COPPE
Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia
UFRJ

SIMULAÇÃO DE SUSPENSÃO DE BACTÉRIAS UTILIZANDO *DISSIPATIVE PARTICLE DYNAMICS*

Ronaro Liziero Picoli

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Mecânica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Mecânica.

Orientador: Fernando Pereira Duda

Rio de Janeiro
Março de 2018

SIMULAÇÃO DE SUSPENSÃO DE BACTÉRIAS UTILIZANDO *DISSIPATIVE PARTICLE DYNAMICS*

Ronaro Liziero Picoli

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA MECÂNICA.

Examinada por:

Prof. Fernando Pereira Duda, D.Sc.

Prof. Fábio Antonio Tavares Ramos, D.Sc.

Prof. Frederico Wanderley Tavares, D.Sc.

Prof. Roney Leon Thompson, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2018

Picoli, Ronaro Liziero

Simulação de suspensão de bactérias utilizando *dissipative particle dynamics*/Ronaro Liziero Picoli. Rio de Janeiro: UFRJ/COPPE, 2018.

XII, 92 p.: il.; 29, 7cm.

Orientador: Fernando Pereira Duda

Dissertação (mestrado) UFRJ/COPPE/Programa de Engenharia Mecânica, 2018.

Referências Bibliográficas: 44 47

1. Suspensão de Bactérias. 2. *Dissipative Particle Dynamics*. 3. Condições de contorno de Lees-Edwards. I. Duda, Fernando Pereira. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Mecânica. III. Título.

À coexistência da paz e
harmonia
entre os diferentes tipos de
pessoas
em todos os lugares.

Agradecimentos

Primeiramente, agradeço a Deus pela graça da vida e por sempre me mostrar o caminho nos momentos mais difíceis.

Aos meus pais e a minha irmã pelo apoio, companheirismo e incentivo durante esses anos. E a todos os meus familiares pelo apoio e carinho.

Ao meu orientador Fernando Pereira Duda por toda a ajuda, dedicação, paciência, ensinamentos e oportunidades.

Ao professor Roberto Fernandes de Oliveira pelos ensinamentos, conselhos e por ter me indicado o professor Duda.

Aos professores do Programa de Engenharia Mecânica e do Programa de Engenharia Civil da COPPE pelos valiosos ensinamentos.

À secretária Vera Lucia P. S. Noronha por sempre estar disponível para resolver os meus problemas referentes ao curso.

Aos meus amigos Francisco, João Paulo, Maxwel e Weslley pela imensurável ajuda ao longo deste trabalho.

Aos meus amigos da engenharia civil pelo apoio e conversas. Em especial, agradeço a Gabriela, Jéssica e Schirley pelo imenso carinho.

Aos meus amigos da engenharia mecânica pelo apoio, ajuda e inúmeras conversas sobre engenharia e computação.

À Universidade Federal do Rio de Janeiro e a todos os seus funcionários por me permitirem estudar engenharia em um padrão elevado.

À CAPES pelo apoio financeiro.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

SIMULAÇÃO DE SUSPENSÃO DE BACTÉRIAS UTILIZANDO *DISSIPATIVE PARTICLE DYNAMICS*

Ronaro Liziero Picoli

Março/2018

Orientador: Fernando Pereira Duda

Programa: Engenharia Mecânica

O termo 'matéria ativa' é usado para designar sistemas que são formados de (ou que possuem) agentes que tenham capacidade de converter a energia armazenada ou retirada do ambiente em movimento. Esses agentes são capazes de se agrupar e produzer comportamentos emergentes, que se manifestam como diversas transições de fase diferentes nestes sistemas. Uma suspensão de bactérias pode ser considerada uma matéria ativa, que além da transição entre fases apresenta também a capacidade de alterar as propriedades reológicas da mistura. Neste trabalho, avaliamos fases dinâmicas e obtemos propriedades reológicas de suspensões de bactérias utilizando o método *Dissipative Particle Dynamics*. O método DPD foi utilizado por ser capaz descrever efeitos que ocorrem apenas na mesoescala. Para isso, modificamos os programas LAMMPS e DL_MESO para adicionar o modelo de partículas ativas ao método DPD clássico. As simulações realizadas mostram que há divergências entre o diagrama de fases obtidos pelos dois programas. Utilizamos as condições de contorno de Lees-Edwards, afim de modelar o efeito de cisalhamento na caixa de simulação do DPD. No caso de um fluido simples, verifica-se a deficiência do método DPD aumentando-se a velocidade de cisalhamento. Verificamos assim que, para o caso de suspensões de bactérias, essa deficiência persiste, o que acaba por impossibilitar uma análise ampla do efeito das partículas ativas na viscosidade.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

SIMULATION OF BACTERIAL SUSPENSION USING DISSIPATIVE PARTICLE DYNAMICS

Ronaro Liziero Picoli

March/2018

Advisor: Fernando Pereira Duda

Department: Mechanical Engineering

The term 'active matter' is used to designate systems wich are formed of (or have) agents with the capacity to convert stored or ambient energy into movement. These agents are able to cluster and produce emerging behavior, which manifest themselves as several different phase transitions in these systems. A suspension of bacteria can be considered as an active matter, which in addition to the transition between phases also has the ability to alter the rheological properties of the mixture. In this work, we evaluated dynamic phases and obtained rheological properties of suspensions of bacteria using the method Dissipative Particle Dynamics. The method DPD was used for being able to describe effects that occur only in the mesoscale. To do this, we modify the programs LAMMPS and DL_MESO to add the active particle model to the classical DPD method. The simulations carried out show that there are divergences between the phase diagram obtained by the two programs. We used the boundary conditions of Lees-Edwards, in order to model the shear effect in the simulation box of the DPD. In the case of a simple fluid, the deficiency of the DPD method is verified by increasing the shear velocity. We have thus verified that, in the case of bacterial suspensions, this deficiency persists, which impossibilitate a broad analysis of the effect of active particles on viscosity.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Com a evolução das ciências naturais e da tecnologia nas últimas décadas, problemas físicos que antes eram intangivéis passaram a ser estudados. Por exemplo, os sistemas classificados como matéria ativa, que são formados ou possuem agentes, que são capazes de converter a energia armazenada ou retirada do ambiente em um movimento ordenado (MARCHETTI *et al.*, 2013). Estes sistemas são sistemas fora do equilíbrio termodinâmico devido ao comportamento peculiar dos agentes que o constituem, sendo assim fundamentalmente diferentes dos sistemas que estão fora do equilíbrio apenas por efeito de agentes externos (*e.g.*, fluidos turbulentos).

Esse fenômeno pode ser visto em diferentes escalas: agrupamento de animais (cardume de peixes, enxame de insetos) (VICSEK e ZAFEIRIS, 2012), motores moleculares presente em células (FLETCHER e GEISSLER, 2009), suspensão de bactérias (MARCHETTI *et al.*, 2013), suspensão de partículas carregadas (DUNKEL *et al.*, 2004) e sistemas com partículas de Janus (GHOSH *et al.*, 2013).

Dentre os exemplos citados, é possivel encontrar sistemas cujos agentes ativos formem um agrupamento em que se deslocam coerentemente. E em consequência disso, os sistemas ativos são capazes de formar diferentes padrões de movimento, realizar transições entre fases dinâmicas ordenadas e desordenadas fora do equilíbrio e apresentar propriedades mecânicas e reológicas incomuns (MARCHETTI *et al.*, 2013).

O interesse nesses comportamentos levou à elaboração de modelos capazes de descrever a ativade desses agentes. Alguns desses modelos são baseados em partículas, por exemplo os apresentados em VICSEK *et al.* (1995), LEVINE *et al.* (2000) e D'ORSOGNA *et al.* (2006). Enquanto, outros descrevem o comportamento hidrodinâmico utilizando-se de modelos contínuos, como pode ser visto em TONER e TU (1995), SIMHA e RAMASWAMY (2002) e MARCHETTI *et al.* (2013).

Esses modelos são capazes de mostrar a transição entre diferentes regimes de escoamento (fases dinâmicas do sitema). Nesse aspecto, alguns pesquisadores realizaram experimentos *in vitro*, como a análise de suspensão de bactérias (CZIRÓK

et al., 1996; VOLFSON *et al.*, 2008; WENSINK *et al.*, 2012), que conseguem identificar experimentalmente essas fases.

Segundo VICSEK e ZAFEIRIS (2012) as fases identificadas pelos experimentos são a fase vorticial, turbulenta e bionemática. Na fase vorticial as bactérias se aglomeram e rodam em torno de um centro comum, podendo ser em sentido horário ou anti-horário. Na fase turbulenta o movimento das bactérias é desordenado. E a fase bionemática ocorre quando as bactérias se orientam simultaneamente para uma mesma direção. As Figuras 1.1, 1.2 e 1.3 representam essas fases respectivamente.

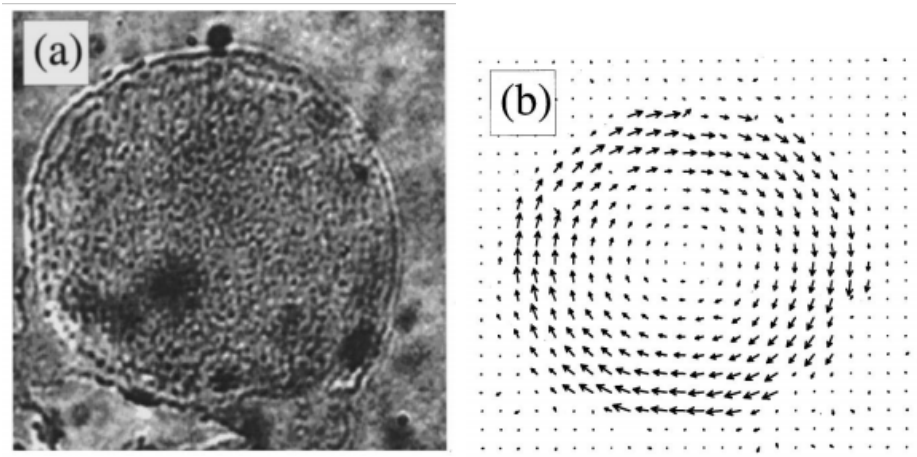


Figura 1.1: (A) Microscopia de um agrupamento de *Bacillus subtilis* em rotação com ampliação de 500x. (B) Campo de velocidades obtido através do pós-processamento do vídeo. (Figura adaptada de CZIRÓK *et al.* (1996))

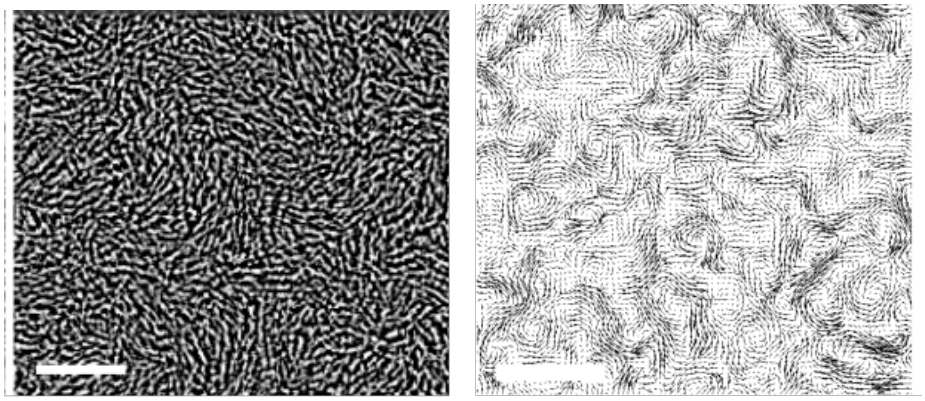


Figura 1.2: Microfotografia de uma suspensao de *Bacillus subtilis* quasi-2D. E o campo de velocides gerado usando velocimetria por imagem de partículas. (Figura adaptada de WENSINK *et al.* (2012))

Em outro experimento, LÓPEZ *et al.* (2015) avaliaram as característica reológicas de uma suspensão bacteriana com E. colis. Eles utilizaram um reômetro de Couette (esquematizado na Figura 1.4) para realizar o experimento. O reômetro é composto por um cilindro interno e outro externo capazes de rotacionar. A suspensão é colocada entre os cilindros e o externo, e a rotação é feita com uma velocidade

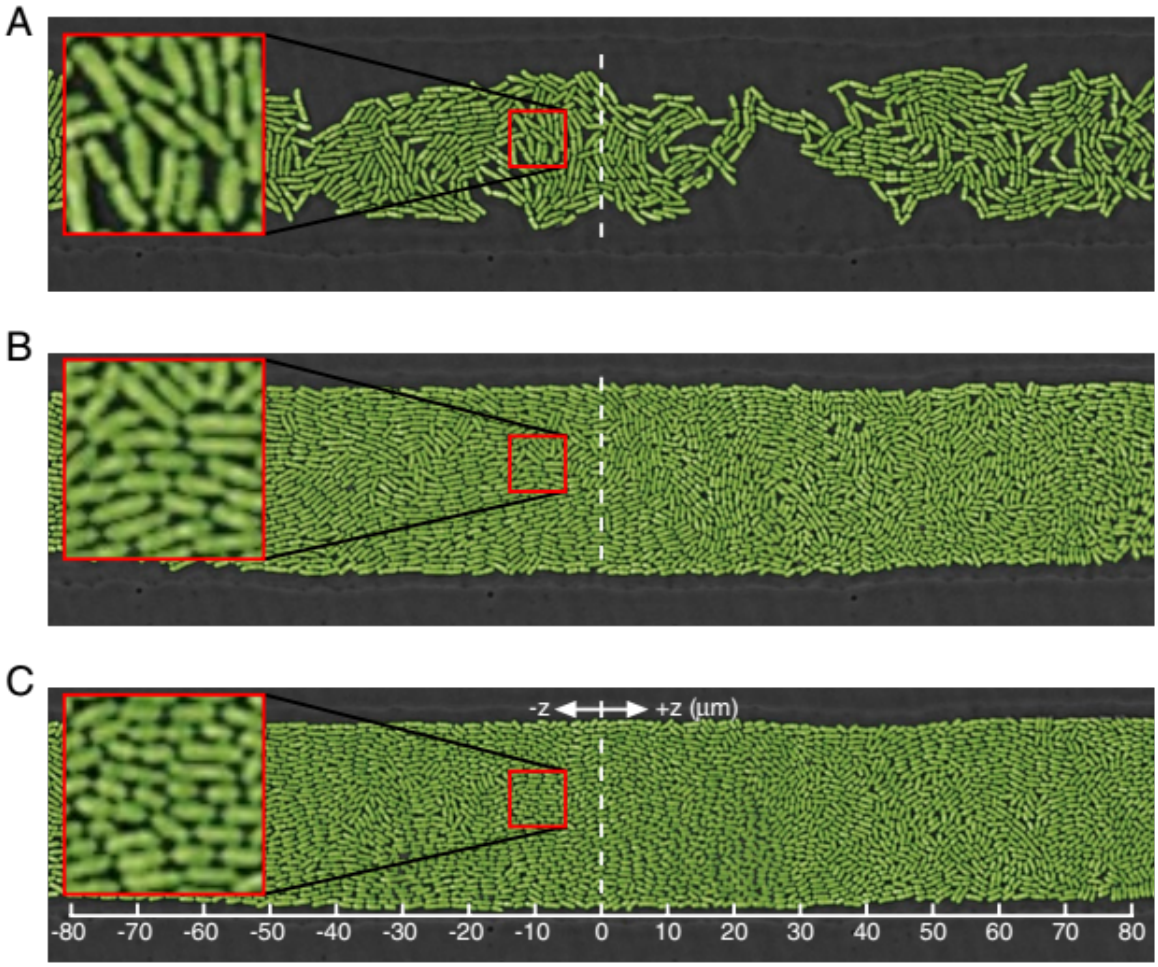


Figura 1.3: Três fotografias de uma aglomeração de *E. coli* crescendo e se alinhando em um canal microfluídico quasi-2D. As fotos foram tiradas após 60, 90, e 138 min do início do experimento. (Figura adaptada de VOLFSON *et al.* (2008))

fixa, promovendo o cisalhamento do fluido. O fluido gera um torque no cilindro interno. A partir da medição do torque gerado é possível determinar a tensão no cilindro e consequentemente a viscosidade da suspensão.

Em seus experimentos eles observaram que para altas taxas de cisalhamento a viscosidade aumenta, como ocorre em suspensões de partículas sem mobilidade (LÓPEZ *et al.*, 2015). Reduzindo-se a taxa de cisalhamento eles determinaram um valor crítico, a partir do qual a viscosidade da suspensão de bactérias começa a decair (ver Figura 1.5). E em baixas taxas de cisalhamento, a viscosidade atinge um patamar abaixo da viscosidade do fluido sem as bactérias. Notaram também que a viscosidade decresce com o aumento da concentração de bactérias.

Outro resultado obtido por LÓPEZ *et al.* (2015) é que em um estado de hiperatividade das bactérias a viscosidade do fluido decresce a zero e além disso alcança valores negativos (ver Figura 1.6). O decaimento com a concentração no estado de

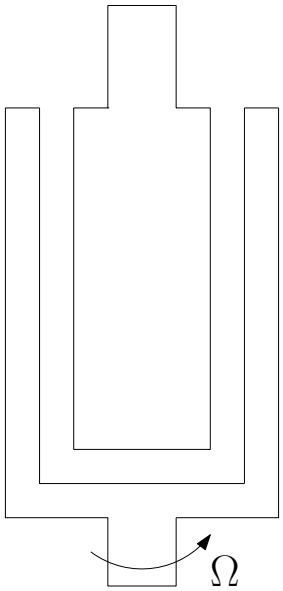


Figura 1.4: Representação de um reômetro de Couette

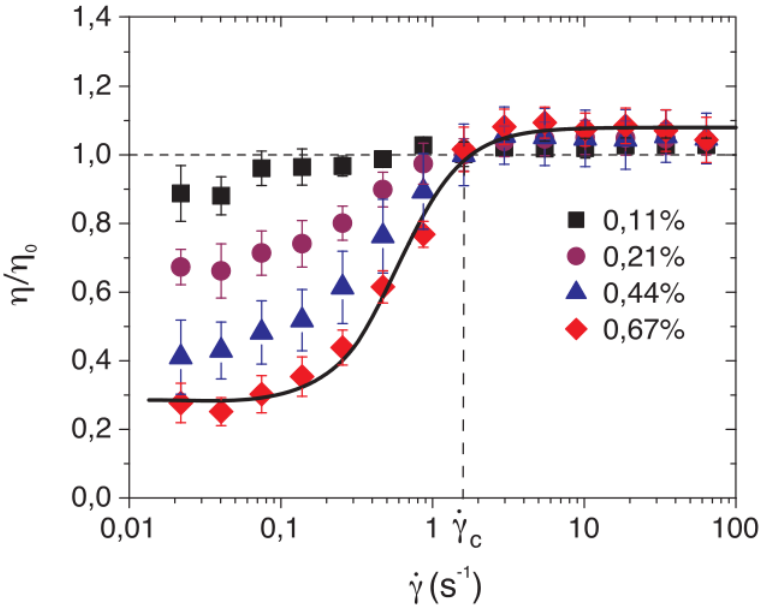


Figura 1.5: Viscosidade relativa em função da taxa de cisalhamento ($\dot{\gamma}$) para diferentes concentrações. (Retirada de LÓPEZ *et al.* (2015))

hiperatividade é mais rápido que no estado normal.

Como a viscosidade decresce, atingindo valores próximos de zero e até mesmo negativos, indica que as bactérias impulsionam o fluido a ponto de superar a dissipação provocada pela viscosidade. Sugerindo que a suspensão de bactérias pode ser usada para gerar potência e movimentar um motor rotatório.

Nesse âmbito, em HIRATSUKA *et al.* (2006) e LEONARDO *et al.* (2010) micromotores nano-fabricados são rotacionados devido à propulsão de bactérias (ver Figura 1.7). No caso de HIRATSUKA *et al.* (2006), *M. mobile* passam por um

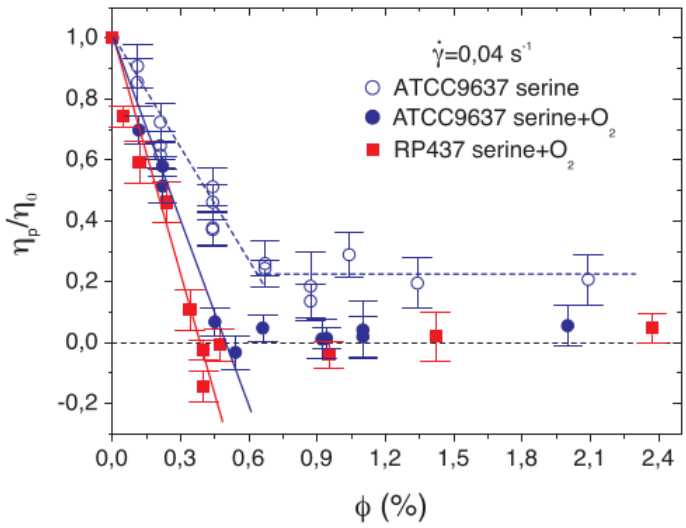


Figura 1.6: Variação da viscosidade em relação a concentração de bactérias (ϕ) em condições oxigenadas (hiperatividade) e desoxigenada. Resultados para dois tipos de bactérias *E. coli* (ATCC9637 e RP437). (Retirada de LÓPEZ *et al.* (2015))

micro-canal para que ocorra um deslocamento unidirecional das bactérias, movimentando um rotor. Em LEONARDO *et al.* (2010) engrenagens de diversos tipos são imersas em uma suspensão contendo *E. Coli*, que preenchem os espaços entre os dentes das engrenagens, o que faz com que empurrem os dentes, ocasionando o giro da engrenagem.

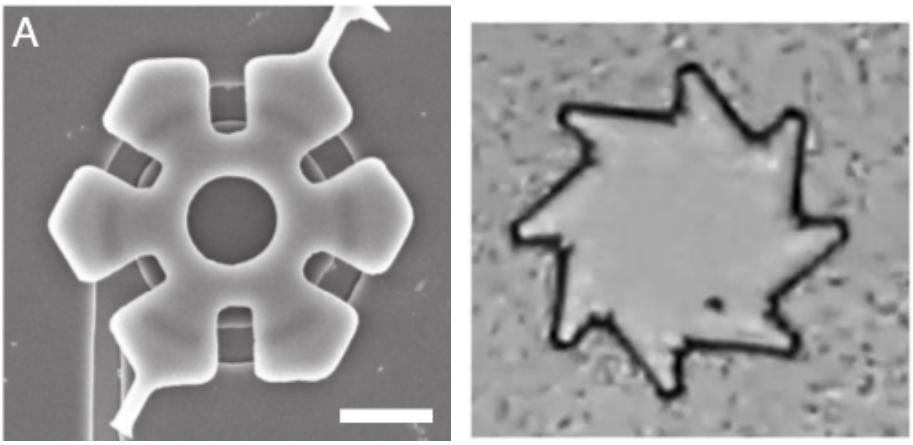


Figura 1.7: Micromotores utilizados em LEONARDO *et al.* (2010) e HIRATSUKA *et al.* (2006)

Haja visto o apelo que este sistema tem apresentado experimentalmente e as possíveis aplicações tecnológicas, pesquisadores tem cada vez mais interesse no estudo desses sistemas (MARCHETTI *et al.*, 2013; VICSEK e ZAFEIRIS, 2012).

Entre esses pesquisadores, HINZ *et al.* (2014) sugerem modificações nas formulações clássicas do *Dissipative Particles Dynamics* para incluir efeitos ocasionados

pelo movimento de agentes ativos. Através de simulações numéricas eles observam transições entre diferentes fases dinâmicas e idealizam parâmetros associados a essas transições. Além disso, eles verificam variações de difusividade no meio variando-se a concentração desses agentes.

1.1 Objetivos

O objetivo inicial deste trabalho é a avaliação das fases dinâmicas e a obtenção de propriedades reológicas de suspenção de bactérias utilizando o método *Dissipative Particles Dynamics*. Durante a implementação do método, surje como objetivo secundário a validação do método via a comparação com resultados prévios na literatura. Por fim, a análise e avaliação do método como implementado acaba por ser o foco deste trabalho. Com isso, podemos citar como objetivos específicos:

- implementar no LAMMPS e no DL_Meso as modificações propostas por HINZ *et al.* (2014), a fim de incluir a contribuição das partículas ativas no DPD;
- comparar os resultados obtidos com os de HINZ *et al.* (2014);
- utilizar as condições de contorno de Lees-Edwards para estimar a viscosidade da suspensão, a fim de verificar o comportamento da viscosidade variando-se a concentração de agentes ativos e a taxa de cisalhamento.

1.2 Organização da dissertação

Este trabalho possui 6 capítulos e 8 apêndices. No Capítulo 1 apresentamos a motivação do trabalho e os objetivos. No Capítulo 2 descrevemos o modelo Browniano para partículas ativas. No Capítulo 3 apresentamos o método DPD e as modificações necessárias para considerar partículas ativas. O Capítulo 4 é dedicado aos programas computacionais utilizados e as técnicas de pós-processamento utilizadas.

Os resultados obtidos são apresentados e discutidos no Capítulo 5. Por fim, no Capítulo 6 estarão as conclusões e sugestões para trabalhos futuros. Nos apêndices, se encontra resultados complementares e códigos computacionais desenvolvidos e exemplos dos arquivos de entrada utilizados pelas ferramentas computacionais.

Capítulo 2

Movimento Browniano

2.1 Movimento Browniano Clássico

O primeiro ciêntista a reconhecer o movimento desordenado de uma partícula foi o botânico escocês Robert Brown, enquanto observava o movimento de grão de pólen suspenso em água (BROWN, 1828).

Ele verificou que o movimento não era ocasionado pelo deslocamento ou evaporação do fluido. Essa locomoção aleatória, que ocorre devido ao impacto das móleculas do fluido no grão, é conhecida como movimento Browniano (MAZO, 2002). A Figura 2.1 esquematiza esse fenômeno.

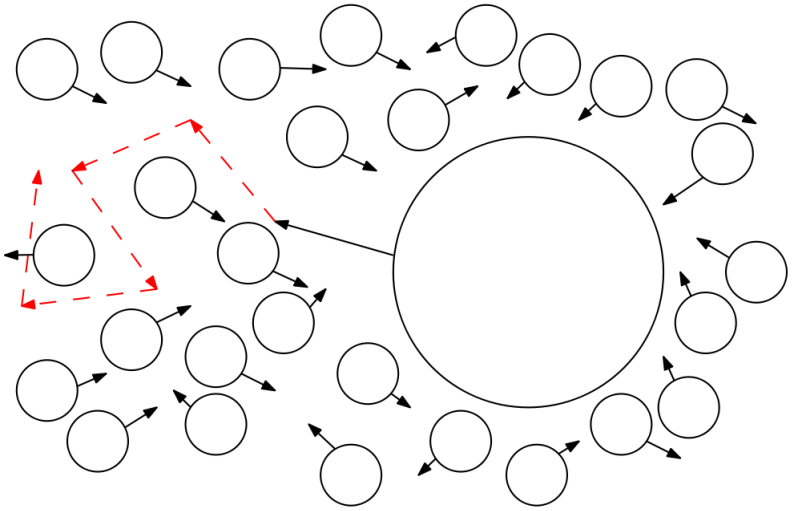


Figura 2.1: Representação do movimento Browniano. A partícula maior colide com as partículas menores, resultando em uma trajétoria aleatória (setas pontilhadas).

