UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
DEPARTAMENTO DE QUÍMICA
PROGRAMA DE PÓS-GRADUAÇÃO EM QUÍMICA

MELQUIADES: UM PROGRAMA DE MONTE CARLO PARA A SIMULAÇÃO DE SISTEMAS MULTICOMPONENTES UTILIZANDO MODELOS DE POTENCIAIS ARBITRÁRIOS

Asdrubal Lozada Blanco†

Tese apresentada como parte dos requisitos para obtenção do título de DOUTOR EM CIÊNCIAS, área de concentração: FÍSICO-QUÍMICA

Orientador: Prof. Dr. Luiz Carlos Gomide Freitas

† Bolsista CAPES

SÃO CARLOS - SP
2017

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Química

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a Defesa de Tese de Doutorado do candidato Asdrubal Lozada Blanco, realizada em 23/02/2017:

Prof. Dr. Luiz Carlos Gomides Freitas
UFSCar

Prof. Dr. João Manuel Marques Cordeiro
UNESP

Prof. Dr. Alejandro Lopez Castillo
UFSCar

Prof. Dr. Erick Lazaro Melo
UFSCar

Prof. Dr. Osmair Vital de Oliveira
IFSP - Catanduva

Ad nihil

Agradecimentos

Ao professor Luiz Carlos Gomide, pela orientação e colaboração dada.

À Comissão Examinadora, pelo tempo dedicado à revisão da tese.

Ao Programa de Pós-Graduação em Química da Universidade Federal de São Carlos.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior(CAPES), pela bolsa de estudos.

Lista de tabelas

Lista de figuras

Lista de Algoritmos

Resumo

MELQUIADES: Um programa de Monte Carlo para a simulação de sistemas multicomponentes usando modelos de potenciais arbitrários. Foi desenvolvido um programa de Monte Carlo via cadeia de Markov que permite utilizar funções matemáticas arbitrárias como funções de potencial para o cálculo de energia de interação em sistemas multicomponentes, **MELQUIADES**.

MELQUIADES está escrito na linguagem FORTRAN 90 com modificações de **Fortran 2003**. Pode ser executado serialmente e compilado com o compilador GNU Fortran em distintas distribuições GNU/Linux e arquiteturas de máquina. Na versão atual este programa implementa: i) uma interface para a análise léxica de palavras e posterior utilização como funções de potencial; ii) além do potencial coulômbico, os potenciais de Lennard-Jones, Buckingham ou Yukawa podem ser empregados para o cálculo de energia de interação; iii) o espaço de configurações é amostrado tanto no ensemble canônico, quanto no isotérmico-isobárico; iv) permite o cálculo de energia em sistemas multicomponentes e a construção de diversas listas de vinzinhos dependendo do tamanho molecular. O código fonte de **MELQUIADES** foi escrito em formato modular e permite a adaptação de ferramentas para a análise de resultados.

Palavras-chave: MELQUIADES, Monte Carlo, Cadeias de Markov, Simulação, Energia potencial

Abstract

MELQUIADES: A Monte Carlo program for simulation of multicomponent systems using arbitrary potential models. It was developed a general purpose Metropolis Monte Carlo program, which allows to use arbitrary mathematical functions as potential functions for interaction energy calculation, in order to simulate multicomponent system, **MELQUIADES**.

MELQUIADES is a serial and stand-alone FORTRAN 90 and 2003 program for simulation of multicomponent systems using the Metropolis Monte Carlo Algorithm. It can be compiled using GNU Fortram Compiler (gfortran) in different GNU/Linux distribution and hardware. In the current version some capabilities implemented are: i) introduction of potential functions as plain text to facilitate the evaluation of new models; ii) besides Coulombic potential, Lennard-Jones, Buckingham or Yukawa potentials to can be used to calculate intermolecular interaction energy; iii) the configuration space is sampled either in canonical or isothermal-isobaric ensembles; iv) allows the multicomponent systems energy calculation and building of diverse neighbor list depending on the molecular size. The source code of **MELQUIADES** was written in modular format and allow the adaptation of analysis tools.
Keywords: MELQUIADES, Monte Carlo, Markov Chains, Simulation, Potential energy

