
 Logical :: parallel
 overlap = .false.
 potij = 0.d0
 dsq = d*d
 lsq = l*l
 dsph = l + d
 dsph2 = dsph*dsph
 r1(:) = ri(:)
 e1(:) = ei(:)
 q1(:) = qi(:)
 halfl = 0.5d0*l
  Do j=1,n
        If (i .ne. j) then
               r2(:) = rc(:,j)
               r12(:) = r2(:) - r1(:)
               e2(:) = e(:,j)
               q2(:) = q(:,j)
               !Minimum Image
               r12(1) = r12(1) - boxlc(1)*Dnint(r12(1)/boxlc(1))
               r12(2) = r12(2) - boxlc(2)*Dnint(r12(2)/boxlc(2))
               r12(3) = r12(3) - boxlc(3)*Dnint(r12(3)/boxlc(3))
               r12sq = r12(1)*r12(1)+r12(2)*r12(2)+r12(3)*r12(3)
               r2(:) = r1(:) + r12(:)

 ! Check if the spheres overlap
               If (r12sq .le. dsph2) then
 ! Check overlap of spherocylinders
                      Call shortest_distance(sd2,parallel)
                      If (sd2 .le. dsq) then
                             Call overlap_cylinder(sd2,parallel,overlap)
                             If(overlap) return
                      End If

               End If

               If (.not. overlap) then
                      Call cylinder_shell_shortest_distance(sd_shell)
                      potij = potij + dlvo_pot(sd_shell,r12sq)
               End If
        End if
   End Do

End Subroutine

Subroutine total_potential_yukawa_hc(boxlc,rc,overlap,potij)
 Implicit None
 Real(8),Dimension(3),Intent(in) :: boxlc
 ! Matrix of Positions to be checked
 Real(8),Intent(in) :: rc(3,n)
 Logical,Intent(out) :: overlap
```

```fortran
Real(8),Intent(out) :: potij
Real(8) :: sd_shell,r12_patch(3),sq_r12_patch
Integer :: i
Integer :: j,k,z,n_inter
Real(8) :: dsph,dsph2,sd2
Logical :: parallel
overlap = .false.
potij = 0.d0
dsq = d*d
lsq = l*l
dsph = l + d
dsph2 = dsph*dsph
halfl = 0.5d0*l
Do i =1,n-1
       r1(:) = rc(:,i)
       e1(:) = e(:,i)
       q1(:) = q(:,i)
      Do j=i+1,n
             r2(:) = rc(:,j)
             r12(:) = r2(:) - r1(:)
             e2(:) = e(:,j)
             q2(:) = q(:,j)
             !Minimum Image
             r12(1) = r12(1) - boxlc(1)*Dnint(r12(1)/boxlc(1))
             r12(2) = r12(2) - boxlc(2)*Dnint(r12(2)/boxlc(2))
             r12(3) = r12(3) - boxlc(3)*Dnint(r12(3)/boxlc(3))
             r12sq = r12(1)*r12(1)+r12(2)*r12(2)+r12(3)*r12(3)
             r2(:) = r1(:) + r12(:)

! Check if the spheres overlap
             If (r12sq .le. dsph2) then
! Check overlap of spherocylinders
                    Call shortest_distance(sd2,parallel)
                    If (sd2 .le. dsq) then
                           Call overlap_cylinder(sd2,parallel,overlap)
                           If(overlap) return
                    End If

             End If

             If (.not. overlap) then
                    Call cylinder_shell_shortest_distance(sd_shell)
                    potij = potij + yukawa_pot(sd_shell,r12sq)
             End If
  End Do
  End Do

End Subroutine
```

```
Subroutine partial_potential_yukawa_hc(boxlc,i,qi,ei,ri,rc,overlap,potij)
 Implicit None
 Real(8),Dimension(3),Intent(in) :: ri,ei,boxlc
 Real(8),Dimension(0:3),Intent(in) :: qi
 ! Matrix of Positions to be checked
 Real(8),Intent(in) :: rc(3,n)
 Logical,Intent(out) :: overlap
 Real(8),Intent(out) :: potij
 Integer,Intent(in) :: i
 Integer :: j,k,z,n_inter
 Real(8) :: sd_shell,dsph,dsph2,sd2
 Logical :: parallel
 overlap = .false.
 potij = 0.d0
 dsq = d*d
 lsq = l*l
 dsph = l + d
 dsph2 = dsph*dsph
 r1(:) = ri(:)
 e1(:) = ei(:)
 q1(:) = qi(:)
 halfl = 0.5d0*l
  Do j=1,n
        If (i .ne. j) then
               r2(:) = rc(:,j)
               r12(:) = r2(:) - r1(:)
               e2(:) = e(:,j)
               q2(:) = q(:,j)
               !Minimum Image
               r12(1) = r12(1) - boxlc(1)*Dnint(r12(1)/boxlc(1))
               r12(2) = r12(2) - boxlc(2)*Dnint(r12(2)/boxlc(2))
               r12(3) = r12(3) - boxlc(3)*Dnint(r12(3)/boxlc(3))
               r12sq = r12(1)*r12(1)+r12(2)*r12(2)+r12(3)*r12(3)
               r2(:) = r1(:) + r12(:)

 ! Check if the spheres overlap
               If (r12sq .le. dsph2) then
 ! Check overlap of spherocylinders
                      Call shortest_distance(sd2,parallel)
                      If (sd2 .le. dsq) then
                             Call overlap_cylinder(sd2,parallel,overlap)
                             If(overlap) return
                      End If

               End If

               If (.not. overlap) then
```

```
                      Call cylinder_shell_shortest_distance(sd_shell)
                      potij = potij + yukawa_pot(sd_shell,r12sq)
               End If
          End if
     End Do

  End Subroutine
  Subroutine total_potential_dlvo_sphere(boxlc,rc,overlap,potij)
   Implicit None
   Real(8),Dimension(3),Intent(in) :: boxlc
   ! Matrix of Positions to be checked
   Real(8),Intent(in) :: rc(3,n)
   Logical,Intent(out) :: overlap
   Real(8),Intent(out) :: potij
   Real(8) :: sd_shell,r12_patch(3),sq_r12_patch
   Integer :: i
   Integer :: j,k,z,n_inter
   Real(8) :: dsph,dsph2,sd2
   Logical :: parallel
   overlap = .false.
   potij = 0.d0
   dsq = d*d
   lsq = l*l
   dsph = l + d
   dsph2 = dsph*dsph
   halfl = 0.5d0*l
   Do i =1,n-1
          r1(:) = rc(:,i)
          e1(:) = e(:,i)
          q1(:) = q(:,i)
         Do j=i+1,n
                r2(:) = rc(:,j)
                r12(:) = r2(:) - r1(:)
                e2(:) = e(:,j)
                q2(:) = q(:,j)
                !Minimum Image
                r12(1) = r12(1) - boxlc(1)*Dnint(r12(1)/boxlc(1))
                r12(2) = r12(2) - boxlc(2)*Dnint(r12(2)/boxlc(2))
                r12(3) = r12(3) - boxlc(3)*Dnint(r12(3)/boxlc(3))
                r12sq = r12(1)*r12(1)+r12(2)*r12(2)+r12(3)*r12(3)
                r2(:) = r1(:) + r12(:)

   ! Check if the spheres overlap
                If (r12sq .le. dsq) then
                       overlap = .true.
                       print*,'i',i,'j',j,'r12sq=',r12sq,'dsq=',dsq
                       return
                End If
```

```
              If (.not. overlap) then
                     potij = potij + sphere_dlvo_pot(r12sq)
              End If
   End Do
   End Do

End Subroutine

Subroutine partial_potential_dlvo_sphere(boxlc,i,qi,ei,ri,rc,overlap,
   potij)
 Implicit None
 Real(8),Dimension(3),Intent(in) :: ri,ei,boxlc
 Real(8),Dimension(0:3),Intent(in) :: qi
 ! Matrix of Positions to be checked
 Real(8),Intent(in) :: rc(3,n)
 Logical,Intent(out) :: overlap
 Real(8),Intent(out) :: potij
 Integer,Intent(in) :: i
 Integer :: j,k,z,n_inter
 Real(8) :: sd_shell,dsph,dsph2,sd2
 Logical :: parallel
 overlap = .false.
 potij = 0.d0
 dsq = d*d
 lsq = l*l
 dsph = l + d
 dsph2 = dsph*dsph
 r1(:) = ri(:)
 e1(:) = ei(:)
 q1(:) = qi(:)
 halfl = 0.5d0*l
  Do j=1,n
       If (i .ne. j) then
              r2(:) = rc(:,j)
              r12(:) = r2(:) - r1(:)
              e2(:) = e(:,j)
              q2(:) = q(:,j)
              !Minimum Image
              r12(1) = r12(1) - boxlc(1)*Dnint(r12(1)/boxlc(1))
              r12(2) = r12(2) - boxlc(2)*Dnint(r12(2)/boxlc(2))
              r12(3) = r12(3) - boxlc(3)*Dnint(r12(3)/boxlc(3))
              r12sq = r12(1)*r12(1)+r12(2)*r12(2)+r12(3)*r12(3)
              r2(:) = r1(:) + r12(:)