Em 1889, o cientista experimental Léon Gouy estudou esse fenômeno e conclui que o movimento Browniano não ocorre devido a influências externas (vibrações, campos elétricos e magnéticos). Outro fato observado é que a intensidade do movimento depende do tamanho da partícula e da temperatura. Motivado por esse

fato, em 1900, Felix Exner realizou experimentos quantitativos, medindo como o movimento depende destes dois fatores (MAZO, 2002).

Albert Einstein (EINSTEIN, 1905) também contribuiu para o entendimento do movimento Browniano. Ele utilizou a teoria molecular-cinética do calor para desenvolver uma formulação quantitaiva do movimento Brownniano. Ele obteve a relação entre constante de difusão e o deslocamento quadrático médio e também obteve uma forma de calcular o número de Avogrado a partir do deslocamento quadrático médio.

Outro cientista que estudou o movimento Browniano foi Paul Langevin (LANGEVIN, 1908). Ele propôs que o movimento é governado pela dinâmica Newtoniana sob o efeito do atrito e por uma força aleatória responsável pelo movimento desordenado. Com isso, ele utiliza a seguinte equação para descrever o movimento Browniano de uma partícula em um fluido:

$$m\frac{dv}{dt} = -F_{disp}(v,x) + \mathcal{F}(t), \tag{2.1}$$

onde m representa massa da particula e v a sua velocidade. $F_{disp}(v,x)$ representa as forças dissipativas e $\mathcal{F}(t)$ a força aleatória agindo na partícula, sendo que ambas as forças decorrem dos impactos das móleculas no seu entorno.

O modelo Browniano clássico considera que F_{disp} é uma força proporcional à velocidade (ZWANZIG (2001)):

$$F_{disp} = \zeta v. \tag{2.2}$$

onde ζ é o coeficiente de atrito.

Além disso, $\mathcal{F}(t)$ possui as seguintes propriedades:

$$\langle \mathcal{F}(t) \rangle = 0, \tag{2.3}$$

$$\langle \mathcal{F}(t)\mathcal{F}(t') \rangle = 2B\delta(t-t'). \tag{2.4}$$

onde B indica a intesidade da força aleatória e δ é a função delta de Dirac. A expressão (2.4) indica que não há correlação entre os impactos das moléculas em intervalos de tempos distintos (ZWANZIG (2001)).

A solução da equação dada por (2.1) juntamente de (2.2) é:

$$v(t) = \exp(\frac{-\zeta t}{m})v(0) + \int_0^t \exp(\frac{-\zeta(t-t')}{m})\frac{\mathcal{F}(t')}{m}dt' \tag{2.5}$$

Logo tem-se que (utilizando (2.4)) (ZWANZIG (2001)):

$$\langle v(t)^2 \rangle = \exp(\frac{-2\zeta t}{m})v(0)^2 + \frac{B}{\zeta m}(1 - \exp(\frac{-2\zeta t}{m})) \tag{2.6}$$

Dado um tempo suficientemente longo os termos exponencias sumiriam e o sistema se aproximaria do equilíbrio térmico. Sabe-se que no equilíbrio teŕmico $\langle v(t)^2 \rangle = k_B T/m$, onde k_B é a constante de Boltzmann e T a temperatura absoluta.

Assim, via (2.6):

$$B = \zeta k_B T, \tag{2.7}$$

que é conhecido como teorema de flutuação-dissipação. Essa relação mostra o balanço entre a força de dissipação e a força de flutuação no equilíbrio. Se não houvesse a força aleatória, de acordo com (2.5) a energia cinética do sistema para um tempo longo decairia a zero. A força aleatória age adicionando energia ao sistema para que a energia cinética, e consequentemente $k_B T$, se mantenha constante. Se a força aletória for grande, então a energia cinética iria aumentar progressivamente e se for pequena, o atrito faria a energia cinética decair à zero.

2.2 Movimento Browniano de uma partícula ativa

O russo Yuri L. Klimontovich foi outro pesquisador na área de movimento Browniano e estudou o movimento Browniano em plasmas, gases não ideais e problemas com radiação eletromagnética EBELING *et al.* (2005). Um dos seus objetos de estudo foi o modelo de Helmholtz-Rayleigh.

Esse modelo considera que existe um influxo de energia adicional que é capaz de intesificar o movimento da partícula, que estaria em um estado ativo. Obtém-se tal efeito, alterando a força dissipativa (F), de forma em que o atrito seja um mecanismo para a partícula ganhar energia.

Para isso Klimontovich utiliza como F_{disp}:

$$F_{disp} = \left(\alpha - \beta v^2\right) v, \tag{2.8}$$

onde α e β são constantes não negativas. Nesse modelo o sistema pode atingir uma velocidade estacionária (característica) dada por $v_0 = \sqrt{\alpha/\beta}$ (EBELING *et al.* (2005)).

Outro modelo utilizado para considerar a atividade de uma particula é encontrado em EBELING (2004). Novamente ocorre uma alteração na força de dissipação, a fim de levar em conta que a partícula possui um depósito de energia. Esse depósito pode ser alterado em três diferentes processos:

1. obter energia do meio;
2. perder energia internamente, equivalente a uma disspação interna;
3. converter energia interna em movimento.

Nesse caso a força de dissipação é dada por:

$$F_{disp} = \left(\gamma_{disp} - \frac{qd}{c + dv^2}\right) v. \tag{2.9}$$

onde γ_{disp} é uma constante positiva, q é a taxa de obteção de energia do ambiente, c a taxa de perda de energia e d a taxa de conversão de graus de liberdade da energia interna em graus de liberdade da energia cinética.

Para esse modelo, o sistema pode atingir uma velocidade estacionária (característica) dada por $v_0 = \sqrt{(q/\gamma_{disp}) - (c/d)}$.

Capítulo 3

Dissipative Particles Dynamics

3.1 Introdução

O avanço da computação durante os últimos 50 anos permitiu que métodos matemáticos possam ser usados com mais facilidade e precisão do que antigamente. Métodos que antes podiam consumir tempo excessivo, hoje são calculados de forma mais rápida e aplicados a problemas com maior complexidade. Além do aumento na eficiência dos métodos já conhecidos, os computadores permitiram o desenvolvimento de novos métodos tais como método dos elementos finitos e dinâmica molecular, entre outros. Muitos desses métodos são aplicados nas áreas de ciências naturais e engenharia. Em particular na engenharia, os métodos computacionais vem sendo amplamente utilizados. Por exemplo, usa-se algoritmos capazes de resolver equações para avaliação de tensões e deslocamentos em sólidos e fluidos, algoritmos de otimização para encontrar formatos eficientes de estruturas, e algoritmos de análise estatística para avaliar comportamento de peças. A escolha do algoritmo pode depender do fenômeno físico a ser estudado e do conhecimento que se deseja obter sobre o fenômeno.

Os problemas encontrados em ciências naturais envolvem diversas escalas espaciais e temporais. Os efeitos observados nas escalas em que a percepção humana é capaz de compreender, espacial e temporalmente, estão associados a efeitos observados em escalas menores.

Da mesma forma com que os equipamentos permitem a visualização de fenômenos em escalas diferentes, a escolha do método computacional a ser utilizado depende da escala em que o fenômeno será avaliado. Por exemplo o método da dinâmica molecular considera efeitos de interação entre partículas de um material, efeitos esses observados na micro e nanoescala. Efeitos observados em escala maiores, *e.g.* viscosidade, não são considerados pela dinâmica molecular. Em contrapartida, métodos como elementos finitos são capazes de considerar efeitos na macroescala, porém ne-

gligenciam efeitos das escalas menores. Outros métodos como Lattice-Boltzmann e o **D***issipative* **P***articles* **D***ynamics* são capazes de considerar efeitos na mesoescala.

A escala mesoscópica é definida entre a micro e a macroescala. Nas áreas de simulações da ciências dos materiais e de mecânica computacional, a mesoescala possui comprimento característico entre 10^{-7} e 10^{-4} metros e tempo característico entre 10^{-9} e 10^{-3} segundos (LIU e LIU, 2016). A Tabela 3.1 apresenta o comprimento e tempo característico para as escalas citadas.

	Comprimento (metros)	Tempo (segundos)
Macroescala	$\geq 10^{-4}$	$\geq 10^{-3}$
Mesoescala	$10^{-7} \leftrightarrow 10^{-4}$	$10^{-9} \leftrightarrow 10^{-3}$
Nanoescala	$10^{-9} \leftrightarrow 10^{-7}$	$10^{-14} \leftrightarrow 10^{-10}$
Escala quântica	$\leq 10^{-9}$	$\leq 10^{-14}$

Tabela 3.1: Comprimento e tempo característico de algumas escalas (valores retirados de LIU e LIU (2016))

Deve-se notar que para simular efeitos da mesoescala a utilização de métodos computacionais baseado em modelos atomístico, se torna impraticável, considerando-se o tempo e capacidade de simulação. Em contrapartida, em métodos computacionais baseados em modelos do contínuo (*e.g.* Elementos Finitos, Diferenças Finitas, Volumes Finitos) considerar efeitos provenientes desses agentes se torna complexo. E além disso os modelos no contínuo são inválidos em alguns sistemas na escala mesoscópica (HOOGERBRUGGE e KOELMAN, 1992; LIU e LIU, 2016; MOEENDARBARY *et al.*, 2009).

3.2 A origem do DPD

O método **D***issipative* **P***articles* **D***ynamics* foi proposto por HOOGERBRUGGE e KOELMAN (1992), a fim de simular o comportamento de fluidos complexos, como suspensões coloidais e soluções poliméricas. O DPD é similar a dinâmica molecular, porém ao invés de cada partícula representar uma mólecula ou átomo, as partículas do DPD representam um agrupamento desses corpúsculos, como mostrado na Figura 3.1. Devido a esse agrupamento o DPD negligencia detalhes das interações entre átomos e/ou moléculas, permitindo que o fenômeno físico na mesoescala seja estudado com intervalos de tempo maiores e deslocamentos maiores do que os considerados nas simulações atomística (LIU e LIU, 2016).

Após o desenvolvimento do modelo inicial, ESPAÑOL e WARREN (1995) derivaram o teorema flutuação-dissipação para o DPD, relacionando os parâmetros do modelo com a temperatura de equilíbrio esperada do sistema. E também observa-

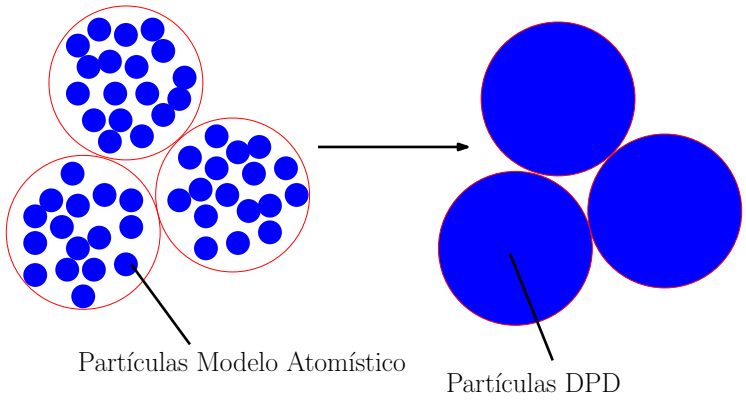


Figura 3.1: Representação de partículas no DPD

ram que erros númericos, associados ao tamanho do passo de tempo, afastavam o sistema da temperatura prescrita.

Para reduzir esses erros númericos, GROOT e WARREN (1997) modificaram o algoritmo utilizados no DPD, reduzindo também a dependência de propriedades do sistema, e.g. temperatura, do passo de tempo escolhido. Além disso, estudaram como os parâmetros deveriam ser escolhidos no DPD.

Desde então, o DPD passou a ser investigado e aplicado em problemas de fluidos complexos como: suspensão de partículas, sistemas coloidais, separação de fases de fluidos imiscíveis e misturas, evolução de filmes finos e fluidos multifásicos (LIU e LIU (2016); MOEENDARBARY *et al.* (2009)).

3.3 Formulação clássica

O sistema formado pelas partículas do DPD na mesoescala pode ser visto na macroescala representando um ponto, tal como indicado na Figura 3.2. Esse sistema evolui segundo as equações de movimento Newtonianas:

$$\mathbf{v}_i = \frac{d\mathbf{r}_i}{dt}, \tag{3.1}$$

$$m_i \frac{d\mathbf{v}_i}{dt} = \mathbf{f}_i^{\text{int}} + \mathbf{f}_i^{\text{ext}}, \tag{3.2}$$

onde m_i é a massa, $\mathbf{r}_i$ o vetor posição e $\mathbf{v}_i$ o vetor velocidade da partícula i, $\mathbf{f}_i^{\text{ext}}$ são as forças externas e $\mathbf{f}_i^{\text{int}}$ as forças de interação entre partículas atuando no centro da partícula i.

No DPD, a força de interação entre as partículas é composta por uma força

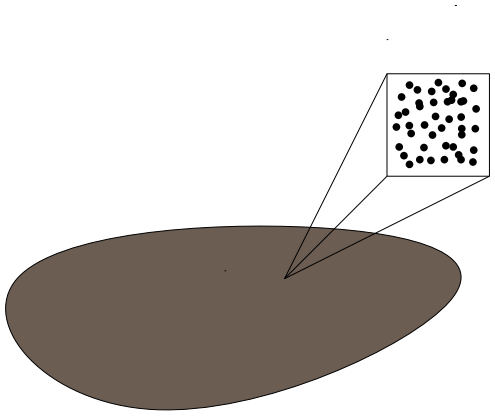


Figura 3.2: Representação da mesoescala na macroescala

conservativa $\mathbf{F}^C$, uma força dissipativa $\mathbf{F}^D$ e uma força randômica $\mathbf{F}^R$.

$$\mathbf{f}_i^{\text{int}} = \sum_{j \neq i} \mathbf{F}_{ij} = \sum_{j \neq i} \mathbf{F}_{ij}^C + \mathbf{F}_{ij}^D + \mathbf{F}_{ij}^R \tag{3.3}$$

onde $\mathbf{F}_{ij}$ é a força na partícula i devido a interação com a partícula j, que é igual em magnitude a $\mathbf{F}_{ji}$ e na direção oposta de $\mathbf{F}_{ji}$.

A força conservativa é dada por:

$$\mathbf{F}_{ij}^C = a_{ij} w^c(r) \hat{\mathbf{r}}_{ij} \tag{3.4}$$

onde a_{ij} é uma constante que representa a repulsão máxima entre as partículas e $w^c(r)$ a função de ponderação definida por:

$$w^c(r) = \begin{cases} 1 - \dfrac{r}{r_c}, & r < r_c \\ 0, & r \geq r_c. \end{cases} \tag{3.5}$$

Sendo $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$, $r = |\mathbf{r}_{ij}|$ e $\hat{\mathbf{r}}_{ij} = \dfrac{\mathbf{r}_{ij}}{r}$.

A força dissipativa é dada por:

$$\mathbf{F}_{ij}^D = -\gamma w^D(r)(\mathbf{r}_{ij} \cdot \mathbf{v}_{ij})\hat{\mathbf{r}}_{ij} \tag{3.6}$$

onde γ é uma constante, $w^D(r)$ uma função de ponderação e $\mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j$.

E a randômica é apresentada como:

$$\mathbf{F}_{ij}^R = \sigma w^R(r) \xi_{ij} \hat{\mathbf{r}}_{ij} \tag{3.7}$$

onde σ é uma constante, $w^R(r)$ uma função de ponderação e ξ_{ij} uma variável aleatória com distribuição Gaussiana que possui média igual a zero e variância igual a

1.

A força randômica representa as flutuações térmicas do sistema e aumenta a temperatura do sistema. Em contrapartida, a força dissipativa representa os efeitos viscosos agindo no sistema reduzindo a sua temperatura. O efeito combinado dessas forças mantém a temperatura do sistema próxima de uma temperatura reduzida prescrita.

Para obter esse efeito, foi mostrado por ESPAÑOL e WARREN (1995) que as forças dissipativa e randômica devem satisfazer as seguintes relações:

$$w^D(r) = [w^R(r)]^2; \tag{3.8}$$

$$\gamma = \frac{\sigma^2}{2k_BT} \tag{3.9}$$

onde k_B é a constante de Boltzman e T é a temperatura. Essas relações representam o teorema de flutuação-dissipação para o DPD.

Usualmente (GROOT e WARREN, 1997; HOOGERBRUGGE e KOELMAN, 1992; LIU e LIU, 2016):

$$w^D(r) = [w^R(r)]^2 = \begin{cases} \left(1 - \dfrac{r}{r_c}\right)^2 & r < r_c \\ 0 & r \geq r_c. \end{cases} \tag{3.10}$$

3.4 Parâmetros do DPD

Uma análise acerca dos parâmetros para o modelo DPD é realizado por GROOT e WARREN (1997). Seus resultados serão apresentados abaixo.

3.4.1 Parâmetro de Repulsão a_{ij}

O parâmetro de repulsão pode ser determinado ajustando-se a compressibilidade do fluido simulado com um fluido real. A compressibilidade adimensional do fluido é dada por:

$$k^{-1} = \frac{1}{k_BT}\left(\frac{\partial p}{\partial n}\right)_T \tag{3.11}$$

sendo que a pressão pode ser aproximada por:

$$p = \rho k_BT + \lambda a_{ij}\rho^2 \tag{3.12}$$

onde ρ é a densidade numérica e $\lambda = 0,101 \pm 0,001$. Logo, a compressibilidade adimensional é dada por:

$$k^{-1} = 1 + \frac{2\lambda a_{ij}\rho}{k_BT} \tag{3.13}$$

Sabendo-se que a compressibilidade adimensional temperatura ambiente é aproximadamente 16, tem-se que $a = 75k_BT/\rho$.

3.4.2 Parâmetro da força randômica σ

GROOT e WARREN (1997) determinaram o tamanho do passo de tempo (Δt) e o parâmetro σ avaliando a temperatura reduzida (k_BT) para diferentes valores desses parâmetros. Eles obtêm que uma boa combinação entre o tempo para o sistema alcançar a temperatura de equilíbrio determinada, a simulação ser rápida e estável, e o sistema ter sentido físico, ocorre para os valores $\Delta t = 0.04$ e $\sigma = 3$. Com a determinação de σ, o valor de γ é conhecido pela relação (3.9), para uma temperatura reduzida determinada.

3.5 Unidades Reduzidas

Unidades reduzidas são similares as unidades adimensionais. Para o DPD todos as unidades ficarão em função de um comprimento característico, massa característica e energia (ou temperatura) característica. Trabalhar em unidades reduzidas possibilita:

- usar valores numéricos de ordem unitária ao invés de valores típicos da micro e nanoescala, reduzindo-se assim erros de aproximação durante a simulação;
- reescalar os resultados para outros sistemas descritos pelo mesmo modelo.

3.6 Condições de Contorno

As simulações do DPD ocorrem no domínio conhecido como caixa de simulação, que possui dimensões L_x x L_y x L_z . As condições de contorno da caixa de simulação é uma escolha que deve ser feita visando os problemas, e as variáveis, que se deseja estudar. Nas seções 3.6.1 e 3.6.2 utiliza-se r_x, r_y, r_z para representar as componentes do vetor posição no passo atual da simulação ($\mathbf{r}_i$); e r_x^{old}, r_y^{old}, r_z^{old} as componentes do vetor posição no passo anterior da simulação ($\mathbf{r}_i^{old}$).

3.6.1 Condições periódicas de contorno

Condições periódicas de contorno são utilizadas para considerar a caixa de simulação no bulk, ou seja, as partículas do fluido estão livres de efeitos devido as partículas de superfícies ou paredes.

A Figura 3.3 representa essas condições. Para impô-las à caixa de simulação, considera-se que em torno da caixa existem réplicas idênticas colocadas lado a lado.

Os movimentos que acontecem na caixa de simulação também acontecem na réplicas. Assim quando uma partícula sai da caixa de simulação, uma outra partícula entra na caixa de simulação na face oposta, com a mesma velocidade daquela que saiu. Deve-se também considerar, respeitando o raio de corte, a interação entre as partículas dentro da caixa com as que estão fora da caixa.

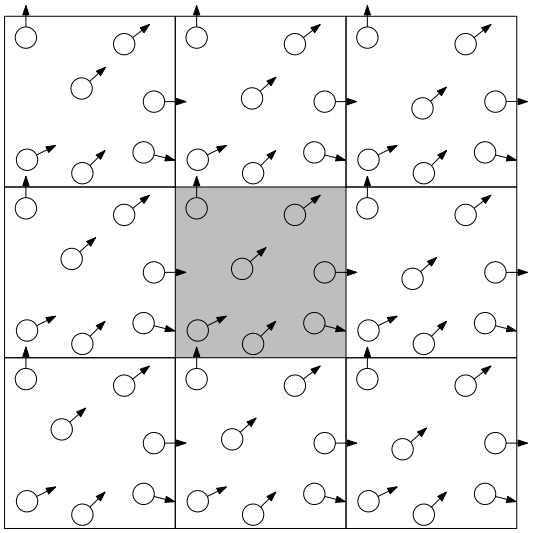


Figura 3.3: Representação das condições periódicas de contorno em duas dimensões. A caixa de simulação está em cinza e as réplicas em branco.(Adaptado de (LIU e LIU, 2016))

As partículas que saem da caixa têm suas posições atualizadas da seguinte forma (SADUS, 1999):

$$\begin{aligned} r_x &= r_x^{old} \bmod L_x, \\ r_y &= r_y^{old} \bmod L_y, \\ r_z &= r_z^{old} \bmod L_z. \end{aligned} \tag{3.14}$$

Não é necessário atualizar as suas velocidades para essa condição de contorno. Ou seja, a velocidade da partícula que reentra da caixa de simulação é a mesma com a qual saiu.

3.6.2 Condições de contorno de Lees-Edwards

Condições de Lees-Edwards (LEES e EDWARDS, 1972) são utilizadas para simular o sistema em um estado de cisalhamento. Nessas condições os efeitos de interação com superfícies não existem.

Para impor o efeito de cisalhamento, na direção x como indicado na Figura 3.4, as condições de Lees-Edwards consideram que as réplicas do sistema acima e abaixo da caixa de simulação se movem com velocidade constante em sentidos opostos.

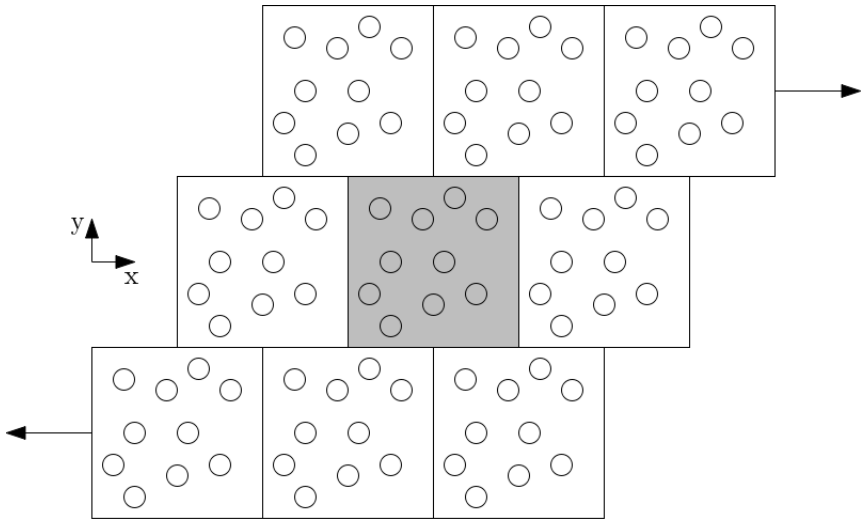


Figura 3.4: Representação das condições de Lee-Edwards em duas dimensões. A caixa de simulação está em cinza e as réplicas em branco. (Adaptado de SADUS (1999))

Deve-se atentar para as partículas que deixam a caixa de simulação. Nas faces paralelas ao eixo y, as condições de contorno periódicas são aplicadas. Nesse caso, a partícula reentraria na face oposta com a mesma velocidade daquela que saiu e a sua posição é atualizada como indicado por (3.15).

Se uma partícula deixa a caixa de simulação pelas faces paralelas ao eixo x, então uma partícula reentra na caixa pela face oposta, porém a sua posição e velocidade devem ser corrigidas para considerar que as caixas superiores e inferiores estão se movendo.

Considere o esquema apresentado na Figura 3.5. A partícula A cruza a face superior, logo ela deve reentrar pela face inferior vinda da caixa que está se deslocando com uma velocidade $-V_S$ em relação ao eixo x. Ela reentra na caixa como A' que deve ter sua posição e a componente x da velocidade modificados. Além disso, a partícula B interage com a partícula A, e a sua imagem B' interage com a partícula A'. Para calcular os efeitos devido a interações com partículas fora da caixa de simulação, altera-se as posições e velocidades das imagens para considerar o deslocamento das caixas superiores e inferiores, mantendo-se as posições relativas ente os pares A-B e A'-B' iguais.

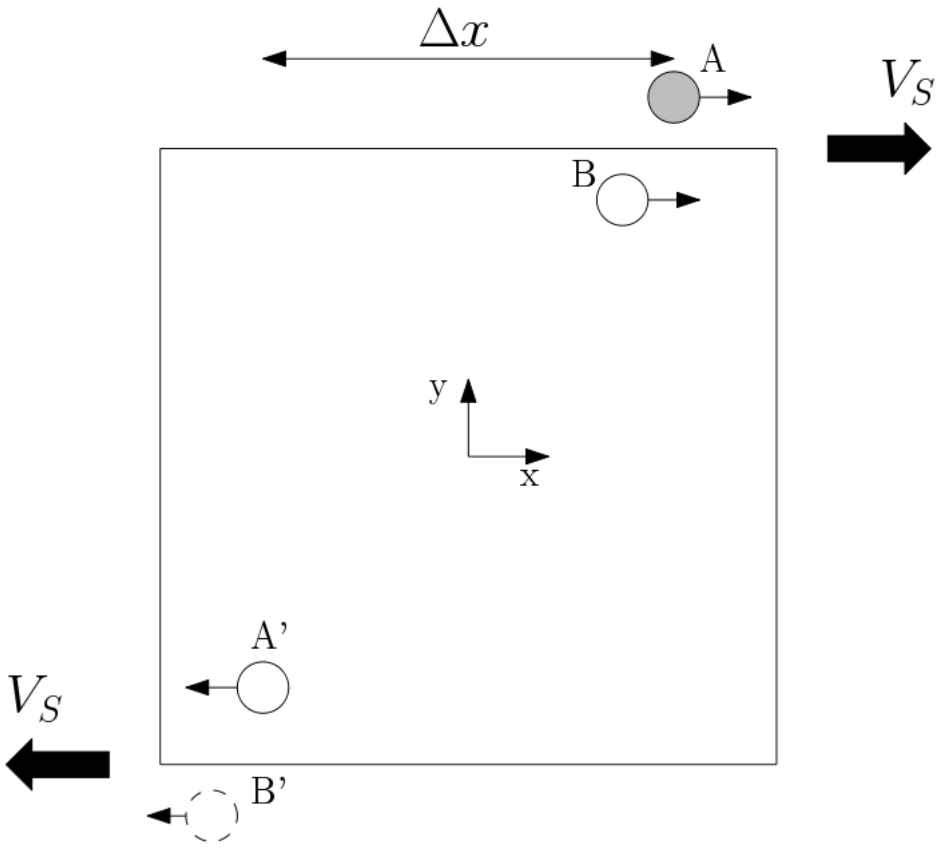


Figura 3.5: Esquema da caixa de simulação com condiços de contorno de Lee-Edwards em duas dimensões. A partícula cinza está saindo da caixa de simulação e a partícula pontilhada é a imagem de uma partícula no interior da caixa. (Adaptado de MOSHFEGH *et al.* (2015))

O deslocamento Δx das partículas A' e B' é proporcional à velocidade com que as caixas superiores e inferiores se movem (V_S) e ao tempo transcorrido (δt). A partícula que sai da caixa pelas faces perpendiculares ao eixo y e as imagens têm a suas posições e a componentes x da velocidade (V_x) atualizadas da seguinte forma (MOSHFEGH *et al.*, 2015):

$$\begin{aligned} r_x &= r_x^{old} \pm 2V_S\delta t, \\ r_y &= r_y^{old} \bmod L_y, \\ r_z &= r_z^{old} \bmod L_z, \\ V_x &= V_x^{old} \pm 2V_S. \end{aligned} \tag{3.15}$$

O sinal positivo deve ser usado para partículas que saem pela face inferior e imagens acima da caixa de simulação. Enquanto o sinal negativo é usado para partículas que saem pela face superior e imagens abaixo da caixa de simulação.