Contents

Introdução

Verifíca-se desde a leitura sobre a quantidade de livros, artigos, publicações e programas relativos à simulação computacional nas áreas de física e química, e em particular à aplicação dos métodos de Monte Carlo, a recente criação, crescimento e consolidação de comunidades científicas interessadas nestes temas.[1] Este crescente interesse coincide, e pode estar relacionado, com o aumento e melhora da capacidade de computo disponivel nestas comunidades.

Específicamente, no contexto da simulação computacional para o cálculo de propriedades termodinâmicas, com fundamento na mecânica estatística, têm sido desenvolvidos varios programas que implementan o Método de Monte Carlo. Dentre os mais usualmente utilizados, em anos recentes, estão: *MCCS Towhee*,[2] *Cassandra*,[3] *DL_MONTE*[4] e *DIADORIM*.[5]

Ao analisar o anterior, um conjunto de questões aparece, dentre estas: quanto difiere um programa de outro? Estão sendo desenvolvidas estas ferramentas para atender necessidades específicas individuais ou de uma comunidade maior? Quão abrangente é a divulgação e disponivilização destes programas? Estas ferramentas servem ao desenvolvimento de estas áreas ou somente replicam questões comuns?

Essa última questão, referida ao aporte na área da físico-química, é a que levou a propor a construção de uma ferramenta versátil, adaptável às necessidades da pesquisa e de fácil uso. Com isto e como resultado desta proposta apresenta-se *MELQUIADES*.

MELQUIADES descreve-se como um programa de Monte Carlo para a simulação de sistemas multicomponentes utilizando modelos de potenciais arbitrários. Sua capacidade para incorporar qualquer função matemática como função de potencial de interação é a característica essencial com a qual este programa procura contribuir a novos desenvolvimentos na área da físico-química. Dada essa caraterística elimina-se a necessidade de codificação ou recodificação de programas, que é um obstaculo no estudo e implementação de novos modelos nesta área. Com isto, o foco pode ser mantido na descrição e construção de modelos de interesse, mas não nos detalhes técnicos de sua implementação.

Neste documento apresentam-se os fundamentos e técnicas empregadas no desenvolvimento de MELQUIADES. No capítulo 2 expõem-se os elementos matemáticos subjacentes aos métodos empregados, no 3 apresenta-se uma descrição das ferramentas de programação com as que o programa foi escrito, e nos capítulos 4 e 5 mostra-se a construção de MELQUIADES e teste de validação do mesmo, respectivamante. Por fim, no capítulo 6 apresenta-se um conjunto de conclusões.

Fundamentação teórica

O programa de simulação computacional apresentado neste documento caracteriza-se por dois aspectos básicos. Por sua aplicação em simulação mediante o método de Monte Carlo. E por sua utilidade como ferramenta na avaliação de modelos de potencial de interação. Para o emprego apropriado deste último é necessaria a comprenssão das teorias subjacentes aos métodos implementados.

Dentre as teorias matemáticas que fundamentam o programa estão as dos grafos, dos grupos e das probabilidades. Caracterizam-se por suas aplicações em áreas da física, como a mecánica estatística e suas derivações, e empregam-se, usualmente, para o estudo de diversos tópicos através de técnicas computacionais.

Pelo acima exposto e o contexto no que este trabalho é desenvolvido, os temas e discusões nesta e nas próximas seções serão apresentadas com relação às teorias mecionadas, em particular ao seu emprego em mecânica ou termodinâmica estatística. Assim, para esta exposição utiliza-se como referência o modelo ferromagnético de Ising. Isto é, o inicio de cada seção mostra como exemplo algum elemento deste modelo, a forma como este foi desenvolvido ou como é utilizado.