 ! Check if the spheres overlap
              If (r12sq .le. dsq) then
                     overlap = .true.
```

```
                      return
                End If

                If (.not. overlap) then
                      potij = potij + sphere_dlvo_pot(r12sq)
                End If
          End if
   End Do

End Subroutine

! Reduced potential (u/kb_cgs*T)
function sphere_dlvo_pot(dist)
 implicit none
 real(8) :: dist
 dist = dsqrt(dist)
 dist = dist/d
 sphere_dlvo_pot = yuk_cte*dexp(-zeta*(dist - 1.d0))/dist

end function


End module
```

C.2.5 Ovito modifier

```
############################################################################
# Ovito modified to colour particles depending on their orientation
# relative to the phase director
############################################################################
# Developer: Joyce Tavares Lopes
############################################################################

from ovito.modifiers import PythonScriptModifier
from ovito.data import ParticleProperty
from ovito.io import import_file
import numpy as np
import math
import ovito

def modify(frame,input,output):
    #lists(vectors) starts from indice 0

# INPUT DATA
#############
    particle_position = input.particle_properties.position.array
    particle_orientation = input.particle_properties.orientation.array
    n = input.number_of_particles
    rx = np.zeros((n))
    ry = np.zeros((n))
    rz = np.zeros((n))
    qx = np.zeros((n))
    qy = np.zeros((n))
    qz = np.zeros((n))
    qw = np.zeros((n))
    color = np.zeros((n))
    blue = np.zeros((n))

    for i in range(0,n):
        rx[i] = particle_position[i][0]
        ry[i] = particle_position[i][1]
        rz[i] = particle_position[i][2]
        qw[i] = particle_orientation[i][3]
        qx[i] = particle_orientation[i][0]
        qy[i] = particle_orientation[i][1]
        qz[i] = particle_orientation[i][2]
#############

    eigenv = [0.0,0.0,0.0]
    evec = [0.0,0.0,0.0]

    #Quaternion to Orientation
    rotM = np.zeros((3,3))
    eold = [0.0,0.0,1.0] # orientation corresponds to z-axis of the
       partcile
    e = np.zeros((3,n))



    for i in range(0,n):
        #Rotation Matrix rotM(3,3) - Allen and TilVdesley, 2th edition
           page 110, equation 3.40
        # This is actually already the transpose of the rotation matrix:
        rotM[0][0] = qw[i]*qw[i] + qx[i]*qx[i] - qy[i]*qy[i] - qz[i]*qz[i]
        rotM[0][1] = 2.00*(qx[i]*qy[i] - qw[i]*qz[i])
        rotM[0][2] = 2.00*(qx[i]*qz[i] + qw[i]*qy[i])
        rotM[1][0] = 2.00*(qx[i]*qy[i] + qw[i]*qz[i])
        rotM[1][1] = qw[i]*qw[i] - qx[i]*qx[i] + qy[i]*qy[i] - qz[i]*qz[i]
        rotM[1][2] = 2.00*(qy[i]*qz[i] - qw[i]*qx[i])
        rotM[2][0] = 2.00*(qx[i]*qz[i] - qw[i]*qy[i])
        rotM[2][1] = 2.00*(qy[i]*qz[i] + qw[i]*qx[i])
        rotM[2][2] = qw[i]*qw[i] - qx[i]*qx[i] - qy[i]*qy[i] + qz[i]*qz[i]

        #New orientation
        e[0][i] = eold[0]*rotM[0][0] + eold[1]*rotM[0][1] + eold[2]*rotM
           [0][2]
        e[1][i] = eold[0]*rotM[1][0] + eold[1]*rotM[1][1] + eold[2]*rotM
           [1][2]
        e[2][i] = eold[0]*rotM[2][0] + eold[1]*rotM[2][1] + eold[2]*rotM
           [2][2]



    Qabi = np.zeros((3,3))
    Qab = np.zeros((3,3))
    A = np.zeros((3,4))

    for i in range(0,n):
        for alpha in range(0,3):
           for beta in range(0,3):
               if alpha == beta:
                   dkrho = 1.000000
               else:
                   dkrho = 0.000
               Qabi[alpha][beta] = 1.5000000*e[alpha][i]*e[beta][i] - \
                      0.500000000*dkrho + Qabi[alpha][beta]

    Qab[0] = Qabi[0]/float(n)
    Qab[1] = Qabi[1]/float(n)
    Qab[2] = Qabi[2]/float(n)

    # Calculate EigenValue of Qab

    m = (Qab[0][0] + Qab[1][1] + Qab[2][2])/3.00

    det = (Qab[0][0]-m)*(Qab[1][1]-m)*(Qab[2][2]-m) + Qab[1][2]*Qab[0][1]*
       Qab[2][0] + \
            Qab[0][2]*Qab[1][0]*Qab[2][1] - Qab[0][2]*(Qab[1][1]-m)*Qab
                [2][0] -\
           (Qab[0][0]-m)*Qab[1][2]*Qab[2][1] - Qab[0][1]*Qab[1][0]*(Qab
               [2][2]-m)

    qq = 0.500*det
    p = (Qab[0][0]-m)*(Qab[0][0]-m)+(Qab[1][1]-m)*(Qab[1][1]-m)+(Qab
        [2][2]-m)\
           *(Qab[2][2]-m) + Qab[0][1]*Qab[0][1] + Qab[0][2]*Qab[0][2]\
           + Qab[1][0]*Qab[1][0] + Qab[1][2]*Qab[1][2] \
           + Qab[2][0]*Qab[2][0] + Qab[2][1]*Qab[2][1]
    p = p/6.00
    pq = p*p*p - qq*qq

    if pq >= 0.00:
           phi = math.atan(math.sqrt(pq)/qq)/3.00
    else:
           phi = 0.00
```

```
eigenv[0] = m + 2.00*math.sqrt(p)*math.cos(phi)
eigenv[1] = m - math.sqrt(p)*(math.cos(phi) + math.sqrt(3.00)*math.sin
    (phi))
eigenv[2] = m - math.sqrt(p)*(math.cos(phi) - math.sqrt(3.00)*math.sin
    (phi))

p2 = max(eigenv)
# Calculate Eigenvector


for i in range(0,3):
    for j in range (0,3):
                A[i][j] = Qab[i][j]

A[0][0] = A[0][0] - p2
A[1][1] = A[1][1] - p2
A[2][2] = A[2][2] - p2
A[0][3] = 0.00
A[1][3] = 0.00
A[2][3] = 0.00

# Gauss Elimination

m1 = A[1][0]/A[0][0]
m2 = A[2][0]/A[0][0]

for j in range(0,3):
        A[1][j] = A[1][j] - A[0][j]*m1
        A[2][j] = A[2][j] - A[0][j]*m2

m3 = A[2][1]/A[1][1]

for j in range(0,3):
        A[2][j] = A[2][j] - A[1][j]*m3


evec[2] = 1.00
evec[1] = [A[1][3] - evec[2]*A[1][2]]/A[1][1]
evec[0] = [A[0][3]-evec[1]*A[0][1]-evec[2]*A[0][2]]/A[0][0]
modevec = evec[0]*evec[0] + evec[1]*evec[1] + evec[2]*evec[2]
modevec = math.sqrt(modevec)
evec[0] = evec[0]/modevec
evec[1] = evec[1]/modevec
evec[2] = evec[2]/modevec
PythonScriptModifier.order_p = p2
output.attributes["order_p"] = p2
color_property = output.create_particle_property(ParticleProperty.Type
    .Color)
PythonScriptModifier.anglei = []
for i in range(0,n):
    color[i] = evec[0]*e[0][i] + evec[1]*e[1][i] + evec[2]*e[2][i]
    blue[i] = math.sqrt(color[i]*color[i])
    PythonScriptModifier.anglei.append(blue[i])
    color_property.marray[i] = (1.0 - blue[i],0.0,blue[i])
```

D

Chapter 6

D.1 Codes

The main codes on this section need three modules to run properly, as described in the appendix C.2. Each main program will generate a file with extension '.dat' and two files containing the configuration of the particles (as described in the appendix C.2).

D.1.1 NPT Monte Carlo code for hard cylinders + patches

This program requires an input file named *patchy_hc_input.sci.* An example is given below.