3.7 Métodos de integração para o DPD

HOOGERBRUGGE e KOELMAN (1992) e ESPAÑOL e WARREN (1995) utilizaram o método de Euler para resolver as equações (3.2) (com $\mathbf{f}_i = \mathbf{f}_i^{\text{int}} + \mathbf{f}_i^{\text{ext}}$).

Esse método é apresentado abaixo:

Algoritmo 1: Método de Euler

Entrada: condições iniciais: $\mathbf{r}_i(t_0)$ e $\mathbf{v}_i(t_0)$; passo de integração: Δt

início

  para cada *Passo de tempo* **faça**

    para $i = 1$ **até** *Total de partículas* **faça**

      Atualizar a posição: $\mathbf{r}_i = \mathbf{r}_i + \Delta t \mathbf{v}_i$

      Atualizar a velocidade: $\mathbf{v}_i = \mathbf{v}_i + \Delta t \mathbf{f}_i$

    fim

    Calcular e atualizar todos $\mathbf{f}_i$

  fim

fim

Entretanto, o DPD depende da velocidade relativa entre as partículas, o que faz a temperatura do sistema se distanciar da temperatura de equilibrio dependendo do tamanho do passo de integração.

GROOT e WARREN (1997) propuseram uma versão modificada do integrador Velocity-Verlet, onde eles realizam a predição da velocidade utilizando um parâmetro λ, corrigindo a velocidade ao final do passo de integração.

O algoritmo proposto em GROOT e WARREN (1997) é apresentado abaixo:

Algoritmo 2: Integrador de GROOT e WARREN (1997) (GW)

Entrada: condições iniciais: $\mathbf{r}_i(t_0)$ e $\mathbf{v}_i(t_0)$; passo de integração: Δt;
parâmetro λ

início

  para cada *Passo de tempo* **faça**

    para $i = 1$ **até** *Total de partículas* **faça**

      Atualizar a posição: $\mathbf{r}_i = \mathbf{r}_i + \Delta t \mathbf{v}_i + \frac{1}{2m}(\Delta t)^2 \mathbf{f}_i$

      Calcular a predição da velocidade: $\widetilde{\mathbf{v}}_i = \mathbf{v}_i + \lambda \frac{1}{m} \Delta t \mathbf{f}_i$

    fim

    Calcular e atualizar todos $\mathbf{f}_i$ utilizando $\widetilde{\mathbf{v}}_i$

    para $i = 1$ **até** *Total de partículas* **faça**

      Atualizar a velocidade: $\mathbf{v}_i = \mathbf{v}_i + \frac{1}{2m} \Delta t \mathbf{f}_i$

    fim

  fim

fim

Nesse método o parâmetro λ deve ser empiricamente determinado para cada problema. ESPAÑOL e WARREN (1995) encontraram o valor de $\lambda = 0,65$ como sendo o melhor para simulações de fluidos não-complexos, reduzindo o problema de desvio da temperatura.

GIBSON *et al.* (1999) introduzem no método de integração GW uma atualização das forças dissipativas com as velocidades atualizadas, possibilitando utilizar passos de tempo maiores e melhorando a estabilidade da temperatura. Eles sugerem que o valor $\lambda = 1$ seja o ideal para a maioria dos casos. Todavia, VATTULAINEN *et al.* (2002) argumentam que valor $\lambda = 1/2$ seria o ideal, tornado desnecessário a calibração do parâmetro. O método com o $\lambda = \frac{1}{2}$ é conhecido como DPD-VV (DPD-Velocity-Verlet).

O método de GIBSON *et al.* (1999) é dado abaixo:

Algoritmo 3: Integrador de GIBSON *et al.* (1999) (Integrador DPD-VV, caso $\lambda = 1/2$)

Entrada: condições iniciais: $\mathbf{r}_i(t_0)$ e $\mathbf{v}_i(t_0)$; passo de integração: Δt; parâmetro λ

início

  para cada *Passo de tempo* **faça**

    para $i = 1$ **até** *Total de partículas* **faça**

      Atualizar a posição: $\mathbf{r}_i = \mathbf{r}_i + \Delta t \mathbf{v}_i + \frac{1}{2m}(\Delta t)^2 \mathbf{f}_i$

      Calcular a predição da velocidade: $\widetilde{\mathbf{v}}_i = \mathbf{v}_i + \lambda \frac{1}{m} \Delta t \mathbf{f}_i$

    fim

    Calcular e atualizar todos $\mathbf{f}_i$ utilizando $\widetilde{\mathbf{v}}_i$

    para $i = 1$ **até** *Total de partículas* **faça**

      Atualizar a velocidade: $\mathbf{v}_i = \mathbf{v}_i + \frac{1}{2m} \Delta t \mathbf{f}_i$

    fim

    Calcular a componente dissipativa de todos os $\mathbf{f}_i$

  fim

fim

3.8 Tensor de tensões e temperatura do sistema

Para obter o tensor de tensões $\mathbf{S}$ associado a caixa de simulação utiliza-se a seguinte fórmula:

$$\mathbf{S} = -\frac{1}{V}\left[\sum_i m_i \mathbf{v}_i \otimes \mathbf{v}_i + \frac{1}{2}\sum_{j \neq i} \mathbf{r}_{ij} \otimes \mathbf{f}_{ij}\right] \tag{3.16}$$

onde $\mathbf{f}_{ij}$ representa a força de interação entre as partículas i e j. O símbolo $\otimes$ denota o produto tensorial entre dois vetores.

O primeiro termo é a contribuição cinética e respresenta a trasnferência de momento linear e o segundo termo é a contribução proveniente das iterações entre as partículas.

Enquanto que a temperatura do sistema é obtida por:

$$k_B T = \frac{1}{3N} \sum_i m_i \mathbf{v}_i \cdot \mathbf{v}_i \tag{3.17}$$

onde N é o número total de partículas no sistema.

3.9 DPD com partículas ativas

HINZ *et al.* (2014) propõem um modelo baseado no DPD para simular sunspensão de partículas ativas. Considera-se que o sistema é formado por partículas de fluido (passivas) e partículas ativas. Logo, deve-se considerar como a interação entre elas ocorrem e adicionar ao modelo a capacidade das partículas ativas de ganhar energia e se locomover.

A interação que ocorre entre duas partículas depende do tipo delas. Caso a interação seja entre duas partículas passivas, então considerada-se as três forcas do DPD convencional. Se a interação for entre uma partícula ativa e uma partícula passiva ou entre duas partículas ativas, então considera-se somente a força de repulsão e a força de dissipação. A Tabela 3.2 sintetiza essa ideia.

Interação	$\mathbf{F}_{ij}$ para $r < r_c$
passiva-passiva	$\mathbf{F}_{ij}^C + \mathbf{F}_{ij}^D + \mathbf{F}_{ij}^R$
ativa-ativa	$\mathbf{F}_{ij}^C + \mathbf{F}_{ij}^D$
ativa-passiva	$\mathbf{F}_{ij}^C + \mathbf{F}_{ij}^D$

Tabela 3.2: Força de interação entre pares de partículas

Para contabilizar a atividade das partículas, HINZ *et al.* (2014) adicionam o modelo de Helmholtz-Rayleigh ao modelo clássico do DPD. Estas partículas adquirem a seguinte força de propulsão:

$$\mathbf{f}_i^P = (\alpha - \beta |\mathbf{v}_i|^2)\mathbf{v}_i. \tag{3.18}$$

onde $\alpha \geq 0$ é o coeficiente de auto-propulsão e $\beta \geq 0$ é o coeficiente de atrito de

Rayleigh.

3.9.1 Número de Péclet

O número de Péclet é um número adimensional que em sistemas contínuos representa a razão entre a taxa de convecção de uma quantidade física e a taxa de difusão dessa mesma quantidade.

(HINZ *et al.*, 2014) interpretam o número de Péclet como sendo a razão entre a energia de propulsão das partículas ativas e a energia associada com as forças aleatórias. Logo, um número de Péclet pequeno indica que o sistema é dominado pelas forças aleatórias e um número de Péclet grande indicaria que as forças associadas à propulsão das partículas comandam o sistema.

O número de Péclet apresentado por (HINZ *et al.*, 2014) é dado por:

$$Pe = \frac{\alpha}{2\beta k_B T}. \tag{3.19}$$

Como será mostrado no Capítulo 5, dependendo do número de Péclet e da concentração de partículas ativas no meio, é possível obter diferentes fases dinâmicas.

Capítulo 4

Ferramentas Computacionais

Neste capítulo será apresentado os programas utilizados para simular o método DPD e os métodos que foram utlizadas para realizar o pós processamento da simulação. Todas as simulações foram realizadas em um servidor Dell PowerEdge T620 que possui 24 unidades de processamento Xeon E5-2630 2.30GHz e um total de 64GB de memória RAM. A versão do LAMMPS utilizada foi a '30Jul16' e a do DL_MESO a 2.6.

4.1 LAMMPS

Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) é um sofware de dinâmica molecular, capaz de simular sistemas atômicos, poliméricos, metálicos, granulares, entre outros. Ele foi programado na liguagem C++[1], utilizando-se do paradigma de programação orientado a objetos. Atualmente ele é distribuído como código aberto pelo Sandia National Labs(PLIMPTON *et al.*, 2005).

Segundo (PLIMPTON *et al.*, 2005) as principais características do LAMMPS são:

1. funciona em paralelo, utilizando MPI ou GPU;
2. facilidade de estender as suas funcionalidades;
3. utiliza scripts como arquivo de entrada, possuindo uma sintaxe própria;
4. é capaz de rodar multiplas simulações a partir do mesmo script;
5. pode ser usado em conjunto com outra interface, por exemplo o Python;
6. e gera arquivos de saída em formatos utilizados por outros programas de visualização (*e.g.* VMD).

[1] As versões atuais. Antes ele foi programado em FORTRAN

4.1.1 Organização estrutural

LAMMPS foi programado usando-se o paradigma de orientação à objetos. Conforme SCOTT (2000), a idéia desse paradigma é baseada na composição e interação entre diversas entidades denominados objetos. Cada objeto, é determinado pela sua classe que determina o comportamento (métodos) e os estados possíveis (atributos) de seus objetos. Mesmo que existam objetos de uma mesma classe cada objeto possui um identidade única. O funcionamento de um programa orientado a objetos ocorre pela troca de mensagens e relacionamento entre esses objetos.

Esse paradigma possibilita que modificações sejam feitas apenas em alguma classe, não se preocupando com todo o programa. Isso é possível devido a hieraquia de classes, que é uma forma de relacionamento entre as classes. Onde uma ou mais classes são derivadas de uma classe base. Logo, a classe derivada possui atributos e métodos da classe base, e cada classe derivada possui seus próprios atributos e métodos.

A hieraquia de classes do LAMMPS é apresentada na Figura 4.1. Um detalhamento das principais classes do LAMMPS pode ser encontrado em PLIMPTON (2011).

4.1.2 Modificações realizadas

Na estrutura do LAMMPS, a classe *Pair.h* é a responsável por definir o tipo de interações entre as partículas. Dela são derivadas as classes que indicam como se deve calcular as forças de interação. Por exemplo, a classe *pair_dpd.h* informa ao programa que as forças serão calculdas como na equação (3.3).

A fim de, adicionar o modelo proposto por HINZ *et al.* (2014) é necessária a criação de duas novas classes. Uma classe para representar as interações entre duas partículas ativas e outra a interação entre uma partícula ativa e uma passiva. Para isso são criadas as classes *pair_dpd_flock.h* e *pair_dpdWR.h* respectivamente.

O arquivos referentes a essas classes está no Apêndice A. E no Apêndice B se encontra um arquivo de entrada do LAMMPS que utiliza essas classes.

4.2 DL_Meso

O DL_MESO é um pacote de simulação voltado para a simulação na mesoescala, desenvolvido por Michael Seaton no Daresbury Laboratory. Atualmente, é distribuído sob licença gratuita para instituições acadêmicas e pesquisas sem fins lucartivos SEATON e SMITH (2016).

O DL_MESO possui dois métodos de simulação implementados: o método de Lattice Boltzmann, que não será abordado neste trabalho, e o método DPD. O

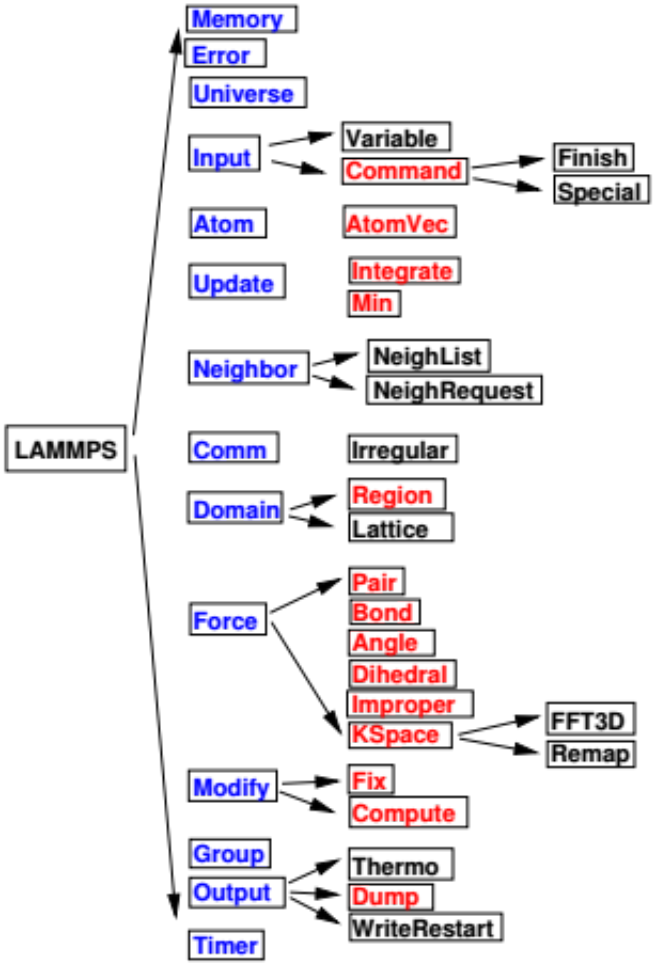


Figura 4.1: Hieraquia de classes do LAMMPS. As classes em azul são as classes visivéis pelas outras classes, as vermelhas são as classes bases do sistema. (Figura retirada de PLIMPTON (2011))

método DPD foi implementado na liguagem FORTRAN e com o paradigma de programação modular.

Entre as principais características do DL_MESO estão (SEATON e SMITH, 2016):

1. possui diferentes termostatos e barostatos;
2. diferentes potenciais;
3. paralelismo utilizando MPI;
4. diferentes condições de contorno, sendo uma delas a condição de contorno de Lees-Edwards (LEES e EDWARDS, 1972);
5. e uma interface gráfica (GUI) para a geração dos arquivos de entrada.

4.2.1 Estrutura

O DL_MESO foi programado usando-se o paradigma de programação modular. Segundo SCOTT (2000), esse paradigma sugere que o programa seja composto por diversas partes, chamadas módulo. Cada módulo agrupa um conjunto de dados que são alterados pelos procedimentos presentes neste módulo.

Nesse paradigma busca-se dividir o problema em subproblemas, cada qual associada a um módulo. Assim os módulos trabalham em conjuto pra resolver o problema.

A modularização do programa, permite que os módulos sejam usados em diferentes programas e facilita a sua manutenção (SCOTT, 2000). Uma lista com os módulos presentes no DL_MESO pode ser encontrada em SEATON *et al.* (2013).

4.2.2 Modificações

O módulo responsável pelo cálculo da força de interação entre os pares de partículas é o *field_module.f90*. Além dele, também é necessário alterar o módulo *config_module.f90* que é responsável pela leitura dos arquivos de entrada.

As alterações realizadas nesses módulos se encontram no Apêndice C. No Apêndice D se encontra um arquivo de entrada onde essas modificações são empregadas.

4.3 Algumas diferenças entre os programas

Uma das diferenças existente entre os programas é o fato de que o LAMMPS não possui implementado o integrador DPD-VV. O LAMMPS utiliza o integrador original de Velocity-Verlet, que é igual ao integrador de GW com $\lambda = 1/2$.

O DL_MESO possui implementado o integrador DPD-VV e as condições de contorno de Lees-Edwards, as quais o LAMMPS não possui implementadas.

Outra diferença entre eles são as unidades reduzidas utilizadas. O DL_MESO utiliza a massa da partícula como massa unitária, o tamanho da partícula como comprimento unitário e a energia definida como k_BT como energia unitária.

No caso do LAMMPS, as unidades reduzidas são: a massa da partícula como massa unitária, o raio de corte da partícula como comprimento unitário e a energia definida como k_BT como energia unitária.

4.4 Pós-processamento

Para elucidar alguns fenômenos que ocorrem no sistema, após realizar as simulações, se faz necessário processar os dados obtidos para uma melhor compreensão dos mesmos.

Os arquivos produzidos pelos programas contém o histórico das posições e velocidades das partículas. Esses arquivos podem ser gerados em formatos usados por alguns programas de visualização (*e.g.* VMD), ou podem ser gerados no formato padrão definido pelos programas. Em ambos os casos, é necessário realizar um pós processamento para que se possa realizar operações com as posições e velocidades das partículas.

Para extrair as posições e velocidades geradas pelo LAMMPS foi criado um script na linguagem Julia (Apêndice E). No caso do DL_MESO, cujo arquivos gerados são binários (não estão em formato de texto), foi gerado um código em FORTRAN (Apêndice F), baseado no código do programa *local* presente no pacote do DL_MESO. Esse código acessa os arquivos binários e salva os dados em formato de texto.

4.4.1 Extração do campo de velocidades

Para facilitar a visualização de como as velocidades das partículas estão correlacionadas, defini-se uma velocidade *coarse-grained* (HINZ *et al.*, 2014):

$$\mathbf{u}_M(\mathbf{x}_M) = \frac{1}{n}\sum_{i=1}^{N} \mathbf{v}_i \psi(\mathbf{x}_M - \mathbf{x}_i) \tag{4.1}$$

que é computada em uma malha uniforme com espaçamento equidistante nas duas direções ($\mathbf{x}_M$). ψ é um filtro Gaussiano com largura adimensional igual a 15.0 e n é o número de vezes em que $\psi \neq 0$.

Foi desenvolvido um script na linguagem Julia para calcular $\mathbf{u}_M$. Ele pode ser encontrado no Apêndice G.

4.4.2 Extração do perfil de velocidades ou de temperatura

Uma alternativa para avaliar como ocorre o escoamento de um fluido na caixa de simulação é observar o seu perfil de velocidade.

O perfil de velocidade é obtido criando-se divisões na caixa de simulação ao longo de uma das direções (esquematizado na Figura 4.2). Em cada divisão é realizada a média da propriedade escolhida. Também é realizada a média temporal a fim de reduzir o efeito de flutuações. Logo, cada divisão possui um valor médio da propriedade escolhida.

Para realizar essa operação no LAMMPS utiliza-se os comandos *compute chunk/atom* e *fix ave/chunk*. O primeiro sendo responsável por dividir a caixa de simulação, e o segundo realiza a média (inclusive temporal) em cada uma das divisões.

No DL_MESO esse perfil pode ser obtido com o programa *local*, que se encontra

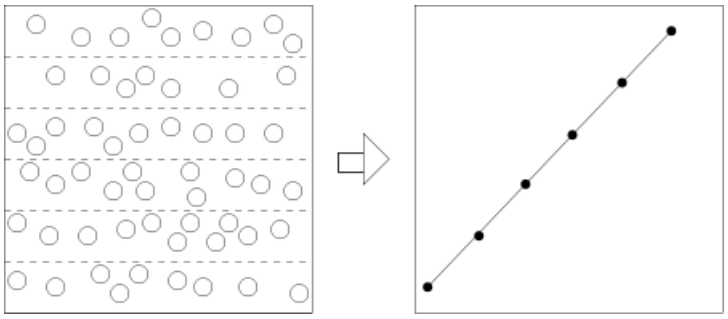


Figura 4.2: Representação de como se obtém um perfil de velocidade da caixa de simulação. A caixa é dividida em regiões, onde toma-se a média da velocidade para cada região.

nas ferramentas do pacote DL_MESO. O programa *local* utiliza os arquivos gerados pela simulação, para gerar as divisões na caixa e calcular a média da propriedade na mesma. Esses dados são salvos no arquivo *average.vtk*.

O perfil de temperatura é obtido de forma análoga. Porém, durante o cálculo da temperatura média para cada divisão da caixa, a *streaming velocity* do sistema (EVANS e MORRISS (1986)) é desconsiderada. Para isso é necessário modificar o programa *local* do DL_MESO.

Capítulo 5

Resultados e discussões

Este capítulo destina-se a aprestar os resultados obtidos utilizando os programas LAMMPS e DL_MESO.

5.1 Avaliação do integrador numérico

A fim de avaliar os intregadores presente no LAMMPS e DL_MESO realizamos a simulação de um fluido DPD simples, baseado em simulações feitas por GROOT e WARREN (1997).

O sistema é composto por 4.000 partículas distrbuidas em uma caixa de simulação com dimensões 10x10x10. É o utilizado o modelo clássico do DPD com os seguintes parâmetros: $r_c = 1$, $m_i = 1$, $a_{ij} = 25$,$\gamma_i j = 4,5$ e temperatura de equilíbrio $k_B T = 1$. O tamanho do passo de tempo (Δt) e o número total de passos de tempo (N_t) variam simultaneamente para que o tempo final nas simulações seja o mesmo. Utilizamos as condições periódicas de contorno.

A temperatura foi obtida através da equação (3.17), realizando-se a média nos últimos $N_t/2$ passos de tempo. Com isso a diferença entre a temperatura de equilíbrio e a temperatura média no sistema pode ser avaliada, indicando o excesso de temperatrura no sistema.

Os resultados obtidos são apresentados na Figura 5.1. O maior desvio de temperatura ocorre no método implementado no LAMMPS. Este fato era esperado, uma vez que o método DPD-VV requer uma atualização adicional das forças dissipativas.

5.2 Fases dinâmicas

Para verificar se as modificações feitas nos programas estão corretas, simulamos um sistema bidimensional como indicado em HINZ *et al.* (2014). É importante destacar que HINZ *et al.* (2014) utilizaram somente o LAMMPS. Eles obtiveram

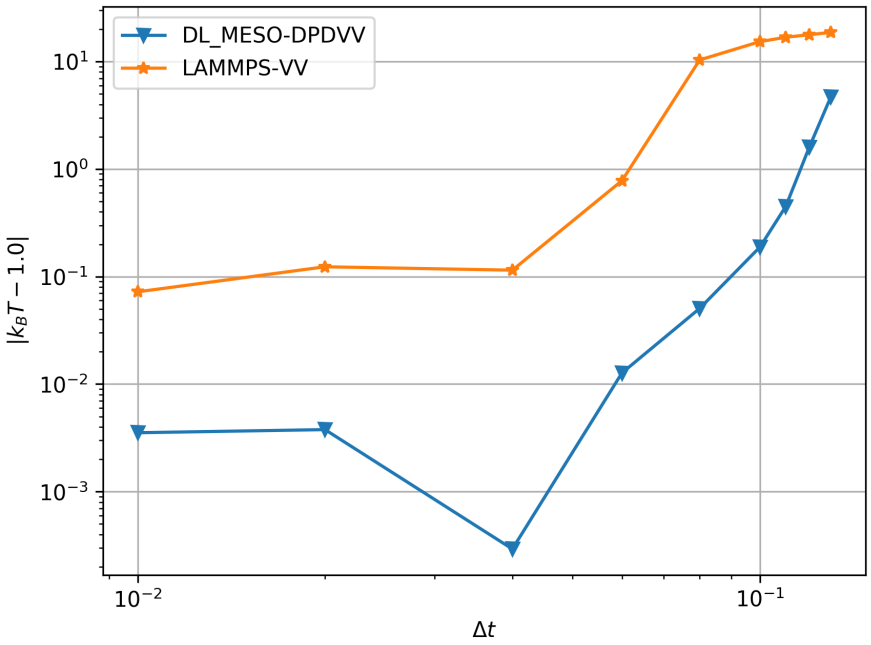


Figura 5.1: Excesso de temperatrura nas simulações variando-se o passo de tempo

diferentes fases dinâmicas para o sistema com partículas ativas. As fases podem ser determinadas alterando-se o número de Péclet (Pe-equação (3.19)) e a concentração de partículas ativas no sistema (ϕ).

São relizadas simulações em duas dimensões com os seguintes parâmetros (HINZ *et al.*, 2014): $r_c = 1$, $m_i = 1$, $a_{ij} = 25$, $\gamma_i j = 4,5$ e a temperatura de equilíbrio $k_BT = 1$. Considerando a inclusão da força de propulsão, o parâmetro β é fixado em 2,25 e α varia, para que haja variação no número de Péclet.

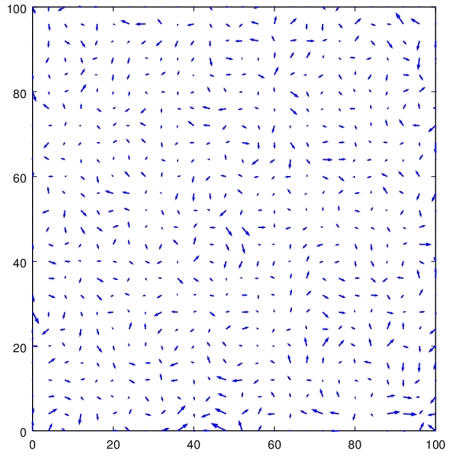
A caixa de simulação possui dimensões 100x100 e o número total de partículas é igual a 25.000. O passo de tempo escolhido é $\Delta t = 0,003$ e são realizadas 2x10^6 iterações. Usamos as condições periódicas de contornos.