Essa forma de exposição é utilizada devido à simplicidade para a implementação do modelo de Ising e à equivalência matemática entre este e aqueles representados através dos métodos de Monte Carlo via cadeias de Markov. Além disso, emprega-se tal modelo já que na sua simulação encontram-se todos os elementos básicos para o desenvolvimento deste tipo de programas computacionais. Com isto, a argumentação teórica essencial para a descrição do programa de simulação é completamente apresentada.

Nas seções sucessivas, apresentam-se os conceitos de retículo, configuração, potencial, cadeias de Markov e momento estatístico. Apresenta-se também, sucintamente, uma descrição dos métodos de Monte Carlo e como estes são empregados.

Sistemas reticulares

Com uma versão simplificada de arranjo reticular regular (denominado modelo de retículo estático), em 1920, Wilhelm Lenz propôs para o estudo do comportamento ferromagnético o que posteriormente seria denominado modelo de Ising.[†] A primeira solução analítica para este, no caso unidimensional (ver a figura 2.1), foi apresentada em 1925 por Ernst

[†]No modelo de Ising cada nó de um retículo associa uma variavel denominada *spin*(σ) ou *número de ocupação*, e a energia de interação é dada por $E = -J_{nn}\Sigma_{nn}\sigma_i\sigma_j - H\Sigma_i\sigma_i$, onde J_{nn} e H são a constante de interação e a intensidade do campo magnético externo, respectivamente.[6,7]

Ising.[8] Nos anos posteriores diversas propostas matemáticas para uma solução analítica geral, que resultaram insuficientes, foram apresentadas. E em 1944, Lars Onsager, supondo nula a intensidade do campo magnético, resolveu o modelo para um retículo bidimensional.[9] Desde então, extensões a múltiplas dimensões tem sido apresentadas, porém sem resolução analítica.

FIGURA 2.1 – Retículo regular unidimensional sugerido por Wilhelm Lenz para o modelo ferromagnético. Neste os sinais $+$ ou $-$ representam as variáveis de *spin*(σ) associadas, e sua sequência descreve uma configuração do sistema.

Caracterizado como uma versão muito simplificada e pouco realista, o tipo de arranjo reticular utilizado no modelo de Ising têm sido empregado, também, em estudos sobre gases, polímeros, micelas, interfaces, entre outros. Segundo DOVE,[10] nos estudos de materia condensada, os retículos estáticos (que utilizam somente as posições médias dos átomos) são apropriados para explicar propriedades como, dureza, forma dos cristais, estrutura eletrônica, características ópticas, etc; mas são insuficientes para a descrição de propriedades térmicas e de transporte, transições de fase, fenômenos dielétricos, etc. E propõe a ideia de um retículo dinâmico nestes casos.[10]

Formalmente, de acordo com a teoria dos grupos, no espaço real n-dimensional ($\mathbb{R}^n$) um retículo (Λ) é um subgrupo discreto (que é isomórfico ao conjunto dos números inteiros $\mathbb{Z}^n$) que satisfaz as seguintes propriedades: i) Λ é fechado com relação à adição e subtração, e ii) existe um $\epsilon > 0$ tal que quaisquer dois pontos distintos do retículo, $x \neq y \in \Lambda$, estão à distancia $||x - y|| \geq \epsilon$.[11] A figura 2.2 mostra uma representação de retículo empregada em geometria euclidiana. Nesta, o arranjo é descrito como uma rede composta por pontos organizados no plano.

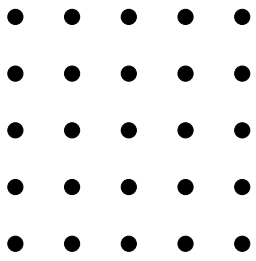


FIGURA 2.2 – Represtação de un retículo regular no espaço euclidiano

Por outro lado, segundo a teoria dos grafos, um retículo é definido como o produto cartesiano $K_m \times K_n$ de grafos* K_i no espaço euclidiano $\mathbb{R}^n$, e é denominado grafo reticular. Nesta definição cada par de vértices do grafo está unido por uma aresta.[12]

*Um grafo define-se como um par ordenado $G(V, A)$, onde V representa os elementos denominados nós ou vértices, e A às arestas.