```
Number_of_particles=    512
Molecule_Type=      1
Cold_Configuration=     .true. ! If false it will require an initial
   configuration file
Production_run=      .false.
Reduced_pressure=     p_red ! p_red = p*D^3/epsilon
Number_of_steps=     2000000
Number_of_cycles_per_step=   1000
Print_every_x_steps=    1000
Max_rotational_displacement_rad= 0.05d0 ! Maximum rotational displacement
    in radians
Max_translational_displacement=  0.05d0
Max_volume_scaling=     0.001d0
Adjust_drmax_every_x_step=   1000
Adjust_dvmax_every_x_step=   1000
Acceptance_Ratio=     0.4
Cylinder_D_and_L=     diameter length
Well_depth_and_potential_range=  eps lambda
Reduced_temperature=    temperature
Number_of_patches_per_cylinder=  2 ! Patches on the top/bottom
```

```
Inputs_to_generate_initial_configuration_only_if_cold
Cubic_box_1_or_fcc_2=    1
Initial_Packing_Fraction=   0.02d0
Initial_Quaternion_axis=   0d0 1.d0 0.d0
Rotation_around_axis_degrees=  45.d0
```

Listing D.1: Example of the input file patchy_hc_input.sci

The output files are the following:

- patchy_hc.dat: The file is organized as follows: step, acceptance ratio (translational and rotational), acceptance ratio (volume moves), maximum translational displacement, maximum angular displacement, maximum change in volume, reduced pressure, packing fraction, nematic order parameter, x component of the phase director, y component of the phase direct or , z component of the phase director, reduced total potential, reduced temperature.
- current_displacements: containing the current maximum translational, rotational, and change in volume.
- parameters.dat

```
!************************************************************************
!************************************************************************
! NPT Monte Carlo simulation of hard cylinders + patches
!************************************************************************
! Developer: Joyce Tavares Lopes
! Supervisor: Dr. Luis Fernando Mercier Franco
! School of chemical engineering (FEQ) - UNICAMP
!************************************************************************
! This code was developed during a research period at
! Universita Ca' Foscari Venezia under the supervision of
! Professor Achille Giacometi. (SEP 2019/FEB 2020)
!************************************************************************
Program Patchy_HC_MC_NPT
  use monte_carlo
  use initial_configuration
  use global_variables
  use order_parameters
  Implicit None

  !----------------------------------------------------------------------
  ! Local Variables
  !----------------------------------------------------------------------
  ! Input for pseudo-random number generator
  !----------------------------------------------------------------------
  Integer,Dimension(:),Allocatable :: vseed
  Integer :: seed_size,seed = 349766914
  Real(8) :: rnum
  !----------------------------------------------------------------------
  ! Properties
```

```
!-----------------------------------------------------------------------
Real(8) :: p,vnew,eta0
!-----------------------------------------------------------------------
! Characterization of Particle
!-----------------------------------------------------------------------
Real(8) :: d3
!-----------------------------------------------------------------------
! Position, orientation and simulation box variables
!-----------------------------------------------------------------------
Real(8), Allocatable :: rnew(:,:)
Real(8) :: boxlnew(3),rij(3),ri(3),ei(3)
Real(8) :: quat0_axis(3),quat0_angle
Logical :: cold_conf,production
Integer :: structure
Integer :: i,j,k,jlayer
Character :: get*100,outfile*100,moltype*1
Character :: inputfile1*100
Character :: inputfile2*100,outfile2*100
Real(8) :: utotal,new_utotal
Integer :: moves_re,move_v
Integer :: step2print,stepdrmax,stepdvmax
Real(8) :: acc_mov,acc_mov_v,acc_ratio
Real(8), dimension(0:3) :: qi
Real(8) :: deltau,deltav
Real(8) :: deltah,new_pot,old_pot
Logical :: overlap
Real :: start,finish,startstep,finishstep
Real(8),dimension(0:3) :: qnew
Real(8) :: s2,evec(3)
Real :: attempts_re,dumpl,dumpd,dump
Character :: file0*1
! Patches
Real(8),allocatable :: patches(:,:)



!=======================================================================
! Initialize Random_Number (Fortran intrinsic function)
!=======================================================================

Call Random_seed(size=seed_size)
Allocate(vseed(seed_size))
vseed(:) = seed
Call Random_seed(put=vseed)

Call cpu_time(start)
```

```
!======================================================================
!Simulation Input Files
!======================================================================

!Monte Carlo simulation and Potential parameters
Open(101,File='patchy_hc_input.sci')
Read(101,*) get,n
Read(101,*) get,moltype
Read(101,*) get,cold_conf
Read(101,*) get,production
Read(101,*) get,p
Read(101,*) get,nsteps
Read(101,*) get,ncycles
Read(101,*) get,step2print
Read(101,*) get,max_angle
Read(101,*) get,max_r
Read(101,*) get,max_v
Read(101,*) get,stepdrmax
Read(101,*) get,stepdvmax
Read(101,*) get,acc_ratio
! Cylinder parameters
Read(101,*) get,d,l
Read(101,*) get,eps,lambda
Read(101,*) get,reduced_temperature
Read(101,*) get,n_patches

!======================================================================
!
!======================================================================
aspect_ratio = l/d
v_particle = 0.25d0*pi*d*d*l
halfl = 0.5d0*l
halfd = 0.5d0*d
!Variable 'space' is only used in packed box initial configuration
spacex = 0.d0
spacez = 0.d0
!======================================================================
! Initial Configuration - Warm or Cold
!======================================================================
Select Case (cold_conf)
Case(.true.)
  Read(101,*) get
  Read(101,*) get,structure
  Read(101,*) get,eta0
  Read(101,*) get,quat0_axis(1),quat0_axis(2),quat0_axis(3)
  Read(101,*) get,quat0_angle
  Select Case(structure)
  Case(1)
```

```
    Allocate(r(3,N),rnew(3,N),q(0:3,N),e(3,N),patches(3,n_patches))
    call cubic(eta0,quat0_axis,quat0_angle)
  Case(2)
    Allocate(r(3,N),rnew(3,N),q(0:3,N),e(3,N),patches(3,n_patches))
    call fcc(eta0,quat0_axis,quat0_angle)
  Case(3)
    call packed_box(eta0,quat0_axis,quat0_angle)
    Allocate(rnew(3,N),patches(3,n_patches))
  End Select
Case(.false.)
  inputfile2 = 'current_conf.xyz '
  Open(102,FILE=trim(adjustl(inputfile2)))
  Read(102,*) n
  Allocate(r(3,N),rnew(3,N),q(0:3,N),e(3,N),patches(3,n_patches))
  Read(102,*) boxl(1),boxl(2),boxl(3)
  Do 99 i=1,n
  Read(102,*) moltype,r(1,i),r(2,i),r(3,i),&
    &q(0,i),q(1,i),q(2,i),q(3,i),dumpd,dumpd,dumpl
  If (dumpl .ne. l .or. dumpd .ne. halfd) then
    print*,&
      &'initial configuration does not match&
      & cylinder specification'
    stop
  end if
  99 End Do
  Close(102)
End Select
Close(101)


!======================================================================
! Simulation Output Files
!======================================================================
1 Format(4x,A4,2x,A7,3x,a7,4x,a6,3x,a7,3x,a7,&
  &3x,a13,3x,a11,4x,a4,3x,a7,6x,a4,10x,a2,11x,a2,10x,a2)
2 Format(I7,f10.5,f10.5,f10.5,f10.5,f10.5,2x,&
  &f10.5,4x,f10.5,2x,f10.5,f10.3,f10.3,2x,f10.3,2x,f10.3,2x,f10.3)
3 Format(A13,f10.5)
4 Format(A13,I5)
!----------------------------------------------------------------------
! Configuration File - Multiple Steps
!----------------------------------------------------------------------
Open(200,File='conf.xyz')
Write(200,*) n
Write(200,*) boxl(1),boxl(2),boxl(3)
Do i=1,n
Write(200,*) moltype,r(1,i),r(2,i),r(3,i),&
  &q(0,i),q(1,i),q(2,i),q(3,i),halfd,halfd,l
```

```
End Do
Call flush(200)
!----------------------------------------------------------------------
! Current Configuration File
!----------------------------------------------------------------------
Open(203,File='current_displacements')
Open(202,File='current_conf.xyz')
!----------------------------------------------------------------------
! Properties File
!----------------------------------------------------------------------
Write(outfile2,'("PropP",f3.1,".dat")') p
Open(201,File='patchy_hc.dat',status='unknown')
!----------------------------------------------------------------------
! Simulation Parameters File
!----------------------------------------------------------------------
Open(204,file = 'parameters.dat')