Após a simulação, ralizamos um pós processamento como descrito na seção 4.4.1, para a obtenção do campo de velocicade $\mathbf{u}_M$. Utilizando-se o LAMMPS obteve-se os resultados mostrado na Figura 5.2

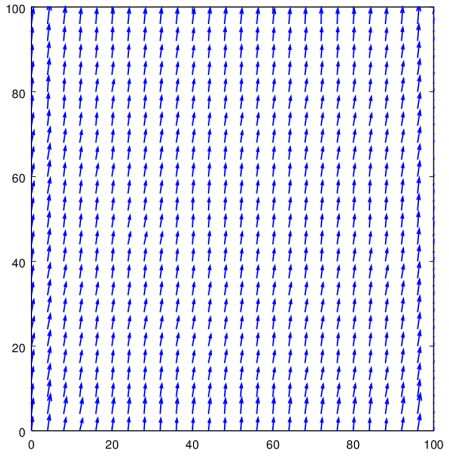
Os gráficos obtidos para as fases dinâmcas são qualitativamente análogos aos encontrados por HINZ *et al.* (2014).

A mesma simulação, foi realizada utilizando-se o DL_MESO. Os resultados são apresentados na Figura 5.3

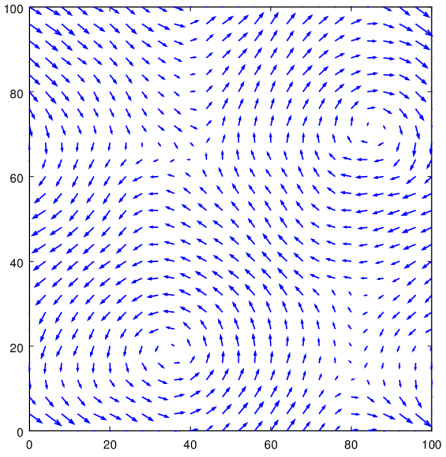
Nota-se que para o caso em que $\phi = 0,1$ e $Pe = 1,11$ não se obteve a fase vorticial como se esperava. Baseado nos resultados apresentados por HINZ *et al.* (2014), espera-se que essa fase apareça para os casos que $\phi = 0,1$ e Pe esteja no intervalo $1,1 - 1,9$ e $2,1 - 2,2$ (considerando os outros parâmetros de simulação



(a) Fase turbulenta ($\phi = 0,1$ e $Pe = 0,11$)

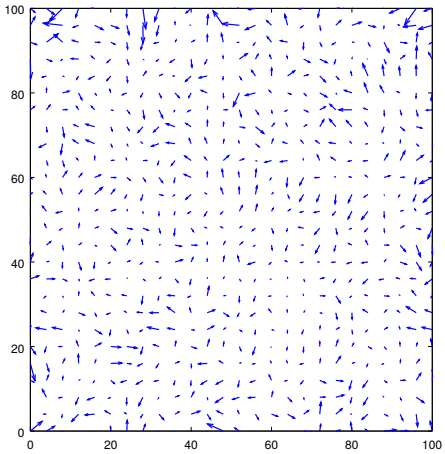


(b) Fase bionemática($\phi = 0,5$ e $Pe = 1,11$)

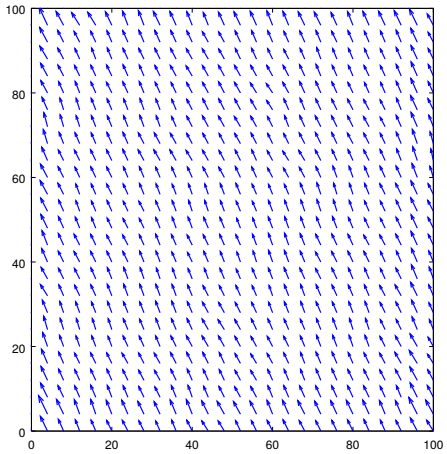


(c) Fase vorticial($\phi = 0,1$ e $Pe = 1,11$)

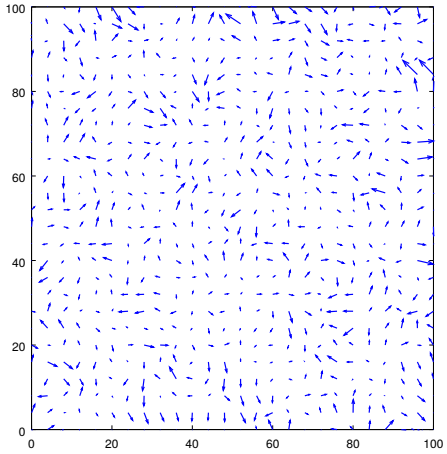
Figura 5.2: $\mathbf{u}_M(\mathbf{x}_M)$ obtido para $\beta = 2,25$, variando-se ϕ e Pe, e utilizando o LAMMPS.



(a) Fase turbulenta ($\phi = 0,1$ e $Pe = 0,11$)



(b) Fase bionemática($\phi = 0,5$ e $Pe = 1,11$)



(c) Fase turbulenta($\phi = 0,1$ e $Pe = 1,11$)

Figura 5.3: $\mathbf{u}_M(\mathbf{x}_M)$ obtido para $\beta = 2,25$, variando-se ϕ e Pe, e utilizando o DL_MESO

.

fixos). E também para os casos em que $\phi = 0,2$ e $Pe = 1,3$ ou $Pe = 1,5$.

Foram realizados testes utilizando-se esses valores para ϕ e Pe com o DL_MESO e não obteve-se a fase vorticial em nenhum caso. Todos casos em que $\phi = 0.1$ a fase dinâmica obtida é a fase turbulenta e para $\phi = 0,2$ chega-se na fase bionemática.

Para estudar as diferenças entre os resultados obtidos utilizando os dois programas, trabalhamos com um sistema menor, a fim de reduzir o custo computacional e consequentemente o tempo de simulação.

Nessa nova simulação, os parâmetros se mantiveram os mesmos e reduzimos o tamanho da caixa de simulação para 50x50. Visando manter a mesma densidade de partículas, o número total delas foi alterado para $N = 6.250$. Os resultados dessa simulação são apresentados na Figura 5.4.

Esses resultados mostram que, aparentemente, a escolha do integrador númerico interfere nas repostas obtidas pelo modelo. HINZ *et al.* (2014) sugeriram que o surgimento da fase vorticial para baixas concentrações, indicaria a relevância das partículas não-ativas para se 'estabilizar' o sistema nessa fase. É possível que houvessem outras propriedades hidrodinâmicas não avaliadas. Ao que tudo indica, a atualização das forças dissipativas no integrador DPD-VV suprimi essas propriedades.

5.3 Obtenção da viscosidade[1]

As condições de contorno de Lees-Edwards permite impor um cisalhamento constante ao sistema, com a intenção de que o sistema alcance um estado permanente que represente um escoamento de Couette(representado na Figura 5.5).

Após o sistema alcançar esse estado, a viscosidade pode ser calculada com a seguinte expressão (utilizado o referencial da Figura 5.5):

$$\eta = \langle S_{xy} \rangle_t \frac{L_y}{2V_s} \tag{5.1}$$

onde S_{xy} é a componente do tensor $\mathbf{S}$ associada ao cisalhamento, η a viscosidade, L_y é a altura da caixa de simulação e V_s a velocidade com que as caixas do contorno superior e inferior se movimentam nas condições de Lees-Edwards. $\langle\rangle_t$ representa a média temporal.

Segundo MOSHFEGH e JABBARZADEH (2015), a ideia de Lees-Edwards foi desenvolvida para a utilização com dinâmica molecular. Como essas condições adicionam energia ao sistema, em dinâmica molecular pode ser necessário utilizar algum artifício para manter a temperatura do sistema constante . O DPD já possui um termostato formado por $\mathbf{F}^D$ e $\mathbf{F}^R$, o qual consegue minimizar a variação de tem-

[1]Para obtenção da viscosidade somente o DL_MESO foi utilizado.

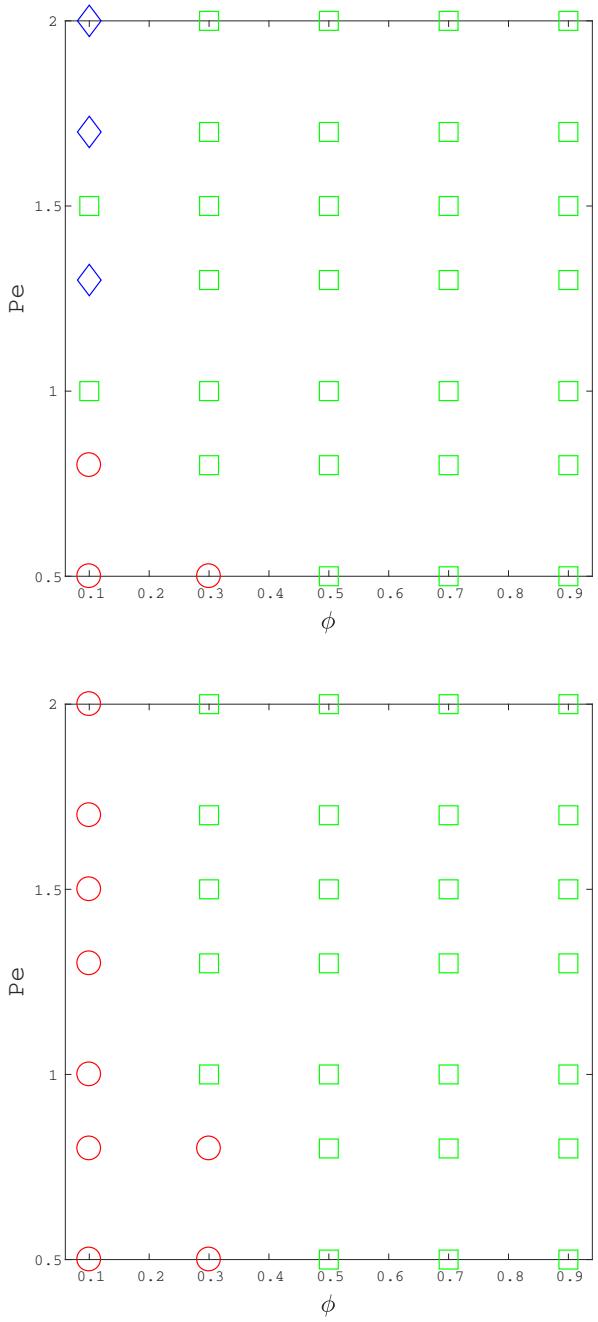


Figura 5.4: Diagrama de fases para o LAMMPS (acima) e DL_MESO-DPD-VV(abaixo). O losângulo representa a fase vorticial, o círculo a fase turbulenta e o quadrado a fase bionemática

peratura do sistema. Em MOSHFEGH *et al.* (2015) também é mostrado que o termostato $\mathbf{F}^D - \mathbf{F}^R$ é mais eficiente em manter a temperatura prescrita para taxas de cisalhamentos menores.

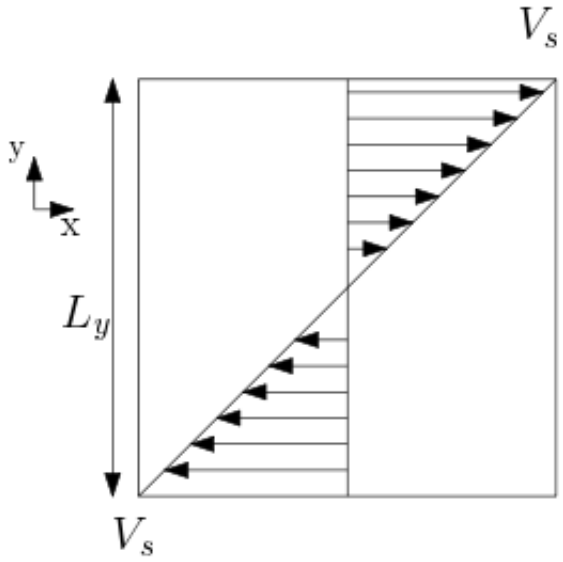


Figura 5.5: Representação do perfil de velocidade em um escoamento de Couette

5.3.1 Caso 1: Fluido simples DPD

Para melhor entender os efeitos das condições de Lees-Edwards no DPD, foi realizada a simulação de um fluido sem partículas ativas.

Baseado em MOSHFEGH e JABBARZADEH (2015), foram relizadas simulações com os seguintes parâmetros: $r_c = 1$, $m_i = 1$, $a_{ij} = 25$, $\gamma_i j = 4,5$ e a temperatura de equilíbrio $k_B T = 1$. A caixa de simulação possui dimensões 10x10x10 e o número total de partículas é igual a 3.000. O passo de tempo escolhido é $\Delta t = 0,01$ e $N_t = 50.000$.

As condições de Lees-Edwards foram utilizadas e as velocidades V_s escolhidas como 0,2; 0,5; 0,8; 1 e 1,2. Os perfis de velocidade e temperatura do sistemas são obtidos como descrito na seção 4.4.2. A Figura 5.6 mostra os perfis de velocidade na direção x e os perfis de temperatura do sistema.

Observa-se que perto das bordas da caixa de simulação, ambos os perfis sofrem um desvio dos valores esperados. Esse fato ocorre devido a força $\mathbf{F}^D$ do DPD. Como ela é proporcional a velocidades relativas, as alterações de velocidades que ocorrem quando uma partícula reentra no sistema são contabilizados no cálculo de $\mathbf{F}^D$ de outras partículas, fazendo com que o termo dissipativo retire energia do sistema, reduzindo a velocidade relativa. Logo, os efeitos da alteração de velocidade de uma partícula que reentra é gradativamente anulado pelas partículas da caixa simulação. E com isso, esses efeitos são capturados de forma mais eficiente nas regiões próximas das bordas na caixa de simulação.

Segundo CHATTERJEE (2007); MOSHFEGH e JABBARZADEH (2015), para os valores apresentados de V_s, os devios são pequenos e podem ser desconsiderados. Porém, para valores elevados de V_s, esses efeitos são amplificados e deve ser

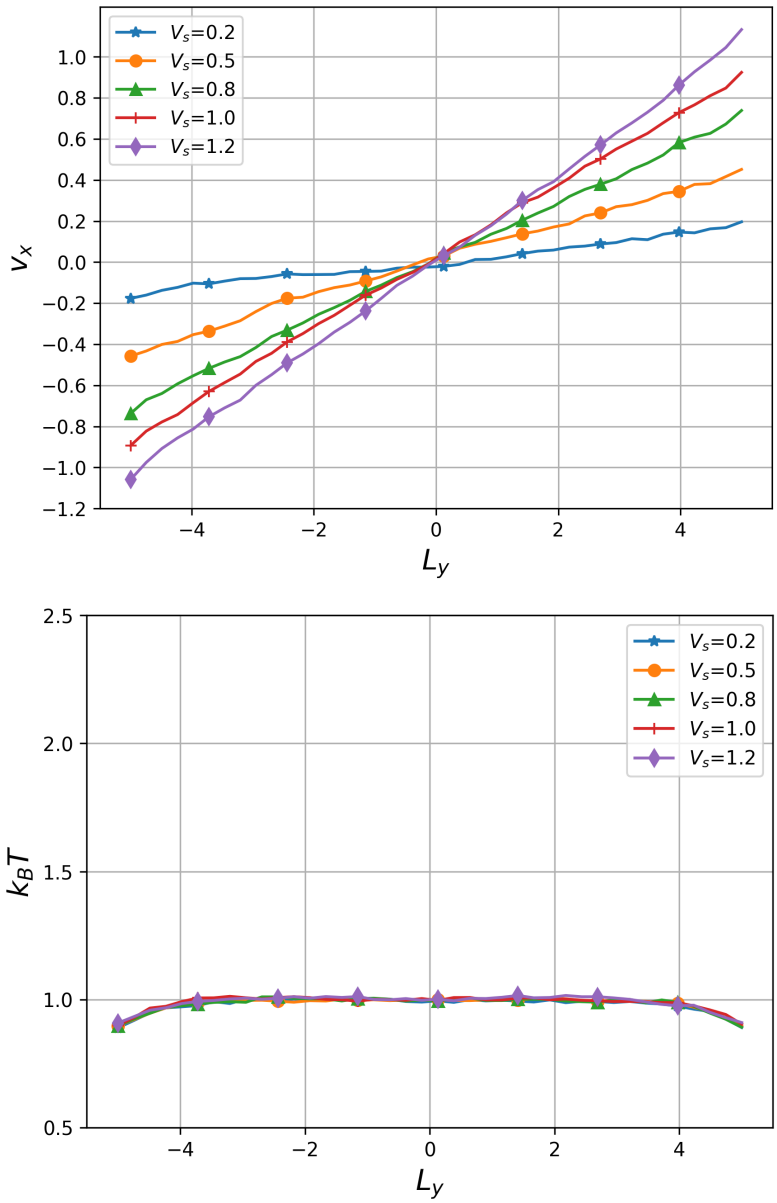


Figura 5.6: Perfil de velocidade e temperatrura obtidos para V_S entre $0.2 - 1.2$

usado algum artifício para corrigir os desvios nos perfis. Esses artifícios não serão abordados neste trabalho.

A fim de demonstrar os efeitos quando se aumenta V_s, foram realizadas simulações mantendo-se os parâmetros anteriores e alterando-se a velocidade V_s entre 2,5; 5; 7,5 e 10. A Figura 5.7 mostra os resultado obtidos.

Considerando as simulações para valores baixos de V_s, obteve-se os valores presentes na Tabela 5.1 de viscosidade. Nela também se encontra o valor obtido por MOSHFEGH e JABBARZADEH (2015).

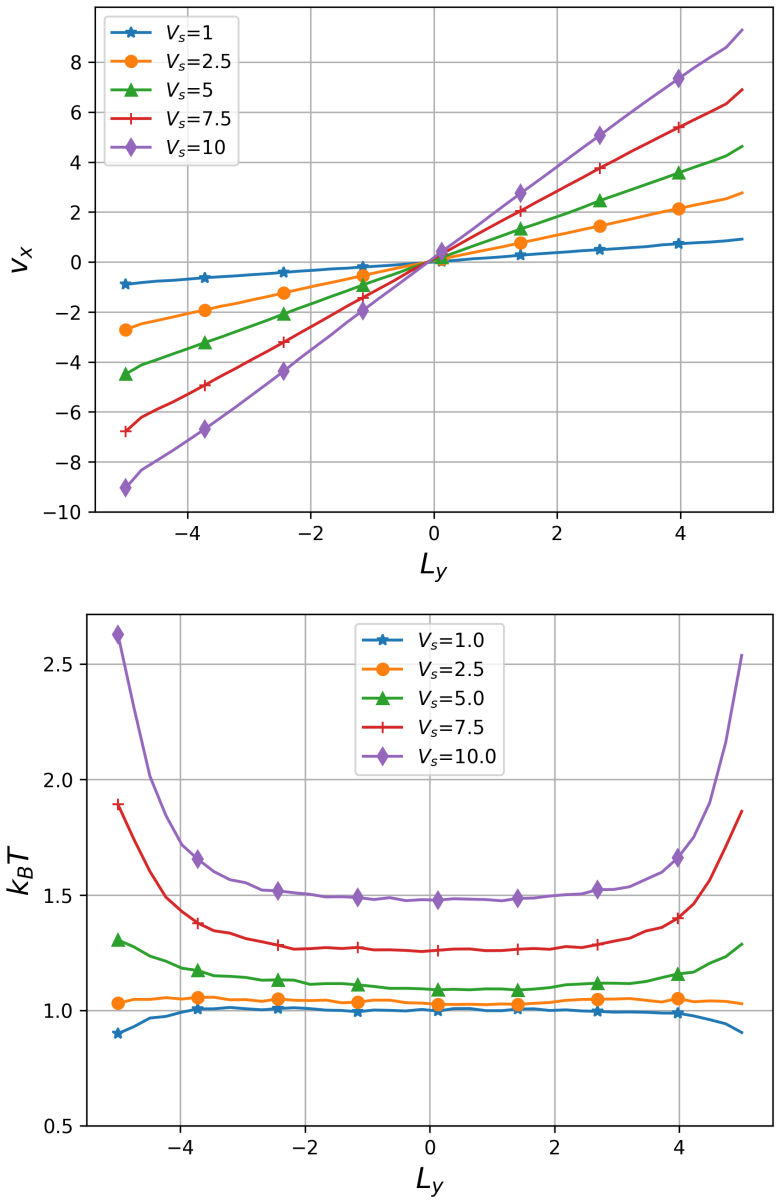


Figura 5.7: Perfil de velocidade e temperatrura obtidos para V_S entre $1.0 - 10.0$

5.3.2 Caso 2: Suspensão de partículas ativas

Para avaliar o efeito da presença de partículas na viscosidade da suspensão, foram realizados experimentos variando-se a concentração das partículas ativas, a velocidade V_s (considerando-se baixas taxas de cisalhamento) e o número de Péclet do sistema.

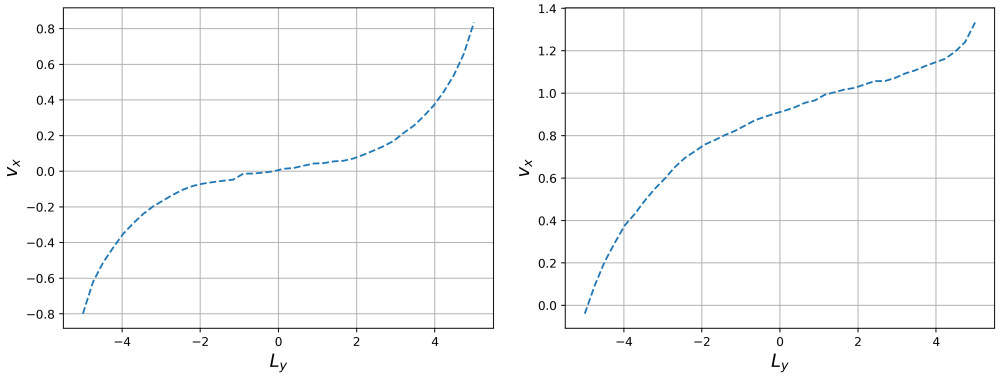
Foram relizadas simulações com os seguintes parâmetros para ambos tipos de partícula: $r_c = 1$, $m_i = 1$, $a_{ij} = 25$, $\gamma_i j = 4,5$ e a temperatura de equilíbrio $k_BT = 1$.A caixa de simulação possui dimensões 10x10x10 e o número total de partículas é igual a 3000. O passo de tempo escolhido é $\Delta t = 0,01$ e $N_t = 50.000$. São utilizadas as condições de contorno de Lees-Edwards.

	MOSHFEGH JABBARZADEH (2015)	$V_s = 0,2$	$V_s = 0,5$	$V_s = 0,8$	$V_s = 1$	$V_s = 1,2$
η	0,86	0,855	0,839	0,848	0,851	0,707

Tabela 5.1: Viscosidades obtidas para diferentes V_s

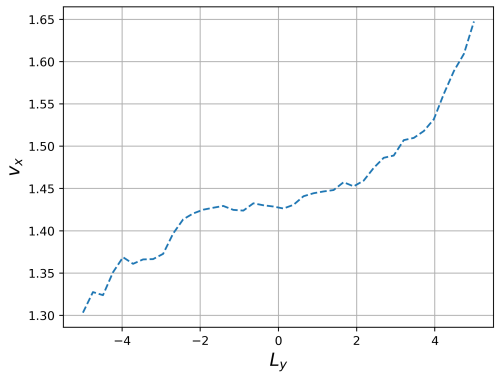
A príncipio o número de Péclet avaliado foi 0,5; 1,5 e 2,0; a concentração utilizadas foi 0,1; 0,3; 0,5 e a velocidade V_s varia entre 0,2; 0,5; 0,8; 1,0 e 1,2.

Na maioria dos casos o sistema não alcança o estado de escoamento de Couette. Isso é exemplificado na Figura 5.8, que mostra alguns perfis de velocidade obtidos.

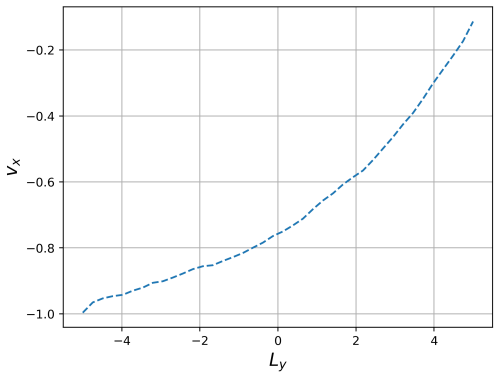


(a) Obtido para $\phi = 0,3$, $Pe = 0,5$ e $V_s = 1$

(b) Obtido para $\phi = 0,3$, $Pe = 1,5$ e $V_s = 0,8$



(c) Obtido para $\phi = 0,1$, $Pe = 2,0$ e $V_s = 0,2$



(d) Obtido para $\phi = 0,5$, $Pe = 1,5$ e $V_s = 0,5$

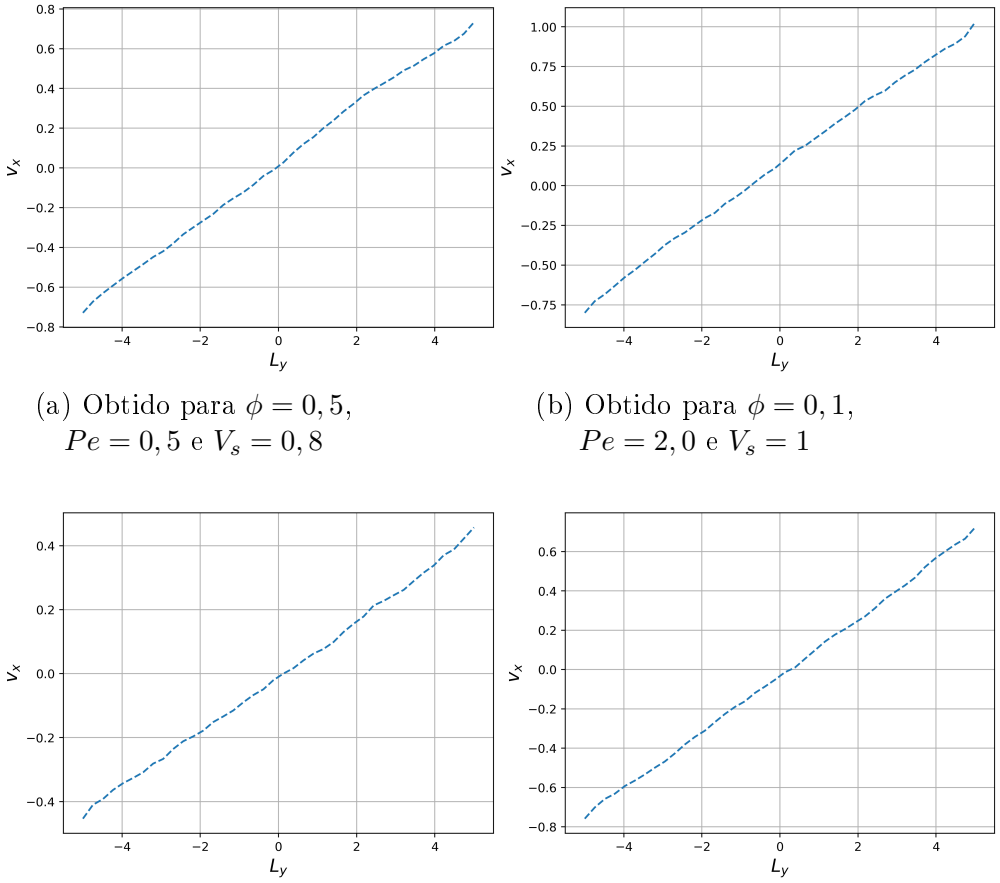
Figura 5.8: Alguns dos perfis de velocidades que não representam o escoamento de Couette

.