Mencionam-se como exemplos de uso dos arranjos reticulares a formulação (denominada *Formalismo termodinâmico*) de um conjunto de teoremas no contexto formal da mecânica estatística de equilibrio, por Ruelle.[13] E o cálculo dos parâmetros críticos do modelo de Ising, T_c e K_c, por Oliveira e Tomé.[6] No primeiro, o conjunto $\mathbb{R}^n$ é reemplazado pelo conjunto $\mathbb{Z}^n(L)$, (o que constitui, segundo o autor, um retículo cristalino), e apresentam-se demonstrações para caracterizar os denominados *estados de Gibbs*. No segundo, os autores empregam representações de rede no espaço euclidiano de 2, 3, e 4 dimensões, e concluem que nos casos bidimensionais (2) os resultados são exatos, enquanto que as redes cúbica (3) e hipercúbica (4) errors são encontrados na última casa decimal.

Um caso análogo ao descrito no segundo exemplo empregado usualmente para o cálculo de propriedades termodinâmicas é apresentado por Allen e Tildesley,[14] no estudo sobre simulação de líquidos. Neste, os autores propõem a construção de retículos cristalinos tridimensionais, como o cúbico de face centrada ou o ortorrômbico, para definir uma configuração inicial do sistema. Embora sejam descritos inicialmente através de um retículo regular, a organização sucessiva das partículas, nestes casos, não é mais regular. Pelo que Landau[7] denomina-os modelos *fora do retículo*.

Configuração

Na proposta contida no modelo de Ising para o comportamento ferromagnético, de acordo com LENZ,[15] citado por BRUSH,[16] átomos dipolares rotam livres ao redor de posições fixas em um retículo, e a cada um deles encontra-se associada uma variavel escalar denominada *spin*, onde um de dois valores é possivel: $+1$ ou -1.

No formalismo termodinâmico, denomina-se *espaço de configuração* ao conjunto formado por esses spins, e define-se-lhe por

$$\Omega = \left\{ \xi \in \prod_{x \in L} \Omega_x : (\forall \Lambda \in \mathrm{F}) \xi | \Lambda \in \bar{\Omega}_\Lambda \right\} \tag{2.1}$$

Neste, ξ é uma configuração, e está descrita como uma coleção dos diferentes estados ξ_x do sistema,[12] F é um conjunto de $L \subset \mathbb{Z}^{d13\dagger}$, e o valor Ω_x é a lista das possiveis orientações de os átomos em x. Denomina-se esta lista *espaço de spin*. Alguns exemplos de *espaço de spin* encontrados na física são: $\Omega_x = \{-1, +1\}$ (modelo de Ising, $\Omega_x = \mathbb{S}^2 \subset \mathbb{R}^3$ (ferromagneto anisotrópico ou modelo tridimensional de Heisenberg[§]), $\Omega_x = \{0, 1\}$ (gás reticular) ou $X = \mathbb{R}^n$.[17]

[†] L é um conjunto infinito numeravel.

[§] Modelo tridimensional de Heisenberg: $H = -J\Sigma(S_{ix}S_{jx} + S_{iy}S_{jy} + S_{iz}Sjz)$

Em mecânica clássica, denomina-se *espaço de configuração* ao espaço de todas as posições possíveis de um sistema. Assim, para uma partícula em $\mathbb{R}^n$ o espaço de configuração é $\mathbb{R}^n$. Por outro lado, denomina-se à dimensão deste espaço, isto é, o menor número de parâmetros que o especificam completamente, *número de graus de liberdade* do sistema.[18] Dois exemplos de espaço de configuração apresentados por ARNOLD,[19] são: i) para o pêndulo planar o círculo, S^1; para um conjunto de n destes pêndulos o espaço correspondente está dado por $S^1 \times S^1 \times \cdots \times S^1 = T^n$ (um toróide n-dimensional), e ii) para um corpo rígido em $\mathbb{R}^3$ com centro de massa em $0 \in \mathbb{R}^3$, o espaço de configuração é o produto cartesiano $SO(3) \times \mathbb{R}^3$; onde $SO(3) = \{R : \mathbb{R}^3 \to \mathbb{R}^3\}$ (matrices de rotação 3×3).