!======================================================================
! Initialization
!======================================================================

v = boxl(1)*boxl(2)*boxl(3)
rho = dble(n)/v
d3 = d*d*d
max_v = max_v*d3
swrange = lambda*d
aspect_ratio = l/d

!----------------------------------------------------------------------
!
!----------------------------------------------------------------------

rcut = 4.d0*d
rcutsq = rcut*rcut
max_r = max_r*d

!----------------------------------------------------------------------
! Calculate orientations from initial quaternion rotation
!----------------------------------------------------------------------

Do 100 i=1,n !Loop over particles
Call quat_to_ori(efixed,q(:,i),e(:,i))
100 End Do

!======================================================================
! Check Initial Configuration and Calculate Total Potential
!======================================================================
Call total_potential_patchy_cylinder(boxl,r,overlap,utotal)
If (overlap) then
  Print*, 'Overlap detected in initial configuration!'
  STOP
End If

!======================================================================
! Print Initial Conditions
!======================================================================

Call orderparameter(s2,evec)

Write(*,4) 'N =',N
Write(*,3) 'L/D =',aspect_ratio
Write(*,3) 'P* =',p
Write(*,3) 'eta_i =',rho*v_particle
Write(*,3) 'rho =',rho
Write(*,3) 'T* =',reduced_temperature
Write(*,3) 'Lambda =',lambda
Write(*,3) 'eps =',eps
Write(*,3) 'U*_i =',utotal/eps/n
Write(*,3) '<P2> =',s2

Write(204,4) 'N =',N
Write(204,3) 'L/D =',aspect_ratio
Write(204,3) 'P* =',p
Write(204,3) 'eta_i =',rho*v_particle
Write(204,3) 'rho =',rho
Write(204,3) 'T* =',reduced_temperature
Write(204,3) 'Lambda =',lambda
Write(204,3) 'eps =',eps
Write(204,3) 'U*_i =',utotal/eps/n
Write(204,3) '<P2> =',s2
flush(204)


step = 0
acc_mov = 0.d0
acc_mov_v = 0.d0

Write(201,2) step,acc_mov,acc_mov_v,max_r/d,max_angle,max_V/d3,&
  &p,rho*v_particle,s2,evec(1),evec(2),evec(3),utotal/eps/n,&
  &reduced_temperature
Call flush(201)
!======================================================================
! Start Trial moves
!======================================================================
```

```fortran
!Loop over steps
Do 103 step=1,nsteps
overlap = .false.
attempts_re = 0
moves_re = 0
move_v = 0
acc_mov = 0
! Call cpu_time(startstep)
Do 104 cyclei=1,ncycles
deltah = 0.d0
Call random_number(rnum)
i = Idint(rnum*dble((n+1))) + 1

!----------------------------------------------------------------------
If (i .le. n) then
  Call partial_potential_patchy_cylinder(boxl,i,q(:,i),e(:,i)&
    &,r(:,i),r,overlap,old_pot)
  if (overlap) print*,'error with overlap from previous conf'
  attempts_re = attempts_re + 1
  !---------------------------------------------------------------------
  ! Translational and Rotational moves ||
  !--------------------------------------------------------------------

  ! Translation move for particle i

  Call new_position(i,boxl,ri)

  ! Rotational move for particle i

  ! Randomly rotate old quaternion
  Call random_rotate_quat(q(:,i),qi)
  ! New orientation after rotation
  Call quat_to_ori(efixed,qi,ei)

  ! Check Overlap and Calculate Partial Potential
  Call partial_potential_patchy_cylinder(boxl,i,qi,ei,ri&
    &,r,overlap,new_pot)


  If (.not. overlap) then
    deltau = new_pot - old_pot
    deltau = deltau/reduced_temperature
    deltau = deltau/eps
    if (deltau .lt. 0.0) then
      q(:,i) = qi(:)
      r(:,i) = ri(:)
      e(:,i) = ei(:)
      utotal = utotal + new_pot - old_pot
      moves_re = moves_re + 1
    else if ((deltau) .lt. 75) then
      Call random_number(rnum)
      if (dexp(-deltau) .gt. rnum) then
        q(:,i) = qi(:)
        r(:,i) = ri(:)
        e(:,i) = ei(:)
        utotal = utotal +&
          & new_pot - old_pot
        moves_re = moves_re + 1
      end if
    end if
  End if
  !------------------------------------------------------------
Else
  !------------------------------------------------------------
  ! Volume Move
  !------------------------------------------------------------

  Call new_volume(rnew,boxlnew,vnew)
  deltav = vnew - v
  !Check Overlap and Calculate Total Potential after volume scaling
  Call total_potential_patchy_cylinder(boxlnew,rnew,&
                                &overlap,new_utotal)
  If (.not. overlap) then
    deltau = new_utotal - utotal
    deltau = deltau/reduced_temperature
    deltau = deltau/eps
    deltah = p*deltav/d3/reduced_temperature
    deltah = deltah - (dble(n)+1d0)*dlog(vnew/v)
    deltah = deltah + deltau
    If (deltah .le. 0.0) then
      r(:,:) = rnew(:,:)
      rho = dble(n)/vnew
      boxl(:) = boxlnew(:)
      v = boxl(1)*boxl(2)*boxl(3)
      utotal = new_utotal
      move_v = 1 + move_v
    Else If ((deltah) .lt. 75) then
      Call random_number(rnum)
      If (dexp(-deltah) .gt. rnum) then
        r(:,:) = rnew(:,:)
        rho = dble(n)/vnew
        boxl(:) = boxlnew(:)
        v = boxl(1)*boxl(2)*boxl(3)
        utotal = new_utotal
        move_v = 1 + move_v
      End If
    End If
  End If
End If
104 End Do


acc_mov = dble(moves_re)/dble(attempts_re)
If (attempts_re .lt. ncycles) &
  & acc_mov_v = dble(move_v)/dble(ncycles - attempts_re)

!======================================================================
! Write properties and configuration every step2print steps
!======================================================================

If (Mod(step,step2print) == 0) then
  !----------------------------------------------------------------------
  ! Properties to print
  !----------------------------------------------------------------------
  eta = rho*v_particle
  !----------------------------------------------------------------------
  ! Order parameters
  !---------------------------------------------------------------------

  Call orderparameter(s2,evec)
  !----------------------------------------------------------------------
  ! Write Properties and Configuration Files
  !----------------------------------------------------------------------

  ! Properties File
  Write(201,2) step,acc_mov,acc_mov_v,max_r/d,&
    &max_angle,max_V/d3,&
    &p,eta,s2,evec(1),evec(2),evec(3),utotal/eps/n,&
    &reduced_temperature
  Call flush(201)
  ! Last Configuration File (position and quaternions)

  Write(203,*) 'Max Translational displacement',max_r
  Write(203,*) 'Max Rotational displacement',max_angle
  Write(203,*) 'Max Volume',max_v
  Rewind(203)
  Write(202,*) n
  Write(202,*) boxl(1),boxl(2),boxl(3)
  Do 200 i=1,n
  Write(202,*) moltype,r(1,i),r(2,i),r(3,i),q(0,i),&
    &q(1,i),q(2,i),q(3,i),halfd,halfd,l
  200 End DO
  Call flush(202)
  Call flush(203)
  Rewind(202)
  ! If (Mod(step,10*step2print) == 0) then
  ! Configuration File (position and quaternions)
  Write(200,*) n
  Write(200,*) boxl(1),boxl(2),boxl(3)
  Do 205 i=1,n
  Write(200,*) moltype,r(1,i),r(2,i),r(3,i),q(0,i),&
    &q(1,i),q(2,i),q(3,i),halfd,halfd,l

  205 End DO
  ! End If

End If
!=======================================================================
! Adjust Maximum Displacement
!=======================================================================

! Adjust max_r every stepdrmax steps
If (mod(step,stepdrmax) == 0) then
  If (acc_mov .gt. acc_ratio) then
    max_r = 1.05*max_r
    max_angle = 1.05*max_angle
  Else
    max_r = 0.95*max_r
    max_angle = 0.95*max_angle
  End If
End If

! Adjust max_v every stepdvmax steps
If (attempts_re .lt. ncycles) then
  If (mod(step,stepdvmax) == 0) then
    If (acc_mov_v .gt. acc_ratio) then
      max_v = 1.05*max_v
    Else
      max_v = 0.95*max_v
    End If
  End If
End If
103 End Do

Call cpu_time(finish)
Call orderparameter(s2,evec)

Write(*,*) '======= Final ======='
Write(*,3) 'U*_f =',utotal/eps/n
Write(*,3) 'eta_f =',eta
Write(*,3) '<P2> =',s2
Write(*,3,advance = "no") 'Run Time =',(finish-start)/60.d0
```

```
  Write(*,*) 'min'

  Write(204,*) '======= Final ======='
  Write(204,3) 'U*_f =',utotal/eps/n
  Write(204,3) 'eta_f =',eta
  Write(204,3) '<P2> =',s2
  Write(204,3,advance = "no") 'Run Time =',(finish-start)/3600.d0
  Write(204,*) 'h'


  Close(200)
  Close(201)
  Close(202)
  Close(203)
  Close(204)
  End
```

D.1.2 NPT Monte Carlo code for hard cylinders + helices

This program requires an input File named *helix_hc_input.sci*. An example is given below.