Novas simulações foram realizadas variando-se V_s, ϕ, Pe, as dimensões da caixa de simulação, a densidade númerica das partículas, o tamanho do passo de tempo e o número total de passos de tempo. A lista com os testes realizados se encontra no

Apêndice H. As simulações indicam que as condições de Lees-Edwards são incapazes, na maioria dos casos, de impor o cisalhamento ao sistema com o DPD modificado.

Para os casos em que se obteve um perfil linear de velocidade (exemplificados na Figura 5.9), calculamos a viscosidade relativa com respeito à viscosidade do fluido sem partículas ativas. Os valores obtidos estão indicados na Tabela H.4. Esses valores indicam um comportamento da viscosidade não condizente com aquele encontrado em experimentos (LÓPEZ *et al.*, 2015).



(a) Obtido para $\phi = 0,5$, $Pe = 0,5$ e $V_s = 0,8$

(b) Obtido para $\phi = 0,1$, $Pe = 2,0$ e $V_s = 1$

(c) Obtido para $\phi = 0,3$, $Pe = 2,0$ e $V_s = 0,5$

(d) Obtido para $\phi = 0,3$, $Pe = 2,0$ e $V_s = 0,8$

Figura 5.9: Alguns dos perfis de velocidades que representam o escoamento de Couette

.

Casos	η_R
$\phi = 0,2$, $Pe = 2,0$ e $V_s = 0,5$	1,469
$\phi = 0,3$, $Pe = 1,5$ e $V_s = 0,5$	2,291
$\phi = 0,5$, $Pe = 2,0$ e $V_s = 0,5$	3,526
$\phi = 0,1$, $Pe = 1,5$ e $V_s = 0,8$	1,1504
$\phi = 0,1$, $Pe = 2,0$ e $V_s = 0,8$	1,149
$\phi = 0,3$, $Pe = 2,0$ e $V_s = 0,8$	1,429
$\phi = 0,5$, $Pe = 0,5$ e $V_s = 0,8$	1,486
$\phi = 0,1$, $Pe = 0,5$ e $V_s = 1$	0,899
$\phi = 0,1$, $Pe = 2,0$ e $V_s = 1$	1,166
$\phi = 0,3$, $Pe = 2,0$ e $V_s = 1$	1,246
$\phi = 0,5$, $Pe = 0,5$ e $V_s = 1$	1,353
$\phi = 0,1$, $Pe = 1,25$ e $V_s = 0,8$	0,99
$\phi = 0,1$, $Pe = 1,25$ e $V_s = 1$	1,01
$\phi = 0,5$, $Pe = 1,11$ e $V_s = 0,5$	4,03
$\phi = 0,5$, $Pe = 1,11$ e $V_s = 0,8$	1,973
$\phi = 0,1$, $Pe = 1,11$ e $V_s = 1$	0,913
$\phi = 0,1$, $Pe = 1,11$ e $V_s = 1$	1,05
$\phi = 0,3$, $Pe = 1,11$ e $V_s = 1$	1,66

Tabela 5.2: Viscosidades relativas (η_R) com respeito viscosidade do fluido sem partícula ativas. Valores obtidos para alguns casos de testes.

Capítulo 6

Conclusão

6.1 Conclusões

Neste trabalho, foi implementado o modelo DPD proposto na literatura para se considerar um sistema com partículas ativas. Permitindo a investigação de fenômenos decorrente desse modelo. As modificações foram realizadas nos programas LAMMPS e DL_MESO. Outros códigos foram desenvolvidos para realizar o pós-processamento e obter os perfis de velocidade, temperatura e um campo de velocidade *coarse-grained*.

Utilizandos-se do modelo clássico, observou-se que o controle de temperatura do método DPD-VV presente no DL_MESO é melhor do que método de Verlet presente no LAMMPS. Sendo esse resultado condizente com a literatura.

Para validar as modificaçõees implementadas nos programas, as diferentes fases dinâmicas do sistema foram obtidas. Os resultados obtidos pelo LAMMPS são qualitativamente análogos com os presentes na literatura. Todavia, os resultados obtidos pelo DL_MESO não foram capazes de obter todas as fases dinâmicas.

A fim de averigar esse fato, construiu-se o diagrama de fases para ambos os programas e observou-se que as fases obtidas são divergentes na maioria dos casos para baixa concentração de partículas ativas. É necessário um estudo mais aprofundado para enteder essa diferença, mas em uma análise preliminar considerou-se que o método DPD-VV suprimi essas fases.

Para se obter a viscosidade do fluido as condições de Lees-Edwards são utilizadas para aplicar cisalhamento na caixa de simulação. Em um primeiro momento, as simulações que sofreram um efeito de cisalhamento menor, são capazes de atingir o regime de escoamento de Couete. Porém, quando se aplica cisalhamentos maiores, as forças dissipativas presentes no modelo do DPD, impede que o efeito seja corretamente propagado das bordas para o centro da caixa de simulação. Os valores obtidos para a viscosidade neste caso é próximo do valor encontrado na literatura.

Utilizando-se do modelo implementado no DL_MESO, foram realizadas simulações em que se aplica o cisalhamento em uma suspensão com partículas ativas. A maioria das simulações realizadas foi incapaz de obter um perfil de velocidade coerente com o cisalhamento aplicado. Nos casos em que o escoamento desejado foram alcançados, as viscosidades relativas ao fluido sem a presença de partículas ativas, não desmostram um comportamento compatível com os da literatura.

O que mostra a ineficiência da combinação do modelo com o método para capturar os efeitos reológicos de suspensões com bactérias.

6.2 Perspectivas Futuras

A partir deste trabalho, alguns projetos no futuros podem ser desenvolvidos, entre eles:

1. implementação de caracteríticas físicas e mecânicas das bactérias, como forma e modo de propulsão;
2. implementadas de outras funções capazes de adicionar 'atividade' nas partículas;
3. parametrização do modelo para um tipo especifico de suspensão de bactéria;
4. e utilização de modelagem multiescala para associar os efeitos observados na mesoescala com a macroescala.

Referências Bibliográficas

BROWN, R., 1828, "On the particles contained in the pollen of plants; and on the general existence of active molecules in organic and inorganic bodies", *Edinburgh New Philosophical Journal*, v. 5, pp. 358 371.

CHATTERJEE, A., 2007, "Modification to Lees Edwards periodic boundary condition for dissipative particle dynamics simulation with high dissipation rates", *Molecular Simulation*, v. 33, n. 15, pp. 1233 1236.

CZIRÓK, A., BEN-JACOB, E., COHEN, I., et al., 1996, "Formation of complex bacterial colonies via self-generated vortices", *Physical Review E*, v. 54, n. 2, pp. 1791.

D'ORSOGNA, M. R., CHUANG, Y.-L., BERTOZZI, A. L., et al., 2006, "Self-propelled particles with soft-core interactions: patterns, stability, and collapse", *Physical review letters*, v. 96, n. 10, pp. 104302.

DUNKEL, J., EBELING, W., TRIGGER, S. A., 2004, "Active and passive Brownian motion of charged particles in two-dimensional plasma models", *Physical Review E*, v. 70, n. 4, pp. 046406.

EBELING, W., 2004, "Nonlinear Brownian motion mean square displacement", *Condensed Matter Physics*.

EBELING, W., DUNKEL, J., ERDMANN, U., et al., 2005, "Klimontovich's contributions to the kinetic theory of nonlinear Brownian motion and new developments". In: *Journal of Physics: Conference Series*, v. 11, p. 89. IOP Publishing.

EINSTEIN, A., 1905, "Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen", *Annalen der physik*, v. 322, n. 8, pp. 549 560.

ESPAÑOL, P., WARREN, P., 1995, "Statistical Mechanics of Dissipative Particle Dynamics", *Europhysics Letters*.

EVANS, D. J., MORRISS, G. P., 1986, "Shear thickening and turbulence in simple fluids", *Physical review letters*, v. 56, n. 20, pp. 2172.

FLETCHER, D. A., GEISSLER, P. L., 2009, "Active biological materials", *Annual review of physical chemistry*, v. 60, pp. 469 486.

GHOSH, P. K., MISKO, V. R., MARCHESONI, F., et al., 2013, "Self-propelled Janus particles in a ratchet: Numerical simulations", *Physical review letters*, v. 110, n. 26, pp. 268301.

GIBSON, J., CHEN, K., CHYNOWETH, S., 1999, "The equilibrium of a velocity-Verlet type algorithm for DPD with finite time steps", *International Journal of Modern Physics C*, v. 10, n. 01, pp. 241 261.

GROOT, R. D., WARREN, P. B., 1997, "Dissipative particle dynamics: Bridging the gap between atomistic and mesoscopic simulation", *Journal of Chemical Physics*.

HINZ, D. F., PANCHENKO, A., KIM, T., et al., 2014, "Motility versus fluctuations: Mixtures of self-propelled and passive particles", *Soft Matter*.

HIRATSUKA, Y., M.MIYATA, TADA, T., et al., 2006, "A microrotary motor powered by bacteria", *Proceedings of National Academy of Sciences of the United States of America*.

HOOGERBRUGGE, P. J., KOELMAN, J. M. V. A., 1992, "Simulating Microscopic Hydrodynamic Phenomena with Dissipative Particle Dynamics", *Europhysics Letters*.

LANGEVIN, P., 1908, "Sur la théorie du mouvement brownien", *CR Acad. Sci. Paris*, v. 146, n. 530-533, pp. 530.

LEES, A. W., EDWARDS, S. F., 1972, "The computer study of transport processes under extreme conditions", *Journal of Physics C: Solid State Physics*.

LEONARDO, R. D., ANGELANI, L., DELL'ARCIPRETEB, D., et al., 2010, "Bacterial ratchet motors", *Proceedings of National Academy of Sciences of the United States of America*.

LEVINE, H., RAPPEL, W.-J., COHEN, I., 2000, "Self-organization in systems of self-propelled particles", *Physical Review E*, v. 63, n. 1, pp. 017101.

LIU, M. B., LIU, G. R., 2016, *Particle Methods for Multi-Scale and Multi-Physics*. Singapore, World Scientific.

LÓPEZ, H. M., GACHELIN, J., DOUARCHE, C., et al., 2015, "Turning Bacteria Suspensions into Superfluids", *PHYSICAL REVIEW LETTERS*.

MARCHETTI, M. C., JOANNY, J.-F., RAMASWAMY, S., et al., 2013, "Hydrodynamics of soft active matter", *Reviews of Modern Physics*, v. 85, n. 3, pp. 1143.

MAZO, R. M., 2002, *Brownian motion: fluctuations, dynamics, and applications*, v. 112. New York, Oxford University Press on Demand.

MOEENDARBARY, E., NG, T. Y., ZANGENEH, M., 2009, "Dissipative Particle Dynamics: introduction, methodology and complex fluid applications- a review", *International Journal of Applied Mechanics*.

MOSHFEGH, A., JABBARZADEH, A., 2015, "Dissipative particle dynamics: effects of parameterization and thermostating schemes on rheology", *Soft Materials*, v. 13, n. 2, pp. 106 117.

MOSHFEGH, A., AHMADI, G., JABBARZADEH, A., 2015, "Thermostatic and rheological responses of DPD fluid to extreme shear under modified Lees-Edwards boundary condition", *The European Physical Journal E*, v. 38, n. 12, pp. 134.

PLIMPTON, S., 2011. "LAMMPS Developer Guide". .

PLIMPTON, S., OTHERS, 2005. "LAMMPS user's manual". .

SADUS, R. J., 1999, *Molecular Simulations of Fluids*. Amsterdam, Elsevier.

SCOTT, M. L., 2000, *Programming language pragmatics*. San Francisco, Morgan Kaufmann.

SEATON, M., SMITH, W., 2016. "DL MESO USER MANUAL". .

SEATON, M. A., ANDERSON, R. L., METZ, S., et al., 2013, "DL_MESO: highly scalable mesoscale simulations", *Molecular Simulation*, v. 39, n. 10, pp. 796 821.

SIMHA, R. A., RAMASWAMY, S., 2002, "Hydrodynamic fluctuations and instabilities in ordered suspensions of self-propelled particles", *Physical review letters*, v. 89, n. 5, pp. 058101.

TONER, J., TU, Y., 1995, "Long-range order in a two-dimensional dynamical XY model: how birds fly together", *Physical review letters*, v. 75, n. 23, pp. 4326.

VATTULAINEN, I., KARTTUNEN, M., BESOLD, G., et al., 2002, "Integration schemes for dissipative particle dynamics simulations: From softly interacting systems towards hybrid models", *The Journal of chemical physics*, v. 116, n. 10, pp. 3967 3979.

VICSEK, T., ZAFEIRIS, A., 2012, "Collective motion", *Physics Reports*, v. 517, n. 3-4, pp. 71 140.

VICSEK, T., CZIRÓK, A., BEN-JACOB, E., et al., 1995, "Novel type of phase transition in a system of self-driven particles", *Physical review letters*, v. 75, n. 6, pp. 1226.

VOLFSON, D., COOKSON, S., HASTY, J., et al., 2008, "Biomechanical ordering of dense cell populations", *Proceedings of the National Academy of Sciences*, v. 105, n. 40, pp. 15346 15351.

WENSINK, H. H., DUNKEL, J., HEIDENREICH, S., et al., 2012, "Meso-scale turbulence in living fluids", *Proceedings of the National Academy of Sciences*, v. 109, n. 36, pp. 14308 14313.

ZWANZIG, R., 2001, *Nonequilibrium statistical mechanics*. New York, Oxford University Press.

Apêndice A

Classes *pair_dpd_flock.h* e *pair_dpd_WR.h*

Os arquivos .h e .cpp de ambas as classes. Para utiliza-lás, basta colocar o arquivo na pasta *src* e recompilar o LAMMPS.

Arquivo *pair_dpd_flock.h*:

```
/* -*- c++ -*- ----------------------------------------------------------
  Pairwise interaction from Hinz et al. 2014. Active-Active interaction (Header File)
---------------------------------------------------------------------- */

#ifdef PAIR_CLASS

PairStyle(dpd/flock,PairDPDFlock)

#else

#ifndef LMP_PAIR_DPD_FLOCK_H
#define LMP_PAIR_DPD_FLOCK_H

#include "pair.h"

namespace LAMMPS_NS {

class PairDPDFlock : public Pair {
 public:
  PairDPDFlock(class LAMMPS *);
  virtual ~PairDPDFlock();
  virtual void compute(int, int);
  virtual void settings(int, char **);
  virtual void coeff(int, char **);
  void init_style();
  double init_one(int, int);
  virtual void write_restart(FILE *);
  virtual void read_restart(FILE *);
  virtual void write_restart_settings(FILE *);
  virtual void read_restart_settings(FILE *);
  virtual void write_data(FILE *);
  virtual void write_data_all(FILE *);
  double single(int, int, int, int, double, double, double, double &);

 protected:
  double cut_global,temperature;
  double **cut;
  double **a0,**gamma;
  double *alpha,*beta; //Flock terms

  void allocate();
};


}

#endif
#endif
```

Arquivo *pair_dpd_flock.cpp*:

```
/* ----------------------------------------------------------------------

   Pairwise interaction from Hinz et al. 2014. Active-Active interaction

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


#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "pair_dpd_flock.h"
#include "atom.h"
#include "atom_vec.h"
#include "comm.h"
#include "update.h"
#include "force.h"
#include "neighbor.h"
#include "neigh_list.h"
#include "memory.h"
#include "error.h"

using namespace LAMMPS_NS;

#define EPSILON 1.0e-10

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

PairDPDFlock::PairDPDFlock(LAMMPS *lmp) : Pair(lmp)
{
  writedata = 1;
}

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

PairDPDFlock::~PairDPDFlock()
{
  if (allocated) {
    memory->destroy(setflag);
    memory->destroy(cutsq);
    memory->destroy(cut);
    memory->destroy(a0);
    memory->destroy(gamma);
    memory->destroy(alpha);
    memory->destroy(beta);
  }
}

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

void PairDPDFlock::compute(int eflag, int vflag)
{
  int i,j,ii,jj,inum,jnum,itype,jtype;
  double xtmp,ytmp,ztmp,delx,dely,delz,evdwl,fpair;
  double vxtmp,vytmp,vztmp,delvx,delvy,delvz;
  double rsq,r,rinv,dot,wd,factor_dpd,vdot;
  int *ilist,*jlist,*numneigh,**firstneigh;

  evdwl = 0.0;
  if (eflag || vflag) ev_setup(eflag,vflag);
  else evflag = vflag_fdotr = 0;

  double **x = atom->x;
  double **v = atom->v;
  double **f = atom->f;
  int *type = atom->type;
  int nlocal = atom->nlocal;
  double *special_lj = force->special_lj;
  int newton_pair = force->newton_pair;

  inum = list->inum;
  ilist = list->ilist;
  numneigh = list->numneigh;
  firstneigh = list->firstneigh;

  // loop over neighbors of my atoms

  for (ii = 0; ii < inum; ii++) {
    i = ilist[ii];
    xtmp = x[i][0];
    ytmp = x[i][1];
```

```
    ztmp = x[i][2];
    vxtmp = v[i][0];
    vytmp = v[i][1];
    vztmp = v[i][2];
    itype = type[i];
    jlist = firstneigh[i];
    jnum = numneigh[i];

        vdot = vxtmp*vxtmp + vytmp*vytmp + vztmp*vztmp; //flock

    for (jj = 0; jj < jnum; jj++) {
      j = jlist[jj];
      factor_dpd = special_lj[sbmask(j)];
      j &= NEIGHMASK;

      delx = xtmp - x[j][0];
      dely = ytmp - x[j][1];
      delz = ztmp - x[j][2];
      rsq = delx*delx + dely*dely + delz*delz;
      jtype = type[j];

      if (rsq < cutsq[itype][jtype]) {
        r = sqrt(rsq);
        if (r < EPSILON) continue;     // r can be 0.0 in DPD systems
        rinv = 1.0/r;
        delvx = vxtmp - v[j][0];
        delvy = vytmp - v[j][1];
        delvz = vztmp - v[j][2];
        dot = delx*delvx + dely*delvy + delz*delvz;

        wd = 1.0 - r/cut[itype][jtype];

        fpair = a0[itype][jtype]*wd;
        fpair -= gamma[itype][jtype]*wd*wd*dot*rinv;
        fpair *= factor_dpd*rinv;

        f[i][0] += delx*fpair;
        f[i][1] += dely*fpair;
        f[i][2] += delz*fpair;
        if (newton_pair    j < nlocal) {
          f[j][0] -= delx*fpair;
          f[j][1] -= dely*fpair;
          f[j][2] -= delz*fpair;
        }



        if (eflag) {
          evdwl = 0.5*a0[itype][jtype]*cut[itype][jtype] * wd*wd;
          evdwl *= factor_dpd;
        }

        if (evflag) ev_tally(i,j,nlocal,newton_pair,
                             evdwl,0.0,fpair,delx,dely,delz);
      }
    }

    // flocking force i = ( alpha -beta * (veli dot vel i) )  vec(veli)
    //beginflock
    f[i][0] += (alpha[itype]-beta[itype]*vdot)*vxtmp;
    f[i][1] += (alpha[itype]-beta[itype]*vdot)*vytmp;
    f[i][2] += (alpha[itype]-beta[itype]*vdot)*vztmp;
        //endflock

  }

  if (vflag_fdotr) virial_fdotr_compute();
}

/* ----------------------------------------------------------------------
   allocate all arrays
------------------------------------------------------------------------- */

void PairDPDFlock::allocate()
{
  allocated = 1;
  int n = atom->ntypes;

  memory->create(setflag,n+1,n+1,"pair:setflag");
  for (int i = 1; i <= n; i++)
```

```
    for (int j = i; j <= n; j++)
      setflag[i][j] = 0;

  memory->create(cutsq,n+1,n+1,"pair cutsq");

  memory->create(cut,n+1,n+1,"pair cut");
  memory->create(a0,n+1,n+1,"pair a0");
  memory->create(gamma,n+1,n+1,"pair gamma");
  memory->create(alpha,n+1,"pair alpha"); //flock
  memory->create(beta,n+1,"pair beta");  //flock
}

/* ----------------------------------------------------------------------
   global settings
------------------------------------------------------------------------- */

void PairDPDFlock::settings(int narg, char **arg)
{
  if (narg != 2) error->all(FLERR,"Illegal pair style command");

  temperature = force->numeric(FLERR,arg[0]);
  cut_global = force->numeric(FLERR,arg[1]);

  if (allocated) {
    int i,j;
    for (i = 1; i <= atom->ntypes; i++)
      for (j = i+1; j <= atom->ntypes; j++)
        if (setflag[i][j]) cut[i][j] = cut_global;
  }
}

/* ----------------------------------------------------------------------
   set coeffs for one or more type pairs
------------------------------------------------------------------------- */

void PairDPDFlock::coeff(int narg, char **arg)
{
  if (narg < 6 || narg > 7)
    error->all(FLERR,"Incorrect args for pair coefficients");
  if (!allocated) allocate();

  int ilo,ihi,jlo,jhi;
  force->bounds(arg[0],atom->ntypes,ilo,ihi);
  force->bounds(arg[1],atom->ntypes,jlo,jhi);

  double a0_one = force->numeric(FLERR,arg[2]);
  double gamma_one = force->numeric(FLERR,arg[3]);

  double alpha_one = force->numeric(FLERR,arg[4]); // flock
  double beta_one = force->numeric(FLERR,arg[5]);        //flock

  double cut_one = cut_global;
  if (narg == 7) cut_one = force->numeric(FLERR,arg[6]);

  int count = 0;
  for (int i = ilo; i <= ihi; i++) {
    for (int j = MAX(jlo,i); j <= jhi; j++) {
      a0[i][j] = a0_one;
      gamma[i][j] = gamma_one;
      alpha[i] = alpha_one; // flock
      beta[i] = beta_one; // flock
      cut[i][j] = cut_one;
      setflag[i][j] = 1;
      count++;
    }
  }

  if (count == 0) error->all(FLERR,"Incorrect args for pair coefficients");
}

/* ----------------------------------------------------------------------
   init specific to this pair style
------------------------------------------------------------------------- */

void PairDPDFlock::init_style()
{
  if (comm->ghost_velocity == 0)
    error->all(FLERR,"Pair dpd requires ghost atoms store velocity");

  // if newton off, forces between atoms ij will be double computed
```

```
  // using different random numbers

  if (force->newton_pair == 0 && comm->me == 0) error->warning(FLERR,
      "Pair dpd needs newton pair on for momentum conservation");

  neighbor->request(this,instance_me);
}

/* ----------------------------------------------------------------------
   init for one type pair i,j and corresponding j,i
------------------------------------------------------------------------- */

double PairDPDFlock::init_one(int i, int j)
{
  if (setflag[i][j] == 0) error->all(FLERR,"All pair coeffs are not set");

  cut[j][i] = cut[i][j];
  a0[j][i] = a0[i][j];
  gamma[j][i] = gamma[i][j];

  return cut[i][j];
}

/* ----------------------------------------------------------------------
   proc 0 writes to restart file
------------------------------------------------------------------------- */

void PairDPDFlock::write_restart(FILE *fp)
{
  write_restart_settings(fp);

  int i,j;
  for (i = 1; i <= atom->ntypes; i++)
    for (j = i; j <= atom->ntypes; j++) {
      fwrite(&setflag[i][j],sizeof(int),1,fp);
      if (setflag[i][j]) {
        fwrite(&a0[i][j],sizeof(double),1,fp);
        fwrite(&gamma[i][j],sizeof(double),1,fp);
        fwrite(&alpha[i],sizeof(double),1,fp); // flock
        fwrite(&beta[i],sizeof(double),1,fp);  // flock
        fwrite(&cut[i][j],sizeof(double),1,fp);
      }
    }
}

/* ----------------------------------------------------------------------
   proc 0 reads from restart file, bcasts
------------------------------------------------------------------------- */

void PairDPDFlock::read_restart(FILE *fp)
{
  read_restart_settings(fp);

  allocate();

  int i,j;
  int me = comm->me;
  for (i = 1; i <= atom->ntypes; i++)
    for (j = i; j <= atom->ntypes; j++) {
      if (me == 0) fread(&setflag[i][j],sizeof(int),1,fp);
      MPI_Bcast(&setflag[i][j],1,MPI_INT,0,world);
      if (setflag[i][j]) {
        if (me == 0) {
          fread(&a0[i][j],sizeof(double),1,fp);
          fread(&gamma[i][j],sizeof(double),1,fp);
          fread(&alpha[i],sizeof(double),1,fp); // flock
          fread(&beta[i],sizeof(double),1,fp); // flock
          fread(&cut[i][j],sizeof(double),1,fp);
        }
        MPI_Bcast(&a0[i][j],1,MPI_DOUBLE,0,world);
        MPI_Bcast(&gamma[i][j],1,MPI_DOUBLE,0,world);
        MPI_Bcast(&alpha[i],1,MPI_DOUBLE,0,world); // flock
        MPI_Bcast(&beta[i],1,MPI_DOUBLE,0,world); // flock
        MPI_Bcast(&cut[i][j],1,MPI_DOUBLE,0,world);
      }
    }
}

/* ----------------------------------------------------------------------
   proc 0 writes to restart file
```

```
------------------------------------------------------------------------- */

void PairDPDFlock::write_restart_settings(FILE *fp)
{
  fwrite(&temperature,sizeof(double),1,fp);
  fwrite(&cut_global,sizeof(double),1,fp);
  fwrite(&mix_flag,sizeof(int),1,fp);
}

/* ----------------------------------------------------------------------
   proc 0 reads from restart file, bcasts
------------------------------------------------------------------------- */

void PairDPDFlock::read_restart_settings(FILE *fp)
{
  if (comm->me == 0) {
    fread(&temperature,sizeof(double),1,fp);
    fread(&cut_global,sizeof(double),1,fp);
    fread(&mix_flag,sizeof(int),1,fp);
  }
  MPI_Bcast(&temperature,1,MPI_DOUBLE,0,world);
  MPI_Bcast(&cut_global,1,MPI_DOUBLE,0,world);
  MPI_Bcast(&mix_flag,1,MPI_INT,0,world);

}

/* ----------------------------------------------------------------------
   proc 0 writes to data file
------------------------------------------------------------------------- */

void PairDPDFlock::write_data(FILE *fp)
{
  for (int i = 1; i <= atom->ntypes; i++)
    fprintf(fp,"%d %g %g %g %g\n",i,a0[i][i],gamma[i][i],alpha[i],beta[i]);
}

/* ----------------------------------------------------------------------
   proc 0 writes all pairs to data file
------------------------------------------------------------------------- */

void PairDPDFlock::write_data_all(FILE *fp)
{
  for (int i = 1; i <= atom->ntypes; i++)
    for (int j = i; j <= atom->ntypes; j++)
      fprintf(fp,"%d %d %g %g %g %g %g\n",i,j,a0[i][j],gamma[i][j],alpha[i],beta[i],cut[i][j]);
}

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

double PairDPDFlock::single(int i, int j, int itype, int jtype, double rsq,
                            double factor_coul, double factor_dpd, double &fforce)
{
  double r,rinv,wd,phi;

  r = sqrt(rsq);
  if (r < EPSILON) {
    fforce = 0.0;
    return 0.0;
  }

  rinv = 1.0/r;
  wd = 1.0 - r/cut[itype][jtype];
  fforce = a0[itype][jtype]*wd * factor_dpd*rinv;

  phi = 0.5*a0[itype][jtype]*cut[itype][jtype] * wd*wd;
  return factor_dpd*phi;
}
```