Em simulação estocástica empregam-se, usualmente, para a definição da dimensionalidade do espaço de configuração de sistemas moleculares em $\mathbb{R}^3$, movimentos translacionais e rotacionais, no caso de moléculas rígidas. De acordo com ALLEN e TILDESLEY,[14] os primeiros referem-se aos deslocamentos aleatórios dos centros de massa das moléculas, e os segundos à rotação destas ao redor de eixos internos. E, para moléculas não rígidas, mediante a definição de um sistema de coordenadas generalizadas $(\vec{r}, \vec{p})$ que associa um subconjunto aberto do espaço de configuração com um subconjunto aberto de $\mathbb{R}^n$,[18] assim a dimensionalidade é dada pelo conjunto de distâncias e ângulos que descrevem a configuração interna de cada molécula.

Além do anterior, a variação do número de moléculas ou partículas, como no caso do ensemble grande canônico,[20] também é utilizada na definição do espaço de configuração. Assim, e sem perda de genarilidade, pode-se entender a dimensionalidade deste espaço como o valor gerado pelo conjunto $GL = \{q_i, n : q_i \in \mathbb{R}^3; n \in \mathbb{N}U\{O\}\}$, onde GL são os graus de liberdade do sistema.

Potencial de interação

Na descrição do fenômeno ferromagnético, como foi proposto por Lenz, somente interagem os átomos em sitios adjacentes, isto é, aqueles átomos que ocupam sitios, nós ou vértices que estão conectados por uma aresta no retículo. E, a energia potencial de interação, $\Phi(\vec{x})$, está dada por,

$$[]\Phi(\vec{x}) = -H \sum_{i}^{N} x_i - J \sum{}' x_i x_j \tag{2.2}$$

A segunda soma no lado direito da equação é divergente, e representa interação, somente, para os sitios i e j a distância $d(i,j) = 1$,[17] isto é, se i e j são vizinhos próximos. H e J são a intensidade do campo magnético externo e constante de interação, respectivamente .

Em termos do momento de dipolo, μ_i, com $\mu_i = constante \times x_i$ a equação 2.2

pode ser escrita como,

$$\Phi = -\sum_{i=1}^{N} \vec{\mu_i} \cdot \vec{B} + \sum_{i \neq j} f(r_{ij}) \vec{\mu}_i \vec{\mu}_j \tag{2.3}$$

onde r_{ij} é a separação entre dipolos, o coeficiente f_{ij} é uma constante e $\vec{B}$ o campo magnético aplicado.[21]

Na teoria das probabilidades, um potencial é uma coleção de conjuntos, $\Phi = (\Phi_A)_{A \in \mathrm{L} \subset \mathbb{Z}^d}$, de funções $\Phi_A : \Omega \to \mathbb{R}$, tal que para todo $\Lambda \in L$ e $\xi \in \Omega$, a serie

$$H_\Lambda^\Phi(\xi) = \sum_{A \in L, A \cap \Lambda = \emptyset} \Phi_A(\xi) \tag{2.4}$$

é convergente. H_Λ^Φ é denominado o hamiltoniano em Λ para Φ e $H_\Lambda^\Phi(\xi)$ a energia de ξ em Λ para Φ. E em física estatística, para distribuições de equilibrio em sistemas finitos, a expressão,

$$h_\Lambda^\Phi = \exp[-H_\Lambda^\Phi(\xi)], \tag{2.5}$$

é conhecida como fator de Boltzmann.[22]