```
Number_of_particles=    512
Molecule_Type=      1
Cold_Configuration=     .true. ! If false it will require an initial
   configuration file
Production_run=      .false.
Reduced_pressure=     p_red ! p_red = p*D^3/epsilon
Number_of_steps=     2000000
Number_of_cycles_per_step=   1000
Print_every_x_steps=    1000
Max_rotational_displacement_rad= 0.05d0 ! Maximum rotational displacement
    in radians
Max_translational_displacement=  0.05d0
Max_volume_scaling=     0.001d0
Adjust_drmax_every_x_step=   1000
Adjust_dvmax_every_x_step=   1000
Acceptance_Ratio=     0.4
Cylinder_D_and_L=     diameter length
Fusion_Bead_diam_and_number_of_pitches= fusion d_b np ! Fusion between
   beads (from 0.0 to 1.0), diameter of the beads and number of pitches (
   integer).
Inputs_to_generate_initial_configuration_only_if_cold
Cubic_box_1_or_fcc_2_or_packed_3=  1
Initial_Packing_Fraction=    0.01d0
Initial_Quaternion_axis=    0d0 1.d0 0.d0
Rotation_around_axis_degrees=   45.d0
```

Listing D.2: Example of the input file helix_hc_input.sci

The output files are the following:

- hc_helices.dat: The file is organized as follows: step, acceptance ratio (translational a nd rotational), acceptance ratio (volume moves), maximum translational displacement, maximum angular displacement, maximum change in volume, reduced pressure, packing fraction, nematic order parameter, x component of the phase director, y component of the phase direct or , z component of the phase director, reduced total potential, reduced temperature.
- current_displacements: containing the current maximum translational, rotational, and change in volume.
- parameters.dat

```
!*************************************************************************
!*************************************************************************
! NPT Monte Carlo simulation of hard cylinders + helical array of beads
!*************************************************************************
! Developer: Joyce Tavares Lopes
! Supervisor: Dr. Luis Fernando Mercier Franco
! School of chemical engineering (FEQ) - UNICAMP
!*************************************************************************
! This code was developed during a research period at
! Universita Ca' Foscari Venezia under the supervision of
! Professor Achille Giacometi. (SEP 2019/FEB 2020)
!*************************************************************************
Program Helices_HC_MC_NPT
  use monte_carlo
  use initial_configuration
  use global_variables
  use order_parameters
  Implicit None
  !-----------------------------------------------------------------------
  ! Input for pseudo-random number generator
  !-----------------------------------------------------------------------
  Integer,Dimension(:),Allocatable :: vseed
  Integer :: seed_size,seed = 349766914
  Real(8) :: rnum
  !-----------------------------------------------------------------------
  ! Properties
  !-----------------------------------------------------------------------
  Real(8) :: p,vnew,eta0
  !-----------------------------------------------------------------------
  ! Characterization of Particle
  !-----------------------------------------------------------------------
  Real(8) :: d3
  !-----------------------------------------------------------------------
  ! Position, orientation and simulation box variables
  !-----------------------------------------------------------------------
```

```fortran
Real(8), Allocatable :: rnew(:,:)
Real(8) :: boxlnew(3),rij(3),ri(3),ei(3)
Real(8) :: quat0_axis(3),quat0_angle
Logical :: cold_conf,production
Integer :: structure
Integer :: i,j,k,jlayer
Character :: get*100,moltype*1,outfile*100
Character :: inputfile1*100,inputfile2*100,outfile2*100
Integer :: moves_re,move_v
Integer :: step2print,stepdrmax,stepdvmax
Real(8) :: acc_mov,acc_mov_v,acc_ratio
Real(8), dimension(0:3) :: qi
Real(8) :: deltap,deltav,deltah,new_pot,old_pot
Logical :: overlap
Real :: start,finish,startstep,finishstep
Real(8),dimension(0:3) :: qnew
! Order Parameter
Real(8) :: s2,evec(3)
Real :: attempts_re
Character :: file0*1
Real(8),allocatable :: helices(:,:)
!======================================================================
! Initialize Random_Number (Fortran intrinsic function)
!======================================================================

Call Random_seed(size=seed_size)
Allocate(vseed(seed_size))
vseed(:) = seed
Call Random_seed(put=vseed)

Call cpu_time(start)

!======================================================================
!Simulation Input Files
!======================================================================

!Monte Carlo simulation and Potential parameters
Open(101,File='helix_hc_input.sci')
Read(101,*) get,n
Read(101,*) get,moltype
Read(101,*) get,cold_conf
Read(101,*) get,production
Read(101,*) get,p
Read(101,*) get,nsteps
Read(101,*) get,ncycles
Read(101,*) get,step2print
Read(101,*) get,max_angle
Read(101,*) get,max_r
Read(101,*) get,max_v
Read(101,*) get,stepdrmax
Read(101,*) get,stepdvmax
Read(101,*) get,acc_ratio
! Cylinder parameters
Read(101,*) get,d,l
Read(101,*) get,fusion,d_bead,np
!======================================================================
!
!======================================================================
aspect_ratio = l/d
v_particle = 0.25d0*pi*d*d*l + n_beads*(pi*d_bead*d_bead*d_bead)/6.d0
halfl = 0.5d0*l
halfd = 0.5d0*d
!Variable 'space' is only used in packed box initial configuration
spacex = d_bead
spacez = d_bead

Call helices_parameters()
!======================================================================
! Initial Configuration - Warm or Cold
!======================================================================


Select Case (cold_conf)
Case(.true.)
  Read(101,*) get
  Read(101,*) get,structure
  Read(101,*) get,eta0
  Read(101,*) get,quat0_axis(1),quat0_axis(2),quat0_axis(3)
  Read(101,*) get,quat0_angle
  Select Case(structure)
  Case(1)
    Allocate(r(3,N),rnew(3,N),q(0:3,N),e(3,N),helices(3,n_beads))
    call cubic(eta0,quat0_axis,quat0_angle)
  Case(2)
    Allocate(r(3,N),rnew(3,N),q(0:3,N),e(3,N),helices(3,n_beads))
    call fcc(eta0,quat0_axis,quat0_angle)
  Case(3)
    call packed_box(eta0,quat0_axis,quat0_angle)
    Allocate(rnew(3,N),helices(3,n_beads))
  End Select
Case(.false.)
  inputfile2 = 'current_conf.xyz '
  Open(102,FILE=trim(adjustl(inputfile2)))
  Read(102,*) n
  Allocate(r(3,N),rnew(3,N),q(0:3,N),e(3,N),helices(3,n_beads))
  ! Box length (1) = x-axis, (2) = y-axis, (3) = z-axis
  Read(102,*) boxl(1),boxl(2),boxl(3)
  Do 99 i=1,n
  Read(102,*) moltype,r(1,i),r(2,i),r(3,i),&
    &q(0,i),q(1,i),q(2,i),q(3,i),halfd,halfd,l
  99 End Do
  Close(102)
End Select
Close(101)

!=======================================================================
! Simulation Output Files
!=======================================================================
! Formats
1 Format(4x,A4,2x,A7,3x,a7,4x,a6,3x,a7,3x,a7,&
  &3x,a13,3x,a11,4x,a4,3x,a7,6x,a4,10x,a2,2x,a2)
2 Format(I7,f10.5,f10.5,f10.5,f10.5,f10.5,2x,&
  &f10.5,4x,f10.5,2x,f10.5,f10.3,f10.3,f10.3)
3 Format(A13,f10.5)
4 Format(A13,I5)
!-----------------------------------------------------------------------
! Configuration File - Multiple Steps
!-----------------------------------------------------------------------
!-----------------------------------------------------------------------
Open(200,File='conf.xyz')
Write(200,*) n
Write(200,*) boxl(1),boxl(2),boxl(3)
Do i=1,n
Write(200,*) moltype,r(1,i),r(2,i),r(3,i),&
  &q(0,i),q(1,i),q(2,i),q(3,i),halfd,halfd,l
End Do
Call flush(200)
!-----------------------------------------------------------------------
! Current Configuration File
!-----------------------------------------------------------------------
Open(203,File='current_displacements')
Open(202,File='current_conf.xyz')
!-----------------------------------------------------------------------
! Properties File
!-----------------------------------------------------------------------