Arquivo *pair_dpd_WR.h*:

```
/* -*- c++ -*- ----------------------------------------------------------
   Pairwise interaction from Hinz et al. 2014. Passive-Active interaction (Header file)
------------------------------------------------------------------------- */

#ifdef PAIR_CLASS

PairStyle(dpd/wr,PairDPDWR)

#else
```

```
#ifndef LMP PAIR DPD WR H
#define LMP PAIR DPD WR H

#include "pair h"

namespace LAMMPS NS {

class PairDPDWR   public Pair {
 public
  PairDPDWR(class LAMMPS *);
  virtual ~PairDPDWR();
  virtual void compute(int, int);
  virtual void settings(int, char **);
  virtual void coeff(int, char **);
  void init style();
  double init one(int, int);
  virtual void write restart(FILE *);
  virtual void read restart(FILE *);
  virtual void write restart settings(FILE *);
  virtual void read restart settings(FILE *);
  virtual void write data(FILE *);
  virtual void write data all(FILE *);
  double single(int, int, int, int, double, double, double, double &);

 protected
  double cut global,temperature;
  double **cut;
  double **a0,**gamma;

  void allocate();
};

}

#endif
#endif
```

Arquivo *pair_dpd_WR.cpp*:

```
/* ----------------------------------------------------------------------

   Pairwise iteraction from Hinz et al. 2014. Passive-Active interaction

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

#include <math h>
#include <stdio h>
#include <stdlib h>
#include "pair dpdWR h"
#include "atom h"
#include "atom vec h"
#include "comm h"
#include "update h"
#include "force h"
#include "neighbor h"
#include "neigh list h"
#include "memory h"
#include "error h"

using namespace LAMMPS NS;

#define EPSILON 1 0e-10

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

PairDPDWR  PairDPDWR(LAMMPS *lmp)   Pair(lmp)
{
  writedata = 1;
}

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

PairDPDWR  ~PairDPDWR()
{
  if (allocated) {
    memory->destroy(setflag);
    memory->destroy(cutsq);

    memory->destroy(cut);
    memory->destroy(a0);
```

```
    memory->destroy(gamma);
  }

}

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

void PairDPDWR::compute(int eflag, int vflag)
{
  int i,j,ii,jj,inum,jnum,itype,jtype;
  double xtmp,ytmp,ztmp,delx,dely,delz,evdwl,fpair;
  double vxtmp,vytmp,vztmp,delvx,delvy,delvz;
  double rsq,r,rinv,dot,wd,factor_dpd,vdot;
  int *ilist,*jlist,*numneigh,**firstneigh;

  evdwl = 0.0;
  if (eflag || vflag) ev_setup(eflag,vflag);
  else evflag = vflag_fdotr = 0;

  double **x = atom->x;
  double **v = atom->v;
  double **f = atom->f;
  int *type = atom->type;
  int nlocal = atom->nlocal;
  double *special_lj = force->special_lj;
  int newton_pair = force->newton_pair;

  inum = list->inum;
  ilist = list->ilist;
  numneigh = list->numneigh;
  firstneigh = list->firstneigh;

  // loop over neighbors of my atoms

  for (ii = 0; ii < inum; ii++) {
    i = ilist[ii];
    xtmp = x[i][0];
    ytmp = x[i][1];
    ztmp = x[i][2];
    vxtmp = v[i][0];
    vytmp = v[i][1];
    vztmp = v[i][2];
    itype = type[i];
    jlist = firstneigh[i];
    jnum = numneigh[i];

    for (jj = 0; jj < jnum; jj++) {
      j = jlist[jj];
      factor_dpd = special_lj[sbmask(j)];
      j &= NEIGHMASK;

      delx = xtmp - x[j][0];
      dely = ytmp - x[j][1];
      delz = ztmp - x[j][2];
      rsq = delx*delx + dely*dely + delz*delz;
      jtype = type[j];

      if (rsq < cutsq[itype][jtype]) {
        r = sqrt(rsq);
        if (r < EPSILON) continue;     // r can be 0.0 in DPD systems
        rinv = 1.0/r;
        delvx = vxtmp - v[j][0];
        delvy = vytmp - v[j][1];
        delvz = vztmp - v[j][2];
        dot = delx*delvx + dely*delvy + delz*delvz;
        wd = 1.0 - r/cut[itype][jtype];


        fpair = a0[itype][jtype]*wd;
        fpair -= gamma[itype][jtype]*wd*wd*dot*rinv;
        fpair *= factor_dpd*rinv;

        f[i][0] += delx*fpair;
        f[i][1] += dely*fpair;
        f[i][2] += delz*fpair;
        if (newton_pair || j < nlocal) {
          f[j][0] -= delx*fpair;
          f[j][1] -= dely*fpair;
          f[j][2] -= delz*fpair;
        }
```

```
        if (eflag) {
          evdwl = 0.5*a0[itype][jtype]*cut[itype][jtype] * wd*wd;
          evdwl *= factor_dpd;
        }

        if (evflag) ev_tally(i,j,nlocal,newton_pair,
                             evdwl,0.0,fpair,delx,dely,delz);
      }
    }
  }

  if (vflag_fdotr) virial_fdotr_compute();
}

/* ----------------------------------------------------------------------
   allocate all arrays
------------------------------------------------------------------------- */

void PairDPDWR::allocate()
{
  allocated = 1;
  int n = atom->ntypes;

  memory->create(setflag,n+1,n+1,"pair:setflag");
  for (int i = 1; i <= n; i++)
    for (int j = i; j <= n; j++)
      setflag[i][j] = 0;

  memory->create(cutsq,n+1,n+1,"pair:cutsq");

  memory->create(cut,n+1,n+1,"pair:cut");
  memory->create(a0,n+1,n+1,"pair:a0");
  memory->create(gamma,n+1,n+1,"pair:gamma");

}

/* ----------------------------------------------------------------------
   global settings
------------------------------------------------------------------------- */

void PairDPDWR::settings(int narg, char **arg)
{
  if (narg != 2) error->all(FLERR,"Illegal pair_style command");

  temperature = force->numeric(FLERR,arg[0]);
  cut_global = force->numeric(FLERR,arg[1]);


  if (allocated) {
    int i,j;
    for (i = 1; i <= atom->ntypes; i++)
      for (j = i+1; j <= atom->ntypes; j++)
        if (setflag[i][j]) cut[i][j] = cut_global;
  }
}

/* ----------------------------------------------------------------------
   set coeffs for one or more type pairs
------------------------------------------------------------------------- */

void PairDPDWR::coeff(int narg, char **arg)
{
  if (narg < 4 || narg > 5)
    error->all(FLERR,"Incorrect args for pair coefficients");
  if (!allocated) allocate();

  int ilo,ihi,jlo,jhi;
  force->bounds(arg[0],atom->ntypes,ilo,ihi);
  force->bounds(arg[1],atom->ntypes,jlo,jhi);

  double a0_one = force->numeric(FLERR,arg[2]);
  double gamma_one = force->numeric(FLERR,arg[3]);

  double cut_one = cut_global;
  if (narg == 5) cut_one = force->numeric(FLERR,arg[4]);

  int count = 0;
  for (int i = ilo; i <= ihi; i++) {
    for (int j = MAX(jlo,i); j <= jhi; j++) {
```

```
      a0[i][j] = a0_one;
      gamma[i][j] = gamma_one;
      cut[i][j] = cut_one;
      setflag[i][j] = 1;
      count++;
    }
  }

  if (count == 0) error->all(FLERR,"Incorrect args for pair coefficients");
}

/* ----------------------------------------------------------------------
   init specific to this pair style
------------------------------------------------------------------------- */

void PairDPDWR::init_style()
{
  if (comm->ghost_velocity == 0)
    error->all(FLERR,"Pair dpd requires ghost atoms store velocity");


  if (force->newton_pair == 0 && comm->me == 0) error->warning(FLERR,
      "Pair dpd needs newton pair on for momentum conservation");

  neighbor->request(this,instance_me);
}

/* ----------------------------------------------------------------------
   init for one type pair i,j and corresponding j,i
------------------------------------------------------------------------- */

double PairDPDWR::init_one(int i, int j)
{
  if (setflag[i][j] == 0) error->all(FLERR,"All pair coeffs are not set");


  cut[j][i] = cut[i][j];
  a0[j][i] = a0[i][j];
  gamma[j][i] = gamma[i][j];

  return cut[i][j];
}

/* ----------------------------------------------------------------------
   proc 0 writes to restart file
------------------------------------------------------------------------- */

void PairDPDWR::write_restart(FILE *fp)
{
  write_restart_settings(fp);

  int i,j;
  for (i = 1; i <= atom->ntypes; i++)
    for (j = i; j <= atom->ntypes; j++) {
      fwrite(&setflag[i][j],sizeof(int),1,fp);
      if (setflag[i][j]) {
        fwrite(&a0[i][j],sizeof(double),1,fp);
        fwrite(&gamma[i][j],sizeof(double),1,fp);
        fwrite(&cut[i][j],sizeof(double),1,fp);
      }
    }
}

/* ----------------------------------------------------------------------
   proc 0 reads from restart file, bcasts
------------------------------------------------------------------------- */

void PairDPDWR::read_restart(FILE *fp)
{
  read_restart_settings(fp);

  allocate();

  int i,j;
  int me = comm->me;
  for (i = 1; i <= atom->ntypes; i++)
    for (j = i; j <= atom->ntypes; j++) {
      if (me == 0) fread(&setflag[i][j],sizeof(int),1,fp);
      MPI_Bcast(&setflag[i][j],1,MPI_INT,0,world);
      if (setflag[i][j]) {
```

```
        if (me == 0) {
          fread(&a0[i][j],sizeof(double),1,fp);
          fread(&gamma[i][j],sizeof(double),1,fp);
          fread(&cut[i][j],sizeof(double),1,fp);
        }
        MPI_Bcast(&a0[i][j],1,MPI_DOUBLE,0,world);
        MPI_Bcast(&gamma[i][j],1,MPI_DOUBLE,0,world);
        MPI_Bcast(&cut[i][j],1,MPI_DOUBLE,0,world);
      }
    }
}

/* ----------------------------------------------------------------------
   proc 0 writes to restart file
------------------------------------------------------------------------- */

void PairDPDWR::write_restart_settings(FILE *fp)
{
  fwrite(&temperature,sizeof(double),1,fp);
  fwrite(&cut_global,sizeof(double),1,fp);
  fwrite(&mix_flag,sizeof(int),1,fp);
  //fwrite(&seed,sizeof(int),1,fp);
}

/* ----------------------------------------------------------------------
   proc 0 reads from restart file, bcasts
------------------------------------------------------------------------- */

void PairDPDWR::read_restart_settings(FILE *fp)
{
  if (comm->me == 0) {
    fread(&temperature,sizeof(double),1,fp);
    fread(&cut_global,sizeof(double),1,fp);
    fread(&mix_flag,sizeof(int),1,fp);
  }
  MPI_Bcast(&temperature,1,MPI_DOUBLE,0,world);
  MPI_Bcast(&cut_global,1,MPI_DOUBLE,0,world);
  MPI_Bcast(&mix_flag,1,MPI_INT,0,world);

}

/* ----------------------------------------------------------------------
   proc 0 writes to data file
------------------------------------------------------------------------- */

void PairDPDWR::write_data(FILE *fp)
{
  for (int i = 1; i <= atom->ntypes; i++)
    fprintf(fp,"%d %g %g\n",i,a0[i][i],gamma[i][i]);
}

/* ----------------------------------------------------------------------
   proc 0 writes all pairs to data file
------------------------------------------------------------------------- */

void PairDPDWR::write_data_all(FILE *fp)
{
  for (int i = 1; i <= atom->ntypes; i++)
    for (int j = i; j <= atom->ntypes; j++)
      fprintf(fp,"%d %d %g %g %g\n",i,j,a0[i][j],gamma[i][j],cut[i][j]);
}

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

double PairDPDWR::single(int i, int j, int itype, int jtype, double rsq,
                         double factor_coul, double factor_dpd, double &fforce)
{
  double r,rinv,wd,phi;

  r = sqrt(rsq);
  if (r < EPSILON) {
    fforce = 0.0;
    return 0.0;
  }

  rinv = 1.0/r;
  wd = 1.0 - r/cut[itype][jtype];
  fforce = a0[itype][jtype]*wd * factor_dpd*rinv;

  phi = 0.5*a0[itype][jtype]*cut[itype][jtype] * wd*wd;
```

```
  return factor dpd*phi;
}
```

Apêndice B

Exemplo de arquivo de entrada do LAMMPS

Exemplo de arquivo de entrada LAMMPS.

```
# DPD lammps file example

variable rcorte equal 1.0
variable L equal ${rcorte}*100

print cut=${rcorte}
print dim=${L}

dimension       2
# unitless see doc
units           lj
comm_modify     vel yes

#liga ou desliga terceira lei de newton parece que somente para acelerar computacao
newton          off

#coarse grain liquids
atom_style      atomic

#how the neighbor list should be created
neighbor        0.3 bin

#list neighbors is updated each  4 steps
neigh_modify    delay 0 every 4 check no

# define a regiao ID=domain style=block  #even being 2d we should define a 3d region, so z-axis is thin
region          domain block 0 ${L} 0 ${L} -0.25 0.25 units box


#boundary x y z
#p is periodic
#f is non-periodic and fixed
#s is non-periodic and shrink-wrapped
#m is non-periodic and shrink-wrapped with a minimum value
boundary p p p

# cria caixa de simulacao indica numero de tipos de atomos na caixa
create_box      2 domain

#define um lattice
lattice         hex 1

# cria atomos tipo random Numero de particulas semente gerador numero aleatorio regiao
create_atoms    1 random 2500 15255 domain #bacterias
create_atoms    2 random 22500 47214 domain

#massa particulas todas=* valor
mass            * 1.0


#pair_style dpd kbT(temperatura) raio_corte semente numero_aleatorio
```

```
pair_style        hybrid dpd 1.0 ${rcorte} 18589478 dpd/flock 1.0 ${rcorte} dpd/wr 1.0 ${rcorte}

#coeficientes de cada par de particulas
#pair_coeff type1 type2 A gamma [rc]
# A  constante forca conservativa
# gamma  forca da dissipacao
# rc(opicional)    raio de corte de um par especifico
pair_coeff       1 1  dpd/flock 25.0 4.5 4.995 2.25
pair_coeff       1 2  dpd/wr 25.0 4.5
pair_coeff       2 2  dpd 25.0 4.5


# mostra estado termodinamico a cada x passos
thermo_style custom step time temp pe ke etotal
thermo           2000

#velocidade inicial das particulas
velocity         all set 0.0 0.0 0.0

#usa o integrador NVE ver doc
#constant volume, constant energy simulations using the standard Velocity-Verlet method
fix              integrator all nve

#passo de tempo
timestep         0.003

#roda com numero de passos
run              2000000
```