De acordo com SINAI,[23] o caso mais simples de potencial relativo ao movimento em um grupo euclidiano é da interação de pares, U. A qual pode ser escrita como,

$$U : [0, \infty) \to \mathbb{R}^1 \cup \{\infty\} \tag{2.6}$$

onde $U : U(r), 0 \leq r \leq \infty$, representa a energia potencial de um par de partículas separadas por uma distância r.[23]

Para a simulação computacional, esses potenciais de interação são escritos, usualmente, utilizando termos que envolvem duplas, triplas ou, em geral, n-tuplas de átomos, íons, moléculas ou partículas. Segundo ALLEN e TILDESLEY (1989),[14] no caso mais simple, em um sistema de N átomos, a energia potencial pode ser dividida dependendo das coordenadas, e está dada por

$$V(\vec{r}) = \sum_i v_1(r_i) + \sum_i \sum_{j>i} v_2(r_{ij}) + ... \tag{2.7}$$

De forma semelhante à definição do potencial Φ no modelo de Ising, v_1 representa a interação com um campo externo, e os termos de ordem superior $(v_2, v_3, ...)$ referem-se à interação das partículas.

Particular interesse tem sido dado à construção de potenciais de interação que dependem somente da magnitude da separação entre duas partículas, isto é, de potenciais de pares. Preferência esta devida à versatilidade na represetação de diversos fenômenos físicos, à

descrição apropriada de suas propriedades, e às dificuldades associadas ao cálculo de potenciais com termos de ordem superior nas simulações. Restringe-se aqui a discussão a três destes potencias, implementados na versão atual do programa de Monte Carlo discutido neste trabalho. Estes são, a saber, os potenciais de pares de Lennard-Jones, Yukawa e Buckingham.

Potencial de Lennard Jones

Na sua discussão[24] sobre como a coesão faz parte dos fenômenos naturais, como condensão ou solificação, em 1931 J. E. Lennard-Jones, baseado na teoria de perturbação, concluiu que uma representação apropriada de potencial no denominado *campo atrativo de van der Waals*, é mediante a forma $-\lambda_{(atr.)}R^{-6}$. Nesta expressão, também apresentada por Slater e Kirkwood[25] nos estudos sobre forças de *van der Waals* originadas pela polarização das moléculas (energia mutua), o coeficiente $-\lambda$ é dado como uma função que depende da polarizabilidade, do número de elétrons, e da energia do átomo no estado normal. Por outro lado, em oposição aos termos de coesão, de acordo com LENNARD-JONES[24] (1931), para os componentes dissociativos (ou campos repulsivos) representações do tipo $\lambda_{(rep.)}R^{-n}$ ou formas contendo termos como $e^{-\alpha R}$ são apropriados.

A partir dos criterios de coesão-dissociação, um modelo que supõe a soma de potenciais atrativos e repulsivos, e que pertence à familia dos potenciais de *Mie*,[26] tem sido apresentado por Lennard-Jones, tornando-se de amplo uso devido a sua simplicidade e versatilidade na descrição de diversos fenômenos. Tal modelo é denominado *potencial de Lennard-Jones* ou *potencial 6-12*, e está dado por

$$\Phi(r_{ij}) = 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right] \tag{2.8}$$

onde os coeficientes ou parâmetros ϵ_{ij} e σ_{ij}, são denominados *profundidade do potencial de Lennard-Jones* e *diâmetro de Lennard-Jones*, respetivamente. Para sua obtenção tem-se tornado convencional o uso das denominadas regras de cruzamento. Segundo DELHOMMELLE e MILLIE (2001) três de tais regras, que têm sido implementadas em estudos de equilíbrio vapor-líquido, líquidos compressíveis e gases raros, e que não requerem parâmetros adicionais, são: regra de Lorentz-Berthelot, de Kong e de Waldman-Hagler.[27] Em particular, as regras de Berthelot e Lorentz, referem-se ao uso das médias geométricas e aritméticas, respectivamente, e estão dadas por