Write(outfile2,'("PropP",f3.1,".dat")') p
Open(201,File='hc_helices.dat',status='unknown')
!-----------------------------------------------------------------------
! Simulation Parameters File
!-----------------------------------------------------------------------
Open(204,file = 'parameters.dat')

!=======================================================================
! Initialization
!======================================================================

v = boxl(1)*boxl(2)*boxl(3)
rho = dble(n)/v
d3 = d*d*d
max_v = max_v*d3
eta = rho*v_particle
rcut = 4.d0*d
max_r = max_r*d

!----------------------------------------------------------------------
! Calculate orientations from initial quaternion rotation
!----------------------------------------------------------------------

Do 100 i=1,n !Loop over particles
Call quat_to_ori(efixed,q(:,i),e(:,i))
100 End Do

!======================================================================
! Check Overlap in Initial Configuration
!======================================================================

overlap = .false.
Do i=1,n-1
ei(:) = e(:,i)
ri(:) = r(:,i)
qi(:) = q(:,i)
Call check_overlap_helices_cylinder(boxl,i,qi,ei,ri,r,overlap)
If (overlap) then
  Print*, 'Overlap detected in initial configuration!'
  STOP
End If
End Do

!======================================================================
! Print Initial Conditions and Parameters
!======================================================================

Call orderparameter(s2,evec)

Write(*,4) 'N =',N
Write(*,4) '# pitches =',np
Write(*,4) '# beads =',n_beads
Write(*,3) 'L/D =',aspect_ratio
Write(*,3) 'D cyl. =',d
Write(*,3) 'd beads =',d_bead
Write(*,3) 'L Pitch =',pitch
Write(*,3) 'L Helix =',l_h
Write(*,3) 'P* =',p
Write(*,3) 'eta_i =',rho*v_particle
Write(*,3) 'rho =',rho
Write(*,3) '<P2> =',s2

Write(204,4) 'N =',N
Write(204,4) '# pitches =',np
Write(204,4) '# beads =',n_beads
Write(204,3) 'L/D =',aspect_ratio
Write(204,3) 'D cyl. =',d
Write(204,3) 'd beads =',d_bead
Write(204,3) 'L Pitch =',pitch
Write(204,3) 'L Helix =',l_h
Write(204,3) 'P* =',p
Write(204,3) 'eta_i =',rho*v_particle
Write(204,3) 'rho =',rho
Write(204,3) '<P2> =',s2
flush(204)


step = 0
acc_mov = 0.d0
acc_mov_v = 0.d0
Write(201,2) step,acc_mov,acc_mov_v,max_r/d,max_angle,max_V/d3,&
  &p,rho*v_particle,s2,evec(1),evec(2),evec(3)
Call flush(201)


!=======================================================================
! Start Trial moves
!=======================================================================

!Loop over steps
Do 103 step=1,nsteps
overlap = .false.
attempts_re = 0
moves_re = 0
move_v = 0
acc_mov = 0
! Call cpu_time(startstep)
Do 104 cyclei=1,ncycles
deltah = 0.d0
Call random_number(rnum)
i = Idint(rnum*dble((n+1))) + 1

!-----------------------------------------------------------------------
If (i .le. n) then
```

```
    attempts_re = attempts_re + 1
    !----------------------------------------------------------------------
    ! Translational and Rotational moves ||
    !---------------------------------------------------------------------

    ! Translation move for particle i

    Call new_position(i,boxl,ri)

    ! Rotational move for particle i

    ! Randomly rotate old quaternion
    Call random_rotate_quat(q(:,i),qi)
    ! New orientation after rotation
    Call quat_to_ori(efixed,qi,ei)

    ! Check Overlap
    Call check_overlap_helices_cylinder(boxl,i,qi,ei,ri,r,overlap)



    If (.not. overlap) then
      q(:,i) = qi(:)
      r(:,i) = ri(:)
      e(:,i) = ei(:)
      moves_re = moves_re + 1
    End if
    !---------------------------------------------------------------------
  Else
    !---------------------------------------------------------------------
    ! Volume Move
    !---------------------------------------------------------------------

    Call new_volume(rnew,boxlnew,vnew)
    deltav = vnew - v
    !Check Overlap after volume scaling
    Do i=1,n-1
    ei = e(:,i)
    ri = rnew(:,i)
    qi = q(:,i)
    Call check_overlap_helices_cylinder(boxlnew,i,qi,ei,ri,rnew,overlap)
    If (overlap) exit
    End Do
    If (.not. overlap) then
      deltah = p*deltav/d3
      deltah = deltah - (dble(n)+1d0)*dlog(vnew/v)
      If (deltah .lt. 0.0) then
        r(:,:) = rnew(:,:)
      rho = dble(n)/vnew
      boxl(:) = boxlnew(:)
      v = boxl(1)*boxl(2)*boxl(3)
      move_v = 1 + move_v
    Else If ((deltah) .lt. 75) then
      Call random_number(rnum)
      If (dexp(-deltah) .gt. rnum) then
        r(:,:) = rnew(:,:)
        rho = dble(n)/vnew
        boxl(:) = boxlnew(:)
        v = boxl(1)*boxl(2)*boxl(3)
        move_v = 1 + move_v
      End If
    End If
  End If
End If
104 End Do


acc_mov = dble(moves_re)/dble(attempts_re)
If (attempts_re .lt. ncycles) &
  & acc_mov_v = dble(move_v)/dble(ncycles - attempts_re)

!=======================================================================
! Write properties and configuration every step2print steps
!=======================================================================

If (Mod(step,step2print) == 0) then
  Call orderparameter(s2,evec)
  !---------------------------------------------------------------------
  ! Properties to print
  !---------------------------------------------------------------------
  eta = rho*v_particle
  !---------------------------------------------------------------------
  ! Order parameters
  !---------------------------------------------------------------------

  Call orderparameter(s2,evec)

  ! Properties File
  Write(201,2) step,acc_mov,acc_mov_v,max_r/d,&
    &max_angle,max_V/d3,&
    &p,eta,s2,evec(1),evec(2),evec(3)
  Call flush(201)
  ! Last Configuration File (position and quaternions)

  Write(203,*) 'Max Translational displacement',max_r
  Write(203,*) 'Max Rotational displacement',max_angle
  Write(203,*) 'Max Volume',max_v
  Rewind(203)
  Write(202,*) n
  Write(202,*) boxl(1),boxl(2),boxl(3)
  Do 200 i=1,n
  Write(202,*) moltype,r(1,i),r(2,i),r(3,i),&
    &q(0,i),q(1,i),q(2,i),q(3,i),halfd,halfd,l
  200 End DO
  Call flush(202)
  Call flush(203)
  Rewind(202)
  Rewind(203)
  If (Mod(step,10*step2print) == 0) then
    ! Configuration File (position and quaternions)
    Write(200,*) n
    Write(200,*) boxl(1),boxl(2),boxl(3)
    Do 205 i=1,n
    Write(200,*) moltype,r(1,i),r(2,i),r(3,i),&
      &q(0,i),q(1,i),q(2,i),q(3,i),halfd,halfd,l

    205 End DO
    Call flush(200)
  End If

End If
!=====================================================================
! Adjust Maximum Displacement
!=====================================================================

! Adjust max_r every stepdrmax steps
If (mod(step,stepdrmax) == 0) then
  If (acc_mov .gt. acc_ratio) then
    max_r = 1.05*max_r
    max_angle = 1.05*max_angle
  Else
    max_r = 0.95*max_r
    max_angle = 0.95*max_angle
  End If
End If

! Adjust max_v every stepdvmax steps
If (attempts_re .lt. ncycles) then
  If (mod(step,stepdvmax) == 0) then
    If (acc_mov_v .gt. acc_ratio) then
      max_v = 1.05*max_v
    Else
      max_v = 0.95*max_v
    End If
  End If
End If
103 End Do



Call cpu_time(finish)

Call orderparameter(s2,evec)

Write(*,*) '======= Final ======='
Write(*,3) 'eta_f =',eta
Write(*,3) '<P2> =',s2
Write(*,3,advance = "no") 'Run Time =',(finish-start)/60.d0
Write(*,*) 'min'

Write(204,*) '======= Final ======='
Write(204,3) 'eta_f =',eta
Write(204,3) '<P2> =',s2
Write(204,3,advance = "no") 'Run Time =',(finish-start)/60.d0
Write(204,*) 'min'

Print*,'Total run time=',(finish-start)/3600.0,'min'


Close(200)
Close(201)
Close(202)
Close(203)
Close(204)
End
```

D.1.3 NPT Monte Carlo code for hard cylinders + Cylindrical Yukawa

This program requires an input File named *yuk_hc_input.sci*. An example is given below.