Apêndice C

Modulos do DL_MESO

Modificaçãoes nos módulos do DL_MESO. Basta substituir as funções abaixo nos módulos *field_module.f90* e *config_module.f90* na pasta DPD, e recompilar o DL_MESO.

```
MODULE field_module

 ******************************************************************

  dl_meso module for link-cell setup and force calculation

 ******************************************************************

      USE constants
      USE variables
      USE error_module
      USE bond_module
      USE manybody_module
      USE ewald_module
      USE surface_module
      IMPLICIT none

CONTAINS

      SUBROUTINE conservativeforce (i, j, k, rrr, rsq, gforce, pot)

 ******************************************************************

      calculation of conservative interaction force for particle pair

 ******************************************************************

      REAL(KIND=dp)     gforce, pot, rrr, rrrr, rsq, rrsq, vk0, vk1, vk2
      REAL(KIND=dp)     aa, bb, rhoi, rhoj, scrn
      INTEGER     i, j, k

      gforce = 0.0_dp
      pot = 0.0_dp

      SELECT CASE (ktype(k))
      CASE (0)

      lennard-jones potential

        rrsq = 1.0_dp / rsq
        vk0 = vvv(1, k)
        vk1 = (vvv(3, k) * rrsq)**3

        pot = vk0 * vk1 * (vk1 - 1.0_dp)
        gforce = 6.0_dp * vk0 * vk1 * (2.0_dp * vk1 - 1.0_dp) * rrsq

      CASE (1)
        IF (rsq<=vvv(4, k)) THEN

      wca potential
```

```
      rrsq = 1 0 dp / rsq
      vk0 = vvv (1, k)
      vk1 = (vvv (3, k) * rrsq)**3
      vk2 = vvv (5, k)

      pot = vk0 * vk1 * (vk1 - 1 0 dp) + vk2
      gforce = 6 0 dp * vk0 * vk1 * (2 0 dp * vk1 - 1 0 dp) * rrsq

    END IF

  CASE (2)
    IF (rsq<=vvv (3, k)) THEN

  espanol and warren potential

      vk0 = vvv (1, k)
      vk1 = vvv (2, k)
      vk2 = 1 0 dp / vk1
      scrn = 1 0 dp / rrr - vk2

      pot = 0 5 dp * vk0 * (vk1 - rrr) * (vk1 - rrr) * vk2
      gforce = vk0 * scrn

    END IF

  CASE (3)
    IF (rsq<=vvv (7, k)) THEN

  many-body dpd  two-term model for vapour-liquid co-existence
  warren, pre 68, 066702 (2003)

      rrrr = 1 0 dp / rrr
      aa = vvv (1, k)
      bb = vvv (2, k)
      vk1 = vvv (6, k)
      vk2 = 1 0 dp / vk1

  calculate 'standard dpd' force and potential energy using first term

      scrn = rrrr - vk2
      pot = 0 5 dp * aa * (vk1 - rrr) * (vk1 - rrr) * vk2
      gforce = aa * scrn

  calculate additional density-dependent force

      IF (rrr<rmbcut) THEN

        rhoi = rhomb (i, ltp (i))
        rhoj = rhomb (j, ltp (j))

        gforce = gforce + bb * (rhoi + rhoj) * (rrrr - rrmbcut)

      END IF

    END IF

  CASE (4)
    IF (rsq<=vvv (3, k)) THEN

  case hinz model

      vk0 = vvv (1, k)
      vk1 = vvv (2, k)
      vk2 = 1 0 dp / vk1
      scrn = 1 0 dp / rrr - vk2

      pot = 0 5 dp * vk0 * (vk1 - rrr) * (vk1 - rrr) * vk2
      gforce = vk0 * scrn

    END IF
   CASE (5)
    IF (rsq<=vvv (3, k)) THEN

  case no random force

      vk0 = vvv (1, k)
      vk1 = vvv (2, k)
      vk2 = 1 0 dp / vk1
      scrn = 1 0 dp / rrr - vk2
```

```
        pot = 0.5_dp * vk0 * (vk1 - rrr) * (vk1 - rrr) * vk2
        gforce = vk0 * scrn

      END IF

    END SELECT

    RETURN
    END SUBROUTINE conservativeforce


    SUBROUTINE forces_dpdvv (nlimit)

!***********************************************************************

    calculation of interaction forces for dpd velocity verlet
    integrator with dpd thermostat


!***********************************************************************

    USE numeric_container

    REAL(KIND=dp)     ab, ai, aj, gforce, pot, rrr, rsq, ran , vdotv
    REAL(KIND=dp)     xdif, ydif, zdif, rforce, dforce, sumforce, scrn
    REAL(KIND=dp)     strsxx, strsxy, strsxz, strsyy, strsyz, strszz

    INTEGER     i, ibox, icell, ipair, j, jbox, k, kc, nlimit, totalcells
    INTEGER     npart, npairs, pairbuf, mxpairs, fail

    INTEGER, ALLOCATABLE     lpart ( , ), tload ( ), pairs ( , )
    REAL(KIND=dp), ALLOCATABLE     pairdata ( , )

    initialise accumulators

    pe = 0.0_dp
    vir = 0.0_dp
    ivrl = 0.0_dp
    stress = 0.0_dp
    strsxx = 0.0_dp
    strsxy = 0.0_dp
    strsxz = 0.0_dp
    strsyy = 0.0_dp
    strsyz = 0.0_dp
    strszz = 0.0_dp
    fxx = 0.0_dp
    fyy = 0.0_dp
    fzz = 0.0_dp
    fvx = 0.0_dp
    fvy = 0.0_dp
    fvz = 0.0_dp
    vdotv = 0.0_dp

    determine total number of cells to search, estimate maximum number of
    interacting particle pairs

    totalcells = nlx * nly * nlz
    pairbuf = mxpcell * (mxpcell - 1) / 2 + 13 * mxpcell * mxpcell

    allocate per-cell arrays for loaded particles, numbers of loaded particles
    and arrays for interacting particle pairs

    ALLOCATE (lpart (mxpcell, 14), tload (14), pairs (3, pairbuf), pairdata (4, pairbuf), STAT=fail)
    IF (fail/=0) CALL error (idnode, 1183, 1)

    primary loop over all cells

    DO icell = 1, totalcells

      lpart = 0
      tload = -1
      ibox = lcell (icell)
      CALL loadpart (ibox, 1, lpart, tload)
      npart = tload (1)

    bypass cell if empty

      IF (tload (1)>0) THEN

    load neighbouring cells contents
```

```
    DO kc = 2, 14

      jbox = lcell neighbour (27*(icell-1)+kc)
      CALL loadpart (jbox, kc, lpart, tload)
      npart = npart + tload (kc)

    END DO

determine maximum number of interacting pairs (self and neighbours)

    mxpairs = tload (1) * (tload (1) - 1) / 2 + tload (1) * (npart - tload (1))

if necessary, reallocate storage for particle pairs and data

    IF (mxpairs>pairbuf) THEN
      DEALLOCATE (pairs, pairdata, STAT=fail)
      IF (fail/=0) CALL error (idnode, 1184, 2)
      ALLOCATE (pairs (3, mxpairs), pairdata (4, mxpairs), STAT=fail)
      IF (fail/=0) CALL error (idnode, 1183, 2)
      pairbuf = mxpairs
    END IF

find all interacting pairs, determining if thermostat is to be applied for
lees-edwards shear

    npairs = 0
    CALL diff (lpart, tload, pairs, pairdata, npairs, icell, 14)

calculate potentials and forces

    DO ipair = 1, npairs

      i = pairs (1, ipair)
      j = pairs (2, ipair)
      xdif = pairdata (1, ipair)
      ydif = pairdata (2, ipair)
      zdif = pairdata (3, ipair)
      rsq = pairdata (4, ipair)
      rrr = SQRT (rsq)
      ai = REAL (ltp (i), KIND=dp)
      aj = REAL (ltp (j), KIND=dp)
      IF (ai>aj) THEN
        ab = ai * (ai - 1 0 dp) * 0 5 dp + aj + 0 5 dp
      ELSE
        ab = aj * (aj - 1 0 dp) * 0 5 dp + ai + 0 5 dp
      END IF
      k = INT (ab)

calculate conservative interaction force and potential energy

      CALL conservativeforce (i, j, k, rrr, rsq, gforce, pot)
      pe = pe + pot

calculate random and drag forces (dpd thermostat)

      rforce = 0 0 dp
      dforce = 0 0 dp
      IF (pairs (3, ipair)>0) THEN
        ran = rt12 * (mtrnd (idnode) - 0 5 dp)
         ran = rt12 * (duni (idnode) - 0 5 dp)
         ran = gaussmp (idnode)
        scrn = 1 0 dp / rrr - rrtcut
        rforce = sigma (k) * scrn * ran
        dforce = xdif * (vxx (j) - vxx(i)) + ydif * (vyy (j) - vyy(i)) + zdif * (vzz (j) - vzz(i))
        dforce = -gamma (k) * scrn * scrn * dforce
      END IF

sum of forces

                  SELECT CASE(ktype(k))

      CASE (0, 1, 2, 3)
       normal DPD
      sumforce = gforce + rforce

                  CASE(4,5)
       Hinz et al (2014)
       and no random force interaction
                  sumforce = gforce
```

```
                    END SELECT


assign virial terms (dissipative terms deferred until
recalculation of dissipative forces)

       vir = vir - sumforce * rsq
       ivrl (1) = ivrl (1) - gforce * xdif * xdif
       ivrl (2) = ivrl (2) - gforce * ydif * ydif
       ivrl (3) = ivrl (3) - gforce * zdif * zdif

assign stress terms

       strsxx = strsxx + sumforce * xdif * xdif
       strsxy = strsxy + sumforce * xdif * ydif
       strsxz = strsxz + sumforce * xdif * zdif
       strsyy = strsyy + sumforce * ydif * ydif
       strsyz = strsyz + sumforce * ydif * zdif
       strszz = strszz + sumforce * zdif * zdif

assign force terms

       fxx (i) = fxx (i) - sumforce * xdif
       fyy (i) = fyy (i) - sumforce * ydif
       fzz (i) = fzz (i) - sumforce * zdif
       fxx (j) = fxx (j) + sumforce * xdif
       fyy (j) = fyy (j) + sumforce * ydif
       fzz (j) = fzz (j) + sumforce * zdif
       fvx (i) = fvx (i) - dforce * xdif
       fvy (i) = fvy (i) - dforce * ydif
       fvz (i) = fvz (i) - dforce * zdif
       fvx (j) = fvx (j) + dforce * xdif
       fvy (j) = fvy (j) + dforce * ydif
       fvz (j) = fvz (j) + dforce * zdif

       IF (ktype(k) == 4) THEN

        Hinz extra force

        print *, j,fxx (j),fyy (j),fzz (j)


       vdotv= vxx(j)* vxx(j) + vyy(j)* vyy(j) + vzz(j)*vzz(j)

       fxx (j) = fxx (j) + (vvv (4, k)  - vvv (5, k)* vdotv) *vxx(j)
       fyy (j) = fyy (j) + (vvv (4, k)  - vvv (5, k)* vdotv) *vyy(j)
       fzz (j) = fzz (j) + (vvv (4, k)  - vvv (5, k)* vdotv) *vzz(j)

        print *, j,fxx (j),fyy (j),fzz (j)
        print *,' '

       END IF

     END DO

   END IF

END DO

assign stress components to full tensor

stress (1) = stress (1) + strsxx
stress (2) = stress (2) + strsxy
stress (3) = stress (3) + strsxz
stress (4) = stress (4) + strsxy
stress (5) = stress (5) + strsyy
stress (6) = stress (6) + strsyz
stress (7) = stress (7) + strsxz
stress (8) = stress (8) + strsyz
stress (9) = stress (9) + strszz

deallocate arrays for loaded particles and pairs

DEALLOCATE (lpart, tload, pairs, pairdata, STAT=fail)
IF (fail/=0) CALL error (idnode, 1184, 1)

RETURN
END SUBROUTINE forces_dpdvv
```

```
MODULE config_module

 ******************************************************************

  dl_meso module for configuration

  copyright - stfc daresbury laboratory
  authors   - w. smith & m. a. seaton august 2015

 ******************************************************************

      USE constants
      USE variables
      USE comms_module
      USE numeric_container
      USE error_module
      IMPLICIT none

      LOGICAL     lnfold, ldyn

CONTAINS


      SUBROUTINE scan_field

 ******************************************************************

      scan FIELD file for array bounds

 ******************************************************************

      USE parse_utils

      LOGICAL     lexist, finish, safe, lrcut
      CHARACTER(LEN=mxword)     key, word
      CHARACTER(LEN=200)     record, record1
      INTEGER     i, ioerror, imol, ibond, iangle, idihed, ipot, j, bdtyp, agtyp, dhtyp
      REAL(KIND=dp)     aa, bb, cc, dd
      INTEGER, ALLOCATABLE     bdtypetmp(), agtypetmp(), dhtypetmp()
      REAL(KIND=dp), ALLOCATABLE     bdaatmp(), bdbbtmp(), bdcctmp(), bdddtmp()
      REAL(KIND=dp), ALLOCATABLE     agaatmp(), agbbtmp(), agcctmp(), agddtmp()
      REAL(KIND=dp), ALLOCATABLE     dhaatmp(), dhbbtmp(), dhcctmp(), dhddtmp()

      INTEGER     fail(8)

      record if cutoff radius has already been defined in CONTROL file

      lrcut = (.NOT. (rcut>1.0e-16_dp))

      check that FIELD file exists

      INQUIRE(file='FIELD', EXIST=lexist)
      IF(.NOT. lexist) CALL error(idnode, 30, 0)

      open FIELD channel for first pass: determine numbers of species and
      molecule types, maximum numbers of beads, bonds, angles and dihedrals,
      maximum number of parameters needed for interaction potentials
      and units for energy parameters

      OPEN(nread, file='FIELD', IOSTAT=ioerror)

      safe = (ioerror==0)
      CALL global_sca_and(safe, 0, idnode)
      IF(.NOT. safe) CALL error(idnode, 31, 0)

      mxmolsize = 0
      mxbonds = 0
      mxangles = 0
      mxdiheds = 0

      READ(nread, '(a80)') record

      finish = .false.

      DO WHILE(.NOT. finish)

        READ(nread, '(a200)', IOSTAT=ioerror) record

        IF(ioerror/=0) finish = .true.
```

```
key = getword (record, 1)
CALL lowercase (key)

IF (key (1 6) =="close") finish =  true

IF (key (1 5) =="speci") THEN

  nspe = INT (getint (record, 2), KIND=si)
  npot = nspe * (nspe + 1) / 2

ELSE IF (key (1 7) =="molecul") THEN

  nmoldef = INT (getint (record, 2), KIND=si)

  DO i = 1, nmoldef

    DO WHILE ( true )
      READ (nread, '(a200)', IOSTAT=ioerror) record1
      word = getword (record1, 1)
      CALL lowercase (word)

      IF (word (1 4) =="bead") THEN

        imol = INT (getint (record1, 2), KIND=si)
        mxmolsize = MAX (mxmolsize, imol)

      ELSE IF (word (1 4) =="bond") THEN

        ibond = INT (getint (record1, 2), KIND=si)
        mxbonds = MAX (mxbonds, ibond)

      ELSE IF (word (1 5) =="angle") THEN

        iangle = INT (getint (record1, 2), KIND=si)
        mxangles = MAX (mxangles, iangle)

      ELSE IF (word (1 5) =="dihed") THEN

        idihed = INT (getint (record1, 2), KIND=si)
        mxdiheds = MAX (mxdiheds, idihed)

      ELSE IF (word (1 6) =="finish") THEN

        EXIT

      END IF

    END DO

  END DO

ELSE IF (key (1 8) =="interact") THEN

  ipot = INT (getint (record, 2), KIND=si)
  mxprm = 0

  DO i = 1, ipot

    READ (nread, '(a200)', IOSTAT=ioerror) record1
    word = getword (record1, 3)
    CALL lowercase (word)
    IF (word (1 2) =="lj") THEN
      mxprm = MAX (mxprm, 3)
      IF (lrcut) rcut = MAX (rcut, getdble (record1, 5))
    ELSE IF (word (1 3) =="wca") THEN
      mxprm = MAX (mxprm, 5)
      IF (lrcut) rcut = MAX (rcut, getdble (record1, 5))
    ELSE IF (word (1 3) =="dpd") THEN
      mxprm = MAX (mxprm, 3)
      IF (lrcut) rcut = MAX (rcut, getdble (record1, 5))
    ELSE IF (word (1 4) =="mdpd") THEN
      mxprm = MAX (mxprm, 7)
      IF (lrcut) rcut = MAX (rcut, getdble (record1, 9))
    ELSE IF (word (1 4) =="hinz") THEN
      mxprm = MAX (mxprm, 5)
      IF (lrcut) rcut = MAX (rcut, getdble (record1, 5))
    ELSE IF (word (1 4) =="norf") THEN
      mxprm = MAX (mxprm, 3)
      IF (lrcut) rcut = MAX (rcut, getdble (record1, 5))
```

```
      END IF

    END DO

    ELSE IF (key (1 5) =="units") THEN

      word = getword (record, 2)
      CALL lowercase (word)
      IF (word (1 5)=="const") THEN
        engunit = 0
      ELSE IF (word (1 2) =="kt") THEN
        engunit = 1
      END IF

  END IF

END DO

CLOSE (nread)

nbonddef = 0
nangdef = 0
ndhddef = 0

fail = 0
ALLOCATE (bdtypetmp (nmoldef*mxbonds), agtypetmp (nmoldef*mxangles), dhtypetmp (nmoldef*mxdiheds), &
 & STAT=fail(1))
ALLOCATE (bdaatmp (nmoldef*mxbonds), bdbbtmp (nmoldef*mxbonds), STAT=fail(2))
ALLOCATE (bdcctmp (nmoldef*mxbonds), bdddtmp (nmoldef*mxbonds), STAT=fail(3))
ALLOCATE (agaatmp (nmoldef*mxangles), agbbtmp (nmoldef*mxangles), STAT=fail(4))
ALLOCATE (agcctmp (nmoldef*mxangles), agddtmp (nmoldef*mxangles), STAT=fail(5))
ALLOCATE (dhaatmp (nmoldef*mxdiheds), dhbbtmp (nmoldef*mxdiheds), STAT=fail(6))
ALLOCATE (dhcctmp (nmoldef*mxdiheds), dhddtmp (nmoldef*mxdiheds), STAT=fail(7))
IF (ANY (fail/=0)) CALL error (idnode, 1002, 1)

open FIELD file for second pass  determine numbers of bond, angle and
dihedral types and store parameters

OPEN (nread, file = 'FIELD', IOSTAT=ioerror)

safe = (ioerror==0)
 CALL global sca and (safe, 0, idnode)
IF ( NOT  safe) CALL error (idnode, 31, 0)

READ (nread, '(a80)') record

DO WHILE ( true )

  READ (nread, '(a200)', IOSTAT=ioerror) record

  IF (ioerror/=0) EXIT

  key = getword (record, 1)
  CALL lowercase (key)

  IF (key (1 6) =="close") EXIT

  IF (key (1 4) =="bond") THEN

    ibond = INT (getint (record, 2), KIND=si)

    DO i = 1, ibond

      READ (nread, '(a200)', IOSTAT=ioerror) record1
      word = getword (record1, 1)
      CALL lowercase (word)
      IF (word (1 4) =="harm") THEN
        bdtyp = 1
      ELSE IF (word (1 4) =="fene") THEN
        bdtyp = 2
      ELSE IF (word (1 3) =="wlc") THEN
        bdtyp = 3
      ELSE IF (word (1 4) =="mors") THEN
        bdtyp = 4
      ELSE
        CALL error (idnode, 32, 0)
      END IF
      aa = getdble (record1, 4)
      bb = getdble (record1, 5)
```

```
      cc = getdble (record1, 6)
      dd = getdble (record1, 7)
      lexist = false
      DO j = 1, nbonddef
        IF (bdtypetmp (j)==bdtyp AND bdaatmp (j)==aa AND bdbbtmp (j)==bb AND &
           &bdcctmp (j)==cc AND bdddtmp (j)==dd) THEN
          lexist = true
        END IF
      END DO
      IF ( NOT lexist) THEN
        nbonddef = nbonddef + 1
        bdtypetmp (nbonddef) = bdtyp
        bdaatmp (nbonddef) = aa
        bdbbtmp (nbonddef) = bb
        bdcctmp (nbonddef) = cc
        bdddtmp (nbonddef) = dd
      END IF

    END DO

  ELSE IF (key (1 5) =="angle") THEN

    iangle = INT (getint (record, 2), KIND=si)

    DO i = 1, iangle

      READ (nread, '(a200)', IOSTAT=ioerror) record1
      word = getword (record1, 1)
      CALL lowercase (word)
      IF (word (1 4) =="harm") THEN
        agtyp = 1
      ELSE IF (word (1 4) =="hcos") THEN
        agtyp = 2
      ELSE IF (word (1 3) =="cos") THEN
        agtyp = 3
      ELSE
        CALL error (idnode, 33, 0)
      END IF
      aa = getdble (record1, 5)
      bb = getdble (record1, 6)
      cc = getdble (record1, 7)
      dd = getdble (record1, 8)
      lexist = false
      DO j = 1, nangdef
        IF (agtypetmp (j)==agtyp AND agaatmp (j)==aa AND agbbtmp (j)==bb AND &
           &agcctmp (j)==cc AND agddtmp (j)==dd) THEN
          lexist = true
        END IF
      END DO
      IF ( NOT lexist) THEN
        nangdef = nangdef + 1
        agtypetmp (nangdef) = agtyp
        agaatmp (nangdef) = aa
        agbbtmp (nangdef) = bb
        agcctmp (nangdef) = cc
        agddtmp (nangdef) = dd
      END IF

    END DO

  ELSE IF (key (1 5) =="dihed") THEN

    idihed = INT (getint (record, 2), KIND=si)

    DO i = 1, idihed

      READ (nread, '(a200)', IOSTAT=ioerror) record1
      word = getword (record1, 1)
      CALL lowercase (word)
      IF (word (1 3) =="cos") THEN
        dhtyp = 1
      ELSE IF (word (1 4) =="harm") THEN
        dhtyp = 2
      ELSE IF (word (1 4) =="hcos") THEN
        dhtyp = 3
      ELSE
        CALL error (idnode, 34, 0)
      END IF
      aa = getdble (record1, 6)
      bb = getdble (record1, 7)
```

```
            cc = getdble (record1, 8)
            dd = getdble (record1, 9)
            lexist = false
            DO j = 1, ndhddef
              IF (dhtypetmp (j)==dhtyp AND dhaatmp (j)==aa AND dhbbtmp (j)==bb AND &
                 &dhcctmp (j)==cc AND dhddtmp (j)==dd) THEN
                lexist = true
              END IF
            END DO
            IF ( NOT lexist) THEN
              ndhddef = ndhddef + 1
              dhtypetmp (ndhddef) = dhtyp
              dhaatmp (ndhddef) = aa
              dhbbtmp (ndhddef) = bb
              dhcctmp (ndhddef) = cc
              dhddtmp (ndhddef) = dd
            END IF

          END DO

        END IF

      END DO

      CLOSE (nread)

      fail = 0
      ALLOCATE (bdtype (nbonddef), angtype (nangdef), dhdtype (ndhddef), STAT=fail(1))
      ALLOCATE (aabond (nbonddef), bbbond (nbonddef), ccbond (nbonddef), ddbond (nbonddef), STAT=fail(2))
      ALLOCATE (aaang (nangdef), bbang (nangdef), ccang (nangdef), ddang (nangdef), STAT=fail(3))
      ALLOCATE (aadhd (nangdef), bbdhd (nangdef), ccdhd (nangdef), dddhd (nangdef), STAT=fail(4))
      IF (ANY (fail/=0)) CALL error (idnode, 1002, 2)

      IF (nbonddef>0) THEN
        bdtype (1 nbonddef) = bdtypetmp (1 nbonddef)
        aabond (1 nbonddef) = bdaatmp (1 nbonddef)
        bbbond (1 nbonddef) = bdbbtmp (1 nbonddef)
        ccbond (1 nbonddef) = bdcctmp (1 nbonddef)
        ddbond (1 nbonddef) = bdddtmp (1 nbonddef)
        lbond = true
      END IF

      IF (nangdef>0) THEN
        angtype (1 nangdef) = agtypetmp (1 nangdef)
        aaang (1 nangdef) = agaatmp (1 nangdef)
        bbang (1 nangdef) = agbbtmp (1 nangdef)
        ccang (1 nangdef) = agcctmp (1 nangdef)
        ddang (1 nangdef) = agddtmp (1 nangdef)
        langle = true
      END IF

      IF (ndhddef>0) THEN
        dhdtype (1 ndhddef) = dhtypetmp (1 ndhddef)
        aadhd (1 ndhddef) = dhaatmp (1 ndhddef)
        bbdhd (1 ndhddef) = dhbbtmp (1 ndhddef)
        ccdhd (1 ndhddef) = dhcctmp (1 ndhddef)
        dddhd (1 ndhddef) = dhddtmp (1 ndhddef)
        ldihed = true
      END IF

      fail = 0
      DEALLOCATE (bdtypetmp, agtypetmp, dhtypetmp, STAT=fail(1))
      DEALLOCATE (bdaatmp, bdbbtmp, bdcctmp, bdddtmp, STAT=fail(2))
      DEALLOCATE (agaatmp, agbbtmp, agcctmp, agddtmp, STAT=fail(3))
      DEALLOCATE (dhaatmp, dhbbtmp, dhcctmp, dhddtmp, STAT=fail(4))
      IF (ANY (fail/=0)) CALL error (idnode, 1003, 0)

      RETURN

      END SUBROUTINE scan_field

      SUBROUTINE read_field

*****************************************************************************

      read FIELD file for interaction and molecule data

*****************************************************************************

      USE parse_utils
```

```
CHARACTER(LEN=8)     spenam
CHARACTER(LEN=mxword)     key, word, word1
CHARACTER(LEN=200)     record, record1
INTEGER     i, ibond, iang, idhd, ioerror, ispe, j, jspe, k, finmol, typ
REAL(KIND=dp)     aa, bb, cc, dd, ee, ff, gg, x0, y0, z0, maxside
REAL(KIND=dp)     el2, fac, pl2, frzwid, eunit
LOGICAL     safe
LOGICAL, ALLOCATABLE     interact ( , )

INTEGER     fail (13)

set energy units for conservative and bond interactions

SELECT CASE (engunit)
CASE (0)
  eunit = 1 0 dp
CASE (1)
  eunit = temp
END SELECT

open FIELD channel

OPEN (nread, file = 'FIELD', IOSTAT=ioerror)

safe = (ioerror==0)
 CALL global sca and (safe, 0, idnode)
IF ( NOT  safe) CALL error (idnode, 31, 0)

allocate arrays for reading in species, molecule and interaction data

fail = 0
ALLOCATE (namspe (nspe), STAT=fail(1))
ALLOCATE (amass (nspe), chge (nspe), STAT=fail(2))
ALLOCATE (nspec (nspe), nspecmol (nspe), lfrzn (nspe), ktype (npot), STAT=fail(3))
ALLOCATE (nammol (0 nmoldef), STAT=fail(4))
ALLOCATE (moliso (nmoldef), STAT=fail(5))
ALLOCATE (mlstrtspe (nmoldef, mxmolsize), nmol (nmoldef), nbdmol (nmoldef), STAT=fail(6))
ALLOCATE (nbond (nmoldef), nangle (nmoldef), ndihed (nmoldef), STAT=fail(7))
ALLOCATE (bdinp1 (nmoldef, mxbonds), bdinp2 (nmoldef, mxbonds), bdinp3 (nmoldef, mxbonds), STAT=fail(8))
ALLOCATE (anginp1 (nmoldef, mxangles), anginp2 (nmoldef, mxangles), anginp3 (nmoldef, mxangles), &
         &anginp4 (nmoldef, mxangles), STAT=fail(9))
ALLOCATE (dhdinp1 (nmoldef, mxdiheds), dhdinp2 (nmoldef, mxdiheds), dhdinp3 (nmoldef, mxdiheds), &
         &dhdinp4 (nmoldef, mxdiheds), dhdinp5 (nmoldef, mxdiheds), STAT=fail(10))
ALLOCATE (mlstrtxxx (nmoldef, mxmolsize), mlstrtyyy (nmoldef, mxmolsize), &
& mlstrtzzz (nmoldef, mxmolsize), STAT=fail(11))
ALLOCATE (cbsize (nmoldef), STAT=fail(12))

ALLOCATE (vvv (mxprm, npot), gamma (npot), sigma (npot), aasrf (nspe), interact (npot, 3), STAT=fail(13))

IF (ANY (fail/=0)) CALL error (idnode, 1004, 0)

nammol (0) = '         '
nsystcell = 0
nusystcell = 0
nfsystcell = 0
nspec = 0
nspecmol = 0
nbond = 0
nangle = 0
ndihed = 0
numbond = 0
numang = 0
numdhd = 0
numbondcell = 0
numangcell = 0
numdhdcell = 0
moliso =  true
interact =  false
clr = 0 0 dp
aasrf = 0 0 dp
frzwdens = 0 0 dp
frzwxwid = 0 0 dp
frzwywid = 0 0 dp
frzwzwid = 0 0 dp

skip over file header

READ (nread, '(a80)') record
```

```
read in species, molecular structures and interaction data

DO WHILE ( true )

  READ (nread, '(a200)', IOSTAT=ioerror) record

  IF (ioerror/=0) EXIT

  key = getword (record, 1)
  CALL lowercase (key)

  IF (key (1 5) =="close") EXIT

  IF (key (1 7) =="species") THEN

    DO i = 1, nspe

      READ (nread, '(a200)', IOSTAT=ioerror) record1
      word = getword (record1, 1)
      IF (LEN (TRIM (word))>8) CALL error (idnode, -35, i)
      namspe (i) = word (1 8)
      amass (i) = getdble (record1, 2)
      chge (i) = getdble (record1, 3)
      nspec (i) = INT (getint (record1, 4), KIND=si)
      lfrzn (i) = INT (getint (record1, 5), KIND=si)
      nusystcell = nusystcell + nspec (i)
      IF (lfrzn (i)>0) nfsystcell = nfsystcell + nspec (i)

    END DO
    nsystcell = nsystcell + nusystcell

  ELSE IF (key (1 7) =="molecul") THEN

    DO i = 1, nmoldef

      READ (nread, '(a200)', IOSTAT=ioerror) record1
      word = getword (record1, 1)
      IF (LEN (TRIM (word))>8) CALL error (idnode, -36, i)
      nammol (i) = word (1 8)

      DO WHILE ( true )

        READ (nread, '(a200)', IOSTAT=ioerror) record1
        word = getword (record1, 1)
        CALL lowercase (word)

        IF (word (1 6) =="nummol") THEN

          nmol (i) = INT (getint (record1, 2), KIND=si)

        ELSE IF (word (1 4) =="bead") THEN

          finmol = INT (getint (record1, 2), KIND=si)
          nbdmol (i) = finmol
          x0 = 0 0 dp
          y0 = 0 0 dp
          z0 = 0 0 dp

          DO j = 1, finmol

            READ (nread, '(a200)', IOSTAT = ioerror) record1
            ispe = 0
            word1 = getword (record1, 1)
            spenam = word1 (1 8)
            DO k = 1, nspe
              IF (spenam==namspe (k)) ispe = k
            END DO
            IF (ispe==0) ispe = INT (getint (record1, 1), KIND=si)
            safe = (ispe>0 AND ispe<=nspe)
            IF ( NOT safe) CALL error (idnode, 37, i)

            mlstrtspe (i, j) = ispe
            mlstrtxxx (i, j) = getdble (record1, 2)
            mlstrtyyy (i, j) = getdble (record1, 3)
            mlstrtzzz (i, j) = getdble (record1, 4)
            x0 = x0 + mlstrtxxx (i, j)
            y0 = y0 + mlstrtyyy (i, j)
            z0 = z0 + mlstrtzzz (i, j)

            nspecmol (ispe) = nspecmol (ispe) + nmol (i)
```

```
      IF (lfrzn (ispe)>0) nfsystcell = nfsystcell + nmol (i)

    END DO

    maxside = 0 0 dp
    x0 = x0 / REAL (finmol, KIND=dp)
    y0 = y0 / REAL (finmol, KIND=dp)
    z0 = z0 / REAL (finmol, KIND=dp)
    DO j = 1, finmol
      mlstrtxxx (i, j) = mlstrtxxx (i, j) - x0
      mlstrtyyy (i, j) = mlstrtyyy (i, j) - y0
      mlstrtzzz (i, j) = mlstrtzzz (i, j) - z0
      maxside = MAX (maxside, ABS (mlstrtxxx (i, j)), ABS (mlstrtyyy (i, j)), ABS (mlstrtzzz (i, j)))
    END DO
    cbsize (i) = maxside

  ELSE IF (word (1 4) =="bond") THEN

    finmol = INT (getint (record1, 2), KIND=si)
    nbond (i) = finmol

    DO j = 1, finmol

      READ (nread, '(a200)', IOSTAT = ioerror) record1
      word1 = getword (record1, 1)
      CALL lowercase (word1)
      typ = 0
      IF (word1 (1 4) =="harm") THEN
        typ = 1
      ELSE IF (word1 (1 4) =="fene") THEN
        typ = 2
      ELSE IF (word1 (1 3) =="wlc") THEN
        typ = 3
      ELSE IF (word1 (1 4) =="mors") THEN
        typ = 4
      ELSE
        CALL error (idnode, 32, 0)
      END IF
      aa = getdble (record1, 4)
      bb = getdble (record1, 5)
      cc = getdble (record1, 6)
      dd = getdble (record1, 7)
      ibond = 0
      DO k = 1, nbonddef
        IF (bdtype (k)==typ  AND  aabond (k)==aa  AND  bbbond (k)==bb  AND  &
           &ccbond (k)==cc  AND  ddbond (k)==dd) THEN
          ibond = k
        END IF
      END DO
      IF (ibond==0) CALL error (idnode, 38, i)
      bdinp1 (i, j) = INT (getint (record1, 2), KIND=si)
      bdinp2 (i, j) = INT (getint (record1, 3), KIND=si)
      bdinp3 (i, j) = ibond

    END DO
    numbondcell = numbondcell + nbond (i) * nmol (i)

  ELSE IF (word (1 5) =="angle") THEN

    finmol = INT (getint (record1, 2), KIND=si)
    nangle (i) = finmol

    DO j = 1, finmol

      READ (nread, '(a200)', IOSTAT = ioerror) record1
      word1 = getword (record1, 1)
      CALL lowercase (word1)
      typ = 0
      IF (word1 (1 4) =="harm") THEN
        typ = 1
      ELSE IF (word1 (1 4) =="hcos") THEN
        typ = 2
      ELSE IF (word1 (1 3) =="cos") THEN
        typ = 3
      ELSE
        CALL error (idnode, 33, 0)
      END IF
      aa = getdble (record1, 5)
      bb = getdble (record1, 6)
      cc = getdble (record1, 7)
```

```
            dd = getdble (record1, 8)
            iang = 0
            DO k = 1, nangdef
              IF (angtype (k)==typ  AND  aaang (k)==aa  AND  bbang (k)==bb  AND  &
                 &ccang (k)==cc  AND  ddang (k)==dd) THEN
                iang = k
              END IF
            END DO
            IF (iang==0) CALL error (idnode, 39, i)
            anginp1 (i, j) = INT (getint (record1, 2), KIND=si)
            anginp2 (i, j) = INT (getint (record1, 3), KIND=si)
            anginp3 (i, j) = INT (getint (record1, 4), KIND=si)
            anginp4 (i, j) = iang

          END DO
          numangcell = numangcell + nangle (i) * nmol (i)

        ELSE IF (word (1 5) =="dihed") THEN

          finmol = INT (getint (record1, 2), KIND=si)
          ndihed (i) = finmol

          DO j = 1, finmol

            READ (nread, '(a200)', IOSTAT = ioerror) record1
            word1 = getword (record1, 1)
            CALL lowercase (word1)
            typ = 0
            IF (word1 (1 3) =="cos") THEN
              typ = 1
            ELSE IF (word1 (1 4) =="harm") THEN
              typ = 2
            ELSE IF (word1 (1 4) =="hcos") THEN
              typ = 3
            ELSE
              CALL error (idnode, 34, 0)
            END IF
            aa = getdble (record1, 6)
            bb = getdble (record1, 7)
            cc = getdble (record1, 8)
            dd = getdble (record1, 9)
            idhd = 0
            DO k = 1, ndhddef
              IF (dhdtype (k)==typ  AND  aadhd (k)==aa  AND  bbdhd (k)==bb  AND  &
                 &ccdhd (k)==cc  AND  dddhd (k)==dd) THEN
                idhd = k
              END IF
            END DO
            IF (idhd==0) CALL error (idnode, 40, i)
            dhdinp1 (i, j) = INT (getint (record1, 2), KIND=si)
            dhdinp2 (i, j) = INT (getint (record1, 3), KIND=si)
            dhdinp3 (i, j) = INT (getint (record1, 4), KIND=si)
            dhdinp4 (i, j) = INT (getint (record1, 5), KIND=si)
            dhdinp5 (i, j) = idhd

          END DO
          numdhdcell = numdhdcell + ndihed (i) * nmol (i)

        ELSE IF (word (1 2) =="no") THEN

          word1 = getword (record1, 2)
          CALL lowercase (word1)
          IF (word1 (1 3) =="iso") moliso (i) =  false

        ELSE IF (word (1 6) =="finish") THEN

          EXIT

        END IF

      END DO

      nsystcell = nsystcell + nbdmol (i) * nmol (i)
      nummolcell = nummolcell + nmol (i)

    END DO

  ELSE IF (key (1 8) =="interact") THEN

    finmol = INT (getint (record, 2), KIND=si)
```

```
DO i = 1, finmol

  READ (nread, '(a200)', IOSTAT = ioerror) record1

  word1 = getword (record1, 1)
  spenam = word1 (1 8)
  ispe = 0
  DO j = 1, nspe
    IF (spenam==namspe (j)) ispe = j
  END DO
  IF (ispe==0) ispe = INT (getint (record1, 1), KIND=si)
  safe = (ispe>0 AND ispe<=nspe)
  IF ( NOT safe) CALL error (idnode, 41, i)
  word1 = getword (record1, 2)
  spenam = word1 (1 8)
  jspe = 0
  DO j = 1, nspe
    IF (spenam==namspe (j)) jspe = j
  END DO
  IF (jspe==0) jspe = INT (getint (record1, 2), KIND=si)
  safe = (jspe>0 AND jspe<=nspe)
  IF ( NOT safe) CALL error (idnode, 41, i)

  IF (ispe>jspe) THEN
    k = (ispe * (ispe - 1)) / 2 + jspe
  ELSE
    k = (jspe * (jspe - 1)) / 2 + ispe
  END IF

  aa = getdble (record1, 4)
  bb = getdble (record1, 5)
  cc = getdble (record1, 6)
  dd = getdble (record1, 7)
  ee = getdble (record1, 8)
  ff = getdble (record1, 9)
  gg = getdble (record1, 10)

  word1 = getword (record1, 3)
  CALL lowercase (word1)
  IF (word1 (1 2) =="lj") THEN

    ktype (k) = 0
    aa = eunit * ABS(aa)
    bb = ABS(bb)
    vvv (1, k) = 4 0 dp * aa
    vvv (2, k) = bb
    vvv (3, k) = bb * bb
    el2 = aa * (bb**12 / (9 0 dp * rcut**9) - bb**6 / (3 0 dp * rcut**3))
    pl2 = aa * (12 0 dp * bb**12 / (9 0 dp * rcut**9) - 2 0 dp * bb**6 / rcut**3)
    fac = 2 0 dp * pi * REAL (nfold, KIND=dp) * REAL (nfold, KIND=dp) * REAL (nspec (ispe) + &
     & nspecmol (ispe), KIND=dp) * REAL (nspec (jspe) + nspecmol (jspe), KIND=dp)
    IF (ispe==jspe) fac = 0 5 dp * fac
    clr (1) = clr (1) + fac * el2
    clr (2) = clr (2) - fac * pl2
    gamma (k) = ABS(cc)
    interact (k, 1) = true
    IF (bb>0 0 dp) interact (k, 2) = true
    interact (k, 3) = true

  ELSE IF (word1 (1 3) =="wca") THEN

    ktype (k) = 1
    aa = eunit * ABS(aa)
    bb = ABS(bb)
    dd = bb * 2 0 dp ** (1 0 dp / 6 0 dp)
    vvv (1, k) = 4 0 dp * aa
    vvv (2, k) = bb
    vvv (3, k) = bb * bb
    vvv (4, k) = dd * dd
    vvv (5, k) = aa
    gamma (k) = ABS(cc)
    interact (k, 1) = true
    IF (bb>0 0 dp) interact (k, 2) = true
    interact (k, 3) = true

  ELSE IF (word1 (1 3) =="dpd") THEN

    ktype (k) = 2
    vvv (1, k) = eunit * ABS(aa)
```

```
        vvv (2, k) = ABS(bb)
        vvv (3, k) = bb * bb
        gamma (k) = ABS(cc)
        interact (k, 1) =  true
        IF (ABS(bb)>0 0 dp) interact (k, 2) =  true
        interact (k, 3) =  true

      ELSE IF (word1 (1 4) =="mdpd") THEN

        ktype (k) = 3
        vvv (1, k) = eunit * aa
        vvv (2, k) = eunit * bb
        vvv (3, k) = eunit * cc
        vvv (4, k) = eunit * dd
        vvv (5, k) = eunit * ee
        vvv (6, k) = ABS(ff)
        vvv (7, k) = ff * ff
        gamma (k) = ABS(gg)
        interact (k, 1) =  true
        IF (ABS(ff)>0 0 dp) interact (k, 2) =  true
        interact (k, 3) =  true
        lmb =  true

                    ELSE IF (word1 (1 4) =="hinz") THEN

        ktype (k) = 4
        vvv (1, k) = eunit * ABS(aa)
        vvv (2, k) = ABS(bb)
        vvv (3, k) = bb * bb
        vvv (4, k)= ABS(dd)
        vvv (5, k)= ABS(ee)
        gamma (k) = ABS(cc)
        interact (k, 1) =  true
        IF (ABS(bb)>0 0 dp) interact (k, 2) =  true
        interact (k, 3) =  true

                    ELSE IF (word1 (1 4) =="norf") THEN

        ktype (k) = 5
        vvv (1, k) = eunit * ABS(aa)
        vvv (2, k) = ABS(bb)
        vvv (3, k) = bb * bb
        gamma (k) = ABS(cc)
        interact (k, 1) =  true
        IF (ABS(bb)>0 0 dp) interact (k, 2) =  true
        interact (k, 3) =  true

      END IF

    END DO

  ELSE IF (key (1 4) =="surf") THEN

    SELECT CASE (srftype)
    CASE (1)
      DO i = 1, nspe
        READ (nread, '(a200)', IOSTAT = ioerror) record1
        word1 = getword (record1, 1)
        spenam = word1 (1 8)
        ispe = 0
        DO j = 1, nspe
          IF (spenam==namspe (j)) ispe = j
        END DO
        IF (ispe==0) ispe = INT (getint (record1, 1), KIND=si)
        safe = (ispe>0  AND  ispe<=nspe)
        IF ( NOT  safe) CALL error (idnode, 42, i)
        aasrf (ispe) = getdble (record1, 2)
      END DO
    CASE (2)
      READ (nread, '(a200)', IOSTAT = ioerror) record1
      word1 = getword (record1, 1)
      spenam = word1 (1 8)
      frzwspe = 0
      DO j = 1, nspe
        IF (spenam==namspe (j)) frzwspe = j
      END DO
      IF (frzwspe==0) frzwspe = INT (getint (record1, 1), KIND=si)
      safe = (frzwspe>0  AND  frzwspe<=nspe)
      IF ( NOT  safe) CALL error (idnode, 43, 0)
      frzwdens = getdble (record1, 2)
```

```
        frzwid = getdble (record1, 3)
        IF (srfx>0) frzwxwid = frzwid
        IF (srfy>0) frzwywid = frzwid
        IF (srfz>0) frzwzwid = frzwid
      END SELECT

    ELSE IF (key (1 6) =="extern") THEN

      READ (nread, '(a200)', IOSTAT = ioerror) record1

      word1 = getword (record1, 1)
      CALL lowercase (word1)
      IF (word1 (1 4) =="grav") THEN
        bdfrcx = getdble (record1, 2)
        bdfrcy = getdble (record1, 3)
        bdfrcz = getdble (record1, 4)
        IF (ABS(bdfrcx)>1 0e-16 dp  OR  ABS(bdfrcy)>1 0e-16 dp  OR  ABS(bdfrcz)>1 0e-16 dp) ldyn =  true
      ELSE IF (word1 (1 4) =="shea") THEN
        shrvx = getdble (record1, 2)
        shrvy = getdble (record1, 3)
        shrvz = getdble (record1, 4)
        IF (srftype==3) THEN
          IF (srfx>0) shrvx = 0 0 dp
          IF (srfy>0) shrvy = 0 0 dp
          IF (srfz>0) shrvz = 0 0 dp
        END IF
        IF (ABS(shrvx)>1 0e-16 dp  OR  ABS(shrvy)>1 0e-16 dp  OR  ABS(shrvz)>1 0e-16 dp) ldyn =  true
      ELSE IF (word1 (1 4) =="elec") THEN
        elecx = getdble (record1, 2)
        elecy = getdble (record1, 3)
        elecz = getdble (record1, 4)
      END IF

    END IF

  END DO

  determine numbers of particles, bonds etc  from unit cell values

  nsyst = nsystcell * nfold
  nusyst = nusystcell * nfold
  nfsyst = nfsystcell * nfold
  numbond = numbondcell * nfold
  numang = numangcell * nfold
  numdhd = numdhdcell * nfold
  nspec = nspec * nfold
  nspecmol = nspecmol * nfold
  nummol = nummolcell * nfold

  checks for interaction data and using mixing rules to assume missing values

  IF (ANY (interact  EQV   false )) THEN

    IF (lmb) THEN

      CALL error (idnode, 44, 0)

    ELSE

      DO i = 1, nspe-1

        DO j = i+1, nspe

          ispe = (i * (i + 1)) / 2
          IF (( NOT  interact (ispe, 1))  AND  ( NOT  interact (ispe, 2))  AND  &
          & ( NOT  interact (ispe, 3))) THEN
            CALL error (idnode, 45, i)
          END IF
          jspe = (j * (j + 1)) / 2
          IF (( NOT  interact (jspe, 1))  AND  ( NOT  interact (jspe, 2))  AND  &
          & ( NOT  interact (jspe, 3))) THEN
            CALL error (idnode, 45, j)
          END IF

          k = (j * (j - 1)) / 2 + i

          IF (( NOT  interact (k, 1))  OR  ( NOT  interact (k, 2))  OR  ( NOT  interact (k, 3))) THEN

            typ = MAX (ktype (ispe), ktype (jspe))
            ktype (k) = typ
```

```
            SELECT CASE (typ)
            CASE (0)
              IF (interact (k, 1)) THEN
                aa = vvv (1, k)
              ELSE
                aa = SQRT (vvv (1, ispe) * vvv (1, jspe))
                vvv (1, k) = aa
              END IF
              IF (interact (k, 2)) THEN
                bb = vvv (2, k)
              ELSE
                bb = 0 5 dp * (vvv (2, ispe) + vvv (2, jspe))
                vvv (2, k) = bb
                vvv (3, k) = bb * bb
              END IF
              el2 = 0 25 dp * aa * (bb**12 / (9 0 dp * rcut**9) - bb**6 / (3 0 dp * rcut**3))
              pl2 = 0 25 dp * aa * (12 0 dp * bb**12 / (9 0 dp * rcut**9) - 2 0 dp * bb**6 / rcut**3)
              fac = 2 0 dp * pi * &
                   &REAL ((nspec (ispe) + nspecmol (ispe)) * (nspec (jspe) + nspecmol (jspe)), KIND=dp)
              IF (( NOT interact (k, 1)) OR ( NOT interact (k, 2))) THEN
                clr (1) = clr (1) + fac * el2
                clr (2) = clr (2) - fac * pl2
              END IF
            CASE (1)
              IF (interact (k, 1)) THEN
                aa = vvv (1, k)
              ELSE
                aa = SQRT (vvv (1, ispe) * vvv (1, jspe))
                vvv (1, k) = aa
                vvv (5, k) = 0 25 dp * aa
              END IF
              IF (interact (k, 2)) THEN
                bb = vvv (2, k)
              ELSE
                bb = 0 5 dp * (vvv (2, ispe) + vvv (2, jspe))
                dd = bb * 2 0 dp ** (1 0 dp / 6 0 dp)
                vvv (2, k) = bb
                vvv (3, k) = bb * bb
                vvv (4, k) = dd * dd
              END IF
            CASE (2)
              IF (interact (k, 1)) THEN
                aa = vvv (1, k)
              ELSE
                aa = SQRT (vvv (1, ispe) * vvv (1, jspe))
                vvv (1, k) = aa
              END IF
              IF (interact (k, 2)) THEN
                bb = vvv (2, k)
              ELSE
                bb = 0 5 dp * (vvv (2, ispe) + vvv (2, jspe))
                vvv (2, k) = bb
                vvv (3, k) = bb * bb
              END IF
            END SELECT

            IF ( NOT (interact (k, 3))) gamma (k) = SQRT (gamma (ispe) * gamma (jspe))
            interact (k, 1) = true
            interact (k, 2) = true
            interact (k, 3) = true

          END IF

        END DO

      END DO

    END IF

  END IF

  DEALLOCATE (interact, STAT=fail(1))
  IF (fail(1)/=0) CALL error (idnode, 1005, 0)

  rescale bond energy parameters

  DO k = 1, nbonddef
    IF (bdtype (k)==1 OR bdtype (k)==2 OR bdtype (k)==4) THEN
      aabond (k) = eunit * aabond (k)
```

```
      END IF
    END DO

    DO k = 1, nangdef
      aaang (k) = eunit * aaang (k)
    END DO

    DO k = 1, ndhddef
      aadhd (k) = eunit * aadhd (k)
    END DO

    rescale electrostatic permittivity parameter

    gammaelec = gammaelec * eunit
    bjerelec = bjerelec * eunit

    RETURN
    END SUBROUTINE read field


END MODULE
```

Apêndice D

Arquivo entrada DL MESO

Exemplo de arquivo de entrada DL_MESO.

```
DL MESO example

volume 100 0 100 0 0 5
temperature 1 0
cutoff 1 0
surface cutoff 1 0
timestep 0 003
steps 2000000
equilibration steps 0
scale temperature every 100
trajectory 1990000 10 1
stats every 10
stack size 100
print every 100
job time 28800 0
close time 100 0
ensemble nvt dpdvv

finish


null

SPECIES 2
SPEC1     1 0 0 0 12500 0
SPEC2     1 0 0 0 12500 0

INTERACTIONS 3
SPEC1     SPEC1     dpd  25 0 1 0 4 5
SPEC1     SPEC2     norf 25 0 1 0 4 5
SPEC2     SPEC2     hinz 25 0 1 0 4 5 4 995 2 25

close
```

Apêndice E

Código de pós-processamento da saída do DL MESO

Basta compilar e usar o executável.

```
PROGRAM extract_vel

 ***********************************************************************

      program for post-processing , extract velocities fields and save in file velocities.dat

 ***********************************************************************

      IMPLICIT none

      INTEGER, PARAMETER     dp = SELECTED_REAL_KIND (15, 307)
      INTEGER, PARAMETER     nuser=5,nlocal=20,nout=7

      CHARACTER(8), ALLOCATABLE     namspe ( ), nammol ( )
      CHARACTER(6)     timechan, nodechan
      CHARACTER(80)     text

      INTEGER     numnodes, nsvcube, isbsv, nspe, nmoldef, nsyst, nusyst, ioerror, frz, &
      & keytrj, srfx, srfy, srfz
      INTEGER     i, ibeads, ibonds, ichain, j, k, l, nsvx, nsvy, nsvz, global, species, &
      & molecule, z1, z2
      INTEGER, ALLOCATABLE     nbeads ( ), nbonds ( )
      INTEGER, ALLOCATABLE     beadnum ( ), beadspe ( , ), beadmol ( , ), ltp ( ), ltm ( ), lfrzn ( )

      REAL(KIND=dp)     aaa, bbb, ccc, x, y, z, vx, vy, vz, fx, fy, fz, ax, ay, az, sax, say, &
      & saz, beads, time
      REAL(KIND=dp)     mbeads, mglobal, delx, dely, delz, volm, dimx, dimy, dimz, shrdx, &
       & shrdy, shrdz

      REAL(KIND=dp), ALLOCATABLE     amass ( ), density ( , ), velx ( ), vely ( ), velz ( ), temperature ( )
      REAL(KIND=dp), ALLOCATABLE     tempx ( ), tempy ( ), tempz ( )
      REAL(KIND=dp), ALLOCATABLE     vxx ( ), vxy ( ), vxz ( ), vyy ( ), vyz ( ), vzz ( )
      REAL(KIND=dp), ALLOCATABLE     fxx ( ), fxy ( ), fxz ( ), fyx ( ), fyy ( ), fyz ( ), &
       & fzx ( ), fzy ( ), fzz ( )
      LOGICAL     eof, lexist

      get number of nodes

      CALL get_command_argument (1, text)
      numnodes = parseint (text)

      IF (numnodes==0) THEN
        PRINT *,"Number of nodes used in calculations  "
        READ (nuser,*) numnodes
      END IF

      ALLOCATE (nbeads (numnodes), nbonds (numnodes))

      determine if HISTORY files exist

      IF (numnodes>1) THEN
```

```
  INQUIRE (file = 'HISTORY000000', EXIST = lexist)
ELSE
  INQUIRE (file = 'HISTORY', EXIST = lexist)
END IF

IF ( NOT  lexist) THEN
  PRINT *, "ERROR  cannot find HISTORY files"
  STOP
END IF

scan HISTORY file for process 0 to determine species/molecule names and properties

IF (numnodes>1) THEN
  OPEN (nlocal, file = 'HISTORY000000', access = 'sequential', &
  & form = 'unformatted',status = 'unknown')
ELSE
  OPEN (nlocal, file = 'HISTORY', access = 'sequential', &
  & form = 'unformatted', status = 'unknown')
END IF

READ (nlocal) nspe, nmoldef, nusyst, nsyst, ibeads, ibonds
READ (nlocal) dimx, dimy, dimz, volm
READ (nlocal) keytrj, srfx, srfy, srfz

ALLOCATE (namspe (nspe), nammol (0 nmoldef))
nammol (0) = 'none'
DO i = 1, nspe
  READ (nlocal) namspe (i), aaa, bbb, frz
END DO

IF (nmoldef>0) THEN
  DO i = 1, nmoldef
    READ (nlocal) nammol (i)
  END DO
END IF

READ (nlocal) text

DO i = 1, ibeads
  READ (nlocal) global, species, molecule, ichain
END DO

IF (ibonds>0) THEN
  DO i = 1, ibonds
    READ (nlocal) molecule, ichain
  END DO
END IF

CLOSE (nlocal)

DO j = 1, numnodes

  WRITE (nodechan, '(i6 6)') j-1
  IF (numnodes>1) THEN
    OPEN (nlocal+j-1, file = 'HISTORY'//nodechan, access = 'sequential', &
    & form = 'unformatted', status = 'unknown')
  ELSE
    OPEN (nlocal+j-1, file = 'HISTORY', access = 'sequential', &
    & form = 'unformatted', status = 'unknown')
  END IF

  READ (nlocal+j-1) nspe, nmoldef, nusyst, nsyst, nbeads (j), nbonds (j)
  READ (nlocal+j-1) dimx, dimy, dimz, volm
  READ (nlocal+j-1) keytrj, srfx, srfy, srfz

END DO

ALLOCATE (amass (nspe), lfrzn (nspe))

DO j = 1, numnodes

  DO i = 1, nspe
    READ (nlocal+j-1) namspe (i), amass (i), bbb, lfrzn (i)
  END DO

  IF (nmoldef>0) THEN
    DO i = 1, nmoldef
      READ (nlocal+j-1) nammol (i)
    END DO
  END IF
```

```
  READ (nlocal+j-1) text

END DO

ALLOCATE (ltp (nsyst), ltm (nsyst))

read in species and molecule data for each bead

DO j = 1, numnodes

  ibeads = nbeads (j)

  DO i = 1, ibeads

    READ (nlocal+j-1) global, species, molecule, ichain
    ltp (global) = species
    ltm (global) = molecule

  END DO

  IF (nbonds (j)>0) THEN

    ibonds = nbonds (j)

    DO i = 1, ibonds
      READ (nlocal+j-1) z1, z2
    END DO

  END IF

END DO

DEALLOCATE (nbeads, nbonds)

read in data for each timestep

OPEN (nout, file = 'velocities dat')

i = 0
eof =  false

DO WHILE ( true )

  READ (nlocal, IOSTAT=ioerror) time, beads, dimx, dimy, dimz, shrdx, shrdy, shrdz

  IF (ioerror/=0) THEN
    eof =  true
    IF (i==0) THEN
      PRINT *, 'ERROR  cannot find trajectory data in HISTORY files'
      STOP
    END IF
    EXIT
  END IF

  i = i + 1


  DO j = 1, numnodes

    IF (j>1) THEN
      READ (nlocal+j-1, IOSTAT=ioerror) time, beads, dimx, dimy, dimz, shrdx, shrdy, shrdz
      IF (ioerror/=0) THEN
        PRINT *, 'ERROR  End of file reached prematurely - ', i-1, ' timesteps written to VTK files'
        eof =  true
        EXIT
      END IF
    END IF

    IF (eof) EXIT

    ibeads = NINT (beads)

    SELECT CASE (keytrj)

    CASE (0)
      PRINT *, "Timestep ",i," keytrj ",keytrj
      DO k = 1, ibeads
        READ (nlocal+j-1) mglobal, x, y, z
        global = NINT (mglobal)
```

```
            WRITE (nout, "(I4,2X,F10 6,2X,F10 6,2X,F10 6)") global, x, y, z
          END DO
        CASE (1)
          PRINT *, "Timestep ",i," keytrj ",keytrj
          DO k = 1, ibeads
            READ (nlocal+j-1) mglobal, x, y, z, vx, vy, vz
            global = NINT (mglobal)
            WRITE (nout, "(I4,2X,F10 6,2X,F10 6,2X,F10 6,2X,F10 6,2X,F10 6,2X,F10 6)") &
            & global, x, y, z, vx, vy, vz
          END DO

        CASE (2)
        PRINT *, "Timestep ",i," keytrj ",keytrj
          DO k = 1, ibeads
            READ (nlocal+j-1) mglobal, x, y, z, vx, vy, vz, fx, fy, fz
              global = NINT (mglobal)
            WRITE (nout, "(I4,2X,F10 6,2X,F10 6,2X,F10 6,2X,F10 6,2X,F10 6,2X,F10 6)") &
            & global, x, y, z, vx, vy, vz
          END DO

        END SELECT

      END DO
    END DO

    write output file with time-averaged values



    CLOSE(nout)


CONTAINS

    INTEGER FUNCTION parseint (word)

 ***********************************************************************

    copyright - daresbury laboratory
    author    - w smith 2001

 ***********************************************************************

    IMPLICIT none

    CHARACTER(1)     u
    CHARACTER(16)    word
    INTEGER     i, m, k, s

    k = 0
    s = 1
    m = LEN (word)

    DO i = 1, m

      u = word (i i)

      IF (u=="-") THEN

        s = -1

      ELSE IF (u=="0") THEN

        k = 10 * k

      ELSE IF (u=="1") THEN

        k = 10 * k + 1

      ELSE IF (u=="2") THEN

        k = 10 * k + 2

      ELSE IF (u=="3") THEN

        k = 10 * k + 3

      ELSE IF (u=="4") THEN
```

```
            k = 10 * k + 4
          ELSE IF (u=="5") THEN
            k = 10 * k + 5
          ELSE IF (u=="6") THEN
            k = 10 * k + 6
          ELSE IF (u=="7") THEN
            k = 10 * k + 7
          ELSE IF (u=="8") THEN
            k = 10 * k + 8
          ELSE IF (u=="9") THEN
            k = 10 * k + 9
          END IF
        END DO
        parseint = s * k
        RETURN
        END FUNCTION parseint
END PROGRAM extract_vel
```

Apêndice F

Script de pós-processamento da saída do LAMMPS

Função que extrai posições e velocidades da saída LAMMPS. Em linguagem Julia.

```
# Read last timestep of LAMMPS dump file
# Input
#       Dump file name with path and number of fields in atoms description
#
# Output
#  atom data struct  id type x y z vx vy vz
#

function readvelocitydata(nome    String, ncol  Int)

        try
            dump = open(nome)
        catch
            error("Dumpfile not found ");
        end
        auxaxxa=0;
        p=0 0; #stores where begin the last step

        # Getting only the last step
        # First get the position of the beginning of the last step in the dumpfile
        while  eof(dump)
                temp = readline(dump);# read a single line
                if length(temp) == 14
                        if temp == "ITEM  TIMESTEP"
                                p= position(dump);# starting position of line next to the header
                                auxaxxa=auxaxxa+1;
                        end

                end
        end


        close(dump);

        dump = open(nome);

        p=convert(Int,p);
        seek(dump,p);
        timestep = parse(Float64,readline(dump));
        lixo=readline(dump);
        Natoms = parse(Int,readline(dump));
        atom data = zeros(Natoms,ncol);#%Allocate memory for atom data
        x bound= -1*ones(1,2);
        y bound= -1*ones(1,2);
        z bound= -1*ones(1,2);

        print(timestep)
        while  eof(dump)
                id = readline(dump)
```

```
            if id == "ITEM  BOX BOUNDS pp pp pp"
                    x bound[1,] = readdlm(IOBuffer(readline(dump)));
                    y bound[1,] = readdlm(IOBuffer(readline(dump)));
                    z bound[1,] = readdlm(IOBuffer(readline(dump)));
            elseif id == "ITEM  ATOMS id type x y z vx vy vz "
                    for j = 1   1  Natoms
                            auxxx=readdlm(IOBuffer(readline(dump)));
                            atom data[j,] =auxxx[1 8];
                    end
            end
        end

        close(dump);

        return atom data

end
```

Apêndice G

Script do cálculo da velocidade *coarse-grained*

Script que calcula a velocidade *coarse-grained*. Em linguagem Julia.

```
include("readdump_last.jl")

function fgauss(x,y,sigma)
        return exp((-x*x-y*y)/2/sigma/sigma)/2/pi/sigma/sigma
end

atom_data=readvelocitydata("nome_entrada.dat",8)

#get position and velocities of atom

velxy=atom_data[:,6:7];

posxy=atom_data[:,3:4];

#creates new vectors equally spaced
aux=collect(0:2:50)
newposxy=zeros(length(aux)*length(aux),2)
for it1 = 1:length(aux)
        for it2 = 1:length(aux)
                newposxy[(it1-1)*length(aux)+it2,:]=[aux[it1], aux[it2]]
        end
end

#new vector with the values of coarse-grained velocities
newvelxy=zeros(size(newposxy))


for it1 = 1:size(newposxy,1)
        aux=0;
        aux1=0;
        for it2=1:Natoms
                aux=fgauss(newposxy[it1,1] - posxy[it2,1], newposxy[it1,2] - posxy[it2,2],2.5)
                newvelxy[it1,1]=newvelxy[it1,1] + velxy[it2,1]*aux;
                newvelxy[it1,2]=newvelxy[it1,2] + velxy[it2,2]*aux;
                if abs(aux)>1e-5
                        aux1=aux1+1

                end
        end
        newvelxy[it1,1]=newvelxy[it1,1]/aux1;
        newvelxy[it1,2]=newvelxy[it1,2]/aux1;
end

newposvelxy=[newposxy newvelxy]

#save new positions and velocities
writedlm("newposxy.dat",newposxy)
writedlm("newvelxy.dat",newvelxy)
writedlm("newposvelxy.dat",newposvelxy)
```

Apêndice H

Lista de simulações realizadas

A abaixo encontra-se tabelas listandos os testes, com os principais parâmetros utilizados nas simulações que não obtiveram um perfil linear ou em que ocorreram erros durante a simulação.

ϕ, V_s e Pe são como descritos no Capítulo 5. L o tamanho de um dos lados de uma caixa de simulação com dimensões L x L x L. ρ representa a densidade numérica de partículas.

ϕ	V_s	Pe	L	ρ
0.5	0.8	1.11	10	3
0.5	1	1.11	10	3
0.5	3	1.11	10	3
0.5	5	1.11	10	3
0.5	0.5	1.11	10	4
0.5	0.8	1.11	10	4
0.5	1	1.11	10	4
0.5	3	1.11	10	4
0.5	5	1.11	10	4

Tabela H.1: Testes variando V_s e ρ

ϕ	V_s	Pe	L	ρ
0.1	0.5	0.8	10	3
0.1	0.5	1	10	3
0.1	0.5	1.25	10	3
0.1	0.5	1.5	10	3
0.1	0.5	1.75	10	3
0.1	0.5	2	10	3
0.1	0.5	2.2	10	3
0.1	0.5	2.4	10	3

Tabela H.2: Testes variando Pe

ϕ	V_s	Pe	L	ρ
0.9	1	1.11	10	4
0.3	1	1.11	10	3
0.05	0.5	1.11	10	3
0.2	0.5	1.11	10	3
0.3	0.5	1.11	10	3
0.05	0.8	1.11	10	3
0.1	0.8	1.11	10	3
0.2	0.8	1.11	10	3
0.3	1	1.11	10	3
0.3	0.5	1.11	10	3
0.5	1	1.11	10	3
0.7	1	1.11	10	3
0.9	1	1.11	10	3
0.5	1	1.11	10	4
0.7	1	1.11	10	4

Tabela H.3: Testes variando ϕ

ϕ	V_s	Pe	L	ρ
0.1	0.2	0.5	10	3
0.1	0.2	0.5	10	3
0.3	0.2	0.5	10	3
0.5	0.5	0.5	10	3
0.3	0.5	0.5	10	3
0.5	0.5	0.5	10	3
0.1	0.8	0.5	10	3
0.3	0.8	0.5	10	3
0.1	1.2	0.5	10	3
0.3	1.2	0.5	10	3
0.5	1.2	0.5	10	3
0.3	1	0.5	10	3
0.1	0.2	1.5	10	3
0.3	0.2	1.5	10	3
0.5	0.2	1.5	10	3
0.1	0.5	1.5	10	3
0.5	0.5	1.5	10	3
0.3	0.8	1.5	10	3
0.5	0.8	1.5	10	3
0.1	1.2	1.5	10	3
0.3	1.2	1.5	10	3
0.5	1.2	1.5	10	3
0.1	1	1.5	10	3
0.3	1	1.5	10	3
0.5	1	1.5	10	3
0.1	0.2	2	10	3
0.3	0.2	2	10	3
0.5	0.2	2	10	3
0.1	0.5	2	10	3
0.3	0.5	2	10	3
0.5	0.8	2	10	3
0.1	1.2	2	10	3
0.3	1.2	2	10	3
0.5	1.2	2	10	3
0.5	1	2	10	3
0.1	0.5	0.5	10	3

Tabela H.4: Outros testes realizados