$$\epsilon_{ij} = \sqrt{\epsilon_{ii}\epsilon_{jj}} \tag{2.9}$$

$$\sigma_{ij} = \frac{\sigma_{ii} + \sigma_{jj}}{2} \tag{2.10}$$

Dentre as diversas implementações do potencial 6-12, em sua forma básica ou

com alterações mediante a adição de outros tipos de potencial, podem-se mencionar as seguintes: estudos de fluidos, termodinâmica de cluster, cálculo de energia livre, cálculo de coeficientes do virial, cristalização de misturas, estudos de diagramas de fase e outros. Com relação ao modelo de Ising, um criterio de isomorfismo com fluidos de Lennard-Jones foi proposto por Kulimskii.[28]

Em forma genérica, na representação de sistemas de moléculas, o *potencial de Lennard-Jones* pode ser escrito como,

$$E_{ab} = \sum_{i}^{n_a} \sum_{j}^{n_b} \frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}} \tag{2.11}$$

onde n_a e n_b representam o número de sítios (átomos ou cargas) nas moléculas. Nesta, de acordo com FREITAS e BOTELHO (1994), os parâmetros de interação não diagonal A_{ij} e B_{ij} podem ser obtidos via regra de cruzamento[29] como,

$$A_{ij} = \sqrt{A_{ii} A_{jj}} \tag{2.12}$$

$$B_{ij} = \sqrt{B_{ii} B_{jj}} \tag{2.13}$$

Finalmente, é importante mencionar que através de uma adptação, isto é, em soma conjunta com o potencial coulombico, denominada *potencial 12-6-1* (*potencial de Haegler-Lifson*), o *potencial de Lennard-Jones* tem-se tornado de uso extensivo, e cabe mencionar como exemplo a derivação das denominadas *funções de potencial transferíveis*, para líquidos orgânicos e soluções, apresentadas por Jorgensen.[30]

Potencial de Yukawa

Derivado das ideas de Heisenberg e Fermi sobre interação de partículas elementares, e com a introdução de uma nova interpretação, em 1935, Hideki Yukawa[31] apresentou uma modificação à teoria de desintegração β. Na sua forma inicial, esta empregava a hipótese do "neutrino" (ver equação (2.14))

$$[]N^0 \rightarrow +p^+ + e^- + n \tag{2.14}$$

onde um nêutron, N^0 e um próton, p^+ interagem produzindo um par neutrino, n, e um elétron e^-. De acordo com YUKAWA, nesse processo nem sempre se apresenta a emissão destas partículas, mas a energia liberada é absorvida por outro nêutron, que posteriormente será transformado ao estado próton.

Análogo ao modelo de interação de partículas carregadas em um campo electromagnético, uma equação é construida para o campo entre um nêutron e um próton, e está dada

por,

$$\left[\Delta - \frac{1}{c^2}\frac{\partial^2}{\partial t^2}\right]\Phi = 0 \tag{2.15}$$

onde o potencial da força, $\Phi(r)$, entre estas partículas é escrito como,

$$[]\Phi(r) = \pm g^2 \frac{e^{-\lambda r}}{r} \tag{2.16}$$

na equação 2.16, g é uma constante e representa a magnitude do potencial e λ uma constante com unidades cm^{-1}

Este modelo de potencial, também descrito como um potencial coulombico blindado, tem sido utilizado no estudo de sistemas coloidais e para caracterizar partículas em entornos de plasma.[32] De acordo com OREA e DUDA[26] no estudo de aglomeração de coloides e de estabilidade de fases é de uso comun o potencial de Yukawa, na forma,

$$u(r) = \begin{cases} \infty & : r < \sigma \\ -\epsilon\sigma\frac{e^{-\kappa(r-\sigma)}}{r} & : \sigma \leq r \end{cases}$$

onde σ é o díametro de uma partícula, e κ e ϵ são parâmetros do potencial. No estudo de fluidos de spin de Ising para a obtenção de diagramas de fase, tem sido proposto o uso conjunto dos potenciais de Lennard-Jones e Yukawa. Neste, o potencial 6-12 representa a parte não magnética, enquanto o potencial de Yukawa mostra as interações de *spin*.[33]

Além do anterior, na área de física de materia condensada, tem sido proposta uma equivalência entre os potenciais de Lennard-Jones e de Yukawa. Esta consiste em uma aproximação esférica média, onde o pontencial 6-12 é mimetizado mediante a soma de dois potenciais de Yukawa, o que é denominado *LJ2Y*. A importância desta equivalência, de acordo com KREJČÍ et al. (2011), está na sua capacidade de representar fluidos nos quais a repulsão suave é similar à dos de Lennard-Jones.[34]

Potencial de Buckingham

Proposto por R. A. Buckingham, em 1938, como uma extensão do método de Lennard-Jones para o cálculo de energia de interação, $E(r) = \lambda r^{-s} - \mu r^{-t}$, no *potencial de Buckigham* a descrição do potencial repulsivo é dada por expressões como $be^{-r/\rho}$ e $b_1 r^7 e^{-r/\rho_1}$.[35] De isto se obtém a função de potencial repulsivo-atrativo,

$$\Phi(r) = be^{-r/\rho} - \frac{\mu}{r^6} \tag{2.17}$$

Este potencial tem sido utilizado, geralmente, na simulação de materiais iônicos,[36] e do equilibrio de fase de fluidos.[37] No entanto, de acordo com CHEN e LEE,[38] um efeito denominado *Catástrofe de Buckingham* apresenta-se nas diversas combinações de poten-

ciais com as expressões propostas por Buckingham, devido a que o valor de distâncias interatómicas calculadas é inferior a uma distância crítica, de forma que o regime muda de repulsivo para atrativo, induzindo instabilidade.

Um tema comum na implementação dos potenciais discutidos acima, ou daqueles que resultam da derivação dos mesmos, encontra-se relacionado com a eficiência na obtenção de resultados, isto é, o tempo gasto no cálculo. Tem sido discutido amplamente que entre os fatores que determinam tal eficiência estão, por exemplo, o cálculo ou não de raizes quadradas e a implementação de truncamentos no potencial.[14] Este último, comumente empregado em simulações que estão sujeitas a condições periódicas de contorno, refere-se à definição de valor zero (0) no potencial para distâncias entre partículas maiores a um raio especificado, denominado raio de corte, r_c (Ver figura 2.3).

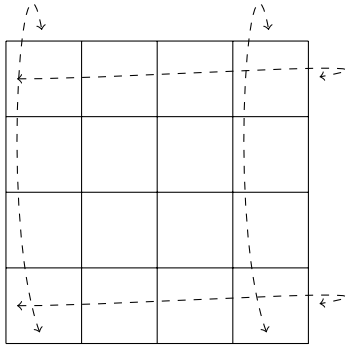


FIGURA 2.3 – Toróide criado pela envoltura dos contornos.

Processos Markovianos

Em 1963, Glauber[39] propôs uma interpretação do modelo de Ising na qual os *spins* das partículas, nos vértices de um retículo, são representados como funções estocásticas do tempo, $\sigma_j(t)$, estão restritos aos valores ± 1, e apresentam transições aleatórias entre eles. Dita interpretação é denominada *modelo estocástico de Ising*. Segundo a teoria das probabilidades, isto significa que os *spins* forman um processo Markoviano, em que o tempo é o argumento das variáveis aleatórias, e a dinâmica do processo no espaço de configurações, $\Omega = \{-1, +1\}^{\mathbb{Z}^d}$, é especificada pelo "giro" de -1 a $+1$, e sua taxa de variação está dada por

$$\exp\left[-\beta \sum_{d(i-j)=1} x_i x_j\right] \tag{2.18}$$

onde $\beta \in \mathbb{R}^+$ representa o inverso da temperatura do sistema.[40]

Da interpretação de Glauber, além dos estudos utilizando campos aleatórios ou sistemas de partículas interagentes