```
Number_of_particles=    512
Molecule_Type=      1
Cold_Configuration=     .true. ! If false it will require an initial
   configuration file
Production_run=      .false.
Reduced_pressure=     p_red ! p_red = p*D^3/epsilon
Number_of_steps=     2000000
Number_of_cycles_per_step=   1000
Print_every_x_steps=    1000
```

```
Max_rotational_displacement_rad= 0.05d0 ! Maximum rotational displacement
    in radians
Max_translational_displacement=  0.05d0
Max_volume_scaling=     0.001d0
Adjust_drmax_every_x_step=   1000
Adjust_dvmax_every_x_step=   1000
Acceptance_Ratio=     0.4
Cylinder_D_and_L=     diameter length
Reduced_Temperture=     1d0
Reduced_Screening_Parameter=  1d0
Inputs_to_generate_initial_configuration_only_if_cold
Cubic_box_1_or_fcc_2=    2
Initial_Packing_Fraction=   0.02d0
Initial_Quaternion_axis=   0d0 1.d0 0.d0
Rotation_around_axis_degrees=  0.d0
```

Listing D.3: Example of the input file yuk_hc_input.sci

The output files are the following:

- yukawa_helices.dat: The file is organized as follows: step, acceptance ratio (translational a nd rotational), acceptance ratio (volume moves), maximum translational displacement, maximum angular displacement, maximum change in volume, reduced pressure, packing fraction, nematic order parameter, x component of the phase director, y component of the phase direct or , z component of the phase director, reduced total potential, reduced temperature.

- current_displacements: containing the current maximum translational, rotational, and change in volume.

- parameters.dat

```
!*************************************************************************
!*************************************************************************
! NPT Monte Carlo simulation of cylindrical yukawa + hard cylinders
!*************************************************************************
! Developer: Joyce Tavares Lopes
! Supervisor: Dr. Luis Fernando Mercier Franco
! School of chemical engineering (FEQ) - UNICAMP
!*************************************************************************
! This code was developed during a research period at
! Universita Ca' Foscari Venezia under the supervision of
! Professor Achille Giacometi. (SEP 2019/FEB 2020)
!*************************************************************************
Program Yukawa_HC_MC_NPT
  use monte_carlo
  use initial_configuration
  use global_variables
  use order_parameters
  Implicit None
```

```
!----------------------------------------------------------------------
! Local Variables
!----------------------------------------------------------------------
! Input for pseudo-random number generator
!----------------------------------------------------------------------
Integer,Dimension(:),Allocatable :: vseed
Integer :: seed_size,seed = 349766914
Real(8) :: rnum
!----------------------------------------------------------------------
! Properties
!----------------------------------------------------------------------
Real(8) :: p,vnew,eta0
!----------------------------------------------------------------------
! Characterization of Particle
!----------------------------------------------------------------------
Real(8) :: d3
!----------------------------------------------------------------------
! Position, orientation and simulation box variables
!----------------------------------------------------------------------
Real(8), Allocatable :: rnew(:,:)
Real(8) :: boxlnew(3),rij(3),ri(3),ei(3)
Real(8) :: quat0_axis(3),quat0_angle
Logical :: cold_conf,production
Integer :: structure
Integer :: i,j,k,jlayer
Character :: get*100,moltype*1,outfile*100
Character :: inputfile1*100
Character :: inputfile2*100,outfile2*100
Real(8) :: utotal,new_utotal,deltau
Integer :: moves_re,move_v
Integer :: step2print,stepdrmax,stepdvmax
Real(8) :: acc_mov,acc_mov_v,acc_ratio
Real(8), dimension(0:3) :: qi
Real(8) :: deltap,deltav,deltah
Real(8) :: new_pot,old_pot
Logical :: overlap
Real :: start,finish,startstep,finishstep
Real(8),dimension(0:3) :: qnew
Real(8) :: s2,evec(3)
Real :: attempts_re
Character :: file0*1

!======================================================================
! Initialize Random_Number (Fortran intrinsic function)
!======================================================================

Call Random_seed(size=seed_size)
Allocate(vseed(seed_size))
```

```
vseed(:) = seed
Call Random_seed(put=vseed)

Call cpu_time(start)

!======================================================================
!Simulation Input File
!======================================================================
!Monte Carlo simulation and Potential parameters
Open(101,File='yuk_hc_input.sci')
Read(101,*) get,n
Read(101,*) get,moltype
Read(101,*) get,cold_conf
Read(101,*) get,production
Read(101,*) get,p_star
Read(101,*) get,nsteps
Read(101,*) get,ncycles
Read(101,*) get,step2print
Read(101,*) get,max_angle
Read(101,*) get,max_r
Read(101,*) get,max_v
Read(101,*) get,stepdrmax
Read(101,*) get,stepdvmax
Read(101,*) get,acc_ratio
! Cylinder parameters
Read(101,*) get,d,l
Read(101,*) get,T_star
Read(101,*) get,zeta
!======================================================================
!
!======================================================================
aspect_ratio = l/d
v_particle = 0.25d0*pi*d*d*l
halfl = 0.5d0*l
halfd = 0.5d0*d
!Variable 'space' is only used in packed box initial configuration
spacex = 0.d0
spacez = 0.d0
!======================================================================
! Initial Configuration - Warm or Cold
!======================================================================
Select Case (cold_conf)
Case(.true.)
  Read(101,*) get
  Read(101,*) get,structure
  Read(101,*) get,eta0
  Read(101,*) get,quat0_axis(1),quat0_axis(2),quat0_axis(3)
  Read(101,*) get,quat0_angle
  Select Case(structure)
  Case(1)
    Allocate(r(3,N),rnew(3,N),q(0:3,N),e(3,N))
    call cubic(eta0,quat0_axis,quat0_angle)
  Case(2)
    Allocate(r(3,N),rnew(3,N),q(0:3,N),e(3,N))
    call fcc(eta0,quat0_axis,quat0_angle)
  Case(3)
    call packed_box(eta0,quat0_axis,quat0_angle)
    Allocate(rnew(3,N))
  End Select
Case(.false.)
  inputfile2 = 'current_conf.xyz '
  Open(102,FILE=trim(adjustl(inputfile2)))
  Read(102,*) n
  Allocate(r(3,N),rnew(3,N),q(0:3,N),e(3,N))
  Read(102,*) boxl(1),boxl(2),boxl(3)
  Do 99 i=1,n
  Read(102,*) moltype,r(1,i),r(2,i),r(3,i),&
    &q(0,i),q(1,i),q(2,i),q(3,i),halfd,halfd,l
  99 End Do
  Close(102)
End Select
Close(101)

!=====================================================================
! Simulation Output Files
!=====================================================================
! Formats
1 Format(4x,A4,2x,A7,3x,a7,4x,a6,3x,a7,3x,a7,&
  &3x,a13,3x,a11,4x,a4,3x,a7,6x,a4,10x,a2,11x,a2,10x,a2)
2 Format(I7,f10.5,f10.5,f10.5,f10.5,f10.5,2x,&
  &f10.5,4x,f10.5,2x,f10.5,f10.3,f10.3,2x,f10.3,2x,f10.3,2x,f10.3)
3 Format(A13,f10.5)
4 Format(A13,I5)
!---------------------------------------------------------------------
! Configuration File - Multiple Steps
!---------------------------------------------------------------------
Open(200,File='conf.xyz')
Write(200,*) n
Write(200,*) boxl(1),boxl(2),boxl(3)
Do i=1,n
Write(200,*) moltype,r(1,i),r(2,i),r(3,i),&
  &q(0,i),q(1,i),q(2,i),q(3,i),halfd,halfd,l
End Do
Call flush(200)
!---------------------------------------------------------------------
! Current Configuration File
```

```
!-----------------------------------------------------------------------
Open(203,File='current_displacements')
Open(202,File='current_conf.xyz')
!-----------------------------------------------------------------------
! Properties File
!-----------------------------------------------------------------------
Write(outfile2,'("PropP",f3.1,".dat")') p
Open(201,File= 'yukawa_hc.dat',status='unknown')
!-----------------------------------------------------------------------
! Simulation Parameters File
!-----------------------------------------------------------------------
Open(204,file = 'parameters.dat')

!=======================================================================
! Initialization
!=======================================================================

v = boxl(1)*boxl(2)*boxl(3)
rho = dble(n)/v
d3 = d*d*d
max_v = max_v*d3
d_cm = d*1d-8
d3_cm3 = d_cm*d_cm*d_cm
rcut = 4.d0*d
max_r = max_r*d

!=======================================================================
! Calculate orientations from initial quaternion rotation
!=======================================================================

Do 100 i=1,n !Loop over particles
Call quat_to_ori(efixed,q(:,i),e(:,i))
100 End Do

!=======================================================================
! Check Overlap in Initial Configuration
!=======================================================================

overlap = .false.

Call total_potential_yukawa_hc(boxl,r,overlap,utotal)
If (overlap) then
  Print*, 'Overlap detected in initial configuration!'
  STOP
End If

!=======================================================================
! Print Initial Conditions and Parameters
```

```
!======================================================================

Call orderparameter(s2,evec)

Write(*,4) 'N =',N
Write(*,3) 'L/D =',aspect_ratio
Write(*,3) 'P* =',p_star
Write(*,3) 'eta_i =',rho*v_particle
Write(*,3) 'rho =',rho
Write(*,3) 'T* =',T_star
Write(*,3) 'Kd* =',zeta
Write(*,3) 'eps_elec =',eps_elec
Write(*,3) 'Z =',Z1
Write(*,3) 'LB =',lb
Write(*,3) 'U*_i =',utotal*T_star/n
Write(*,3) '<P2> =',s2

Write(204,4) 'N =',N
Write(204,3) 'L/D =',aspect_ratio
Write(204,3) 'P* =',p_star
Write(204,3) 'eta_i =',rho*v_particle
Write(204,3) 'rho =',rho
Write(204,3) 'T* =',T_star
Write(204,3) 'Kd* =',zeta
Write(204,3) 'eps_elec =',eps_elec
Write(204,3) 'Z =',Z1
Write(204,3) 'LB =',lb
Write(204,3) 'U*_i =',utotal*T_star/n
Write(204,3) '<P2> =',s2
flush(204)

step = 0
acc_mov = 0.d0
acc_mov_v = 0.d0


Write(201,2) step,acc_mov,acc_mov_v,max_r/d,max_angle,max_V/d3,&
  &p_star,rho*v_particle,s2,evec(1),evec(2),evec(3),&
  &utotal*T_star/n,&
  &T_star
Call flush(201)
!======================================================================
! Start Trial moves
!======================================================================

Do 103 step=1,nsteps !Loop over steps
overlap = .false.
attempts_re = 0
```

```
moves_re = 0
move_v = 0
acc_mov = 0
Do 104 cyclei=1,ncycles !Loop over cycles
deltah = 0.d0
Call random_number(rnum)
i = Idint(rnum*dble((n+1))) + 1

!----------------------------------------------------------------------
If (i .le. n) then
  Call partial_potential_yukawa_hc(boxl,i,q(:,i),e(:,i)&
    &,r(:,i),r,overlap,old_pot)
  if (overlap) print*,'error with overlap from previous conf'
  attempts_re = attempts_re + 1
  !--------------------------------------------------------------------
  ! Translational and Rotational moves
  !--------------------------------------------------------------------
  ! Translation move for particle i

  Call new_position(i,boxl,ri)

  ! Rotational move for particle i

  ! Randomly rotate old quaternion
  Call random_rotate_quat(q(:,i),qi)
  ! New orientation after rotation
  Call quat_to_ori(efixed,qi,ei)

  ! Check Overlap and Calculate New Partial Potential
  Call partial_potential_yukawa_hc(boxl,i,qi,ei&
    &,ri,r,overlap,new_pot)



  If (.not. overlap) then
    deltau = new_pot - old_pot
    if (deltau .le. 0.0) then
      q(:,i) = qi(:)
      r(:,i) = ri(:)
      e(:,i) = ei(:)
      utotal = utotal + new_pot - old_pot
      moves_re = moves_re + 1
    else if ((deltau) .lt. 75) then
      Call random_number(rnum)
      if (dexp(-deltau) .gt. rnum) then
        q(:,i) = qi(:)
        r(:,i) = ri(:)
        e(:,i) = ei(:)
        utotal = utotal +&
          & new_pot - old_pot
        moves_re = moves_re + 1
      end if
    end if
  End if
  !-------------------------------------------------------------------
Else
  !-------------------------------------------------------------------
  ! Volume Move
  !-------------------------------------------------------------------

  Call new_volume(rnew,boxlnew,vnew)
  deltav = vnew - v
  !Check Overlap and Calculate New Total Volume after volume scaling
  Call total_potential_yukawa_hc(boxlnew,rnew,overlap,new_utotal)
  If (.not. overlap) then
    deltau = new_utotal - utotal
    deltah = p_star*deltav*ang3tocm3/d3_cm3/T_star
    deltah = deltah - (dble(n)+1d0)*dlog(vnew/v)
    deltah = deltah + deltau
    If (deltah .le. 0.0) then
      r(:,:) = rnew(:,:)
      rho = dble(n)/vnew
      boxl(:) = boxlnew(:)
      v = boxl(1)*boxl(2)*boxl(3)
      utotal = new_utotal
      move_v = 1 + move_v
    Else If ((deltah) .lt. 75) then
      Call random_number(rnum)
      If (dexp(-deltah) .gt. rnum) then
        r(:,:) = rnew(:,:)
        rho = dble(n)/vnew
        boxl(:) = boxlnew(:)
        v = boxl(1)*boxl(2)*boxl(3)
        utotal = new_utotal
        move_v = 1 + move_v
      End If
    End If
  End If
End If
104 End Do


acc_mov = dble(moves_re)/dble(attempts_re)
if (attempts_re .lt. ncycles) &
  & acc_mov_v = dble(move_v)/dble(ncycles - attempts_re)
```

```
!=======================================================================
! Write properties and configuration every step2print steps
!=======================================================================

if (mod(step,step2print) == 0) then
  !---------------------------------------------------------------------
  ! Properties to print
  !---------------------------------------------------------------------
  eta = rho*v_particle
  !---------------------------------------------------------------------
  ! Order parameters
  !---------------------------------------------------------------------

  Call orderparameter(s2,evec)
  !---------------------------------------------------------------------
  ! Write Properties and Configuration Files
  !---------------------------------------------------------------------
  ! Properties File
  Write(201,2) step,acc_mov,acc_mov_v,max_r/d,&
    &max_angle,max_V/d3,&
    &p_star,eta,s2,evec(1),evec(2),evec(3),&
    &utotal*T_star/n,&
    &T_star
  Call flush(201)
  ! Current Configuration File (position and quaternions)
  Write(203,*) 'Max Translational displacement',max_r
  Write(203,*) 'Max Rotational displacement',max_angle
  Write(203,*) 'Max Volume',max_v
  Rewind(203)
  Write(202,*) n
  Write(202,*) boxl(1),boxl(2),boxl(3)
  Do 200 i=1,n
  Write(202,*) moltype,r(1,i),r(2,i),r(3,i),q(0,i),&
    &q(1,i),q(2,i),q(3,i),halfd,halfd,l
  200 End DO
  Call flush(202)
  Call flush(203)
  Rewind(202)
  ! If (Mod(step,10*step2print) == 0) then
  ! Configuration File (position and quaternions)
  Write(200,*) n
  Write(200,*) boxl(1),boxl(2),boxl(3)
  Do 205 i=1,n
  Write(200,*) moltype,r(1,i),r(2,i),r(3,i),q(0,i),&
    &q(1,i),q(2,i),q(3,i),halfd,halfd,l

  205 End DO
  Call flush(200)
  ! End If

End If
!========================================================================
! Adjust Maximum Displacement
!========================================================================

! Adjust max_r every stepdrmax steps
If (mod(step,stepdrmax) == 0) then
  If (acc_mov .gt. acc_ratio) then
    max_r = 1.05*max_r
    max_angle = 1.05*max_angle
  Else
    max_r = 0.95*max_r
    max_angle = 0.95*max_angle
  End If
End If

! Adjust max_v every stepdvmax steps
If (attempts_re .lt. ncycles) then
  If (mod(step,stepdvmax) == 0) then
    If (acc_mov_v .gt. acc_ratio) then
      max_v = 1.05*max_v
    Else
      max_v = 0.95*max_v
    End If
  End If
End If
103 End Do



Call cpu_time(finish)
Call orderparameter(s2,evec)

Write(*,*) '======= Final ======='
Write(*,3) 'U*_f =',utotal*T_star/n
Write(*,3) 'eta_f =',eta
Write(*,3) '<P2> =',s2
Write(*,3,advance = "no") 'Run Time =',(finish-start)/60.d0
Write(*,*) 'min'

Write(204,*) '======= Final ======='
Write(204,3) 'U*_f =',utotal*T_star/n
Write(204,3) 'eta_f =',eta
Write(204,3) '<P2> =',s2
Write(204,3,advance = "no") 'Run Time =',(finish-start)/3600.d0
Write(204,*) 'h'
```

```
Close(200)
Close(201)
Close(202)
Close(203)
Close(204)
End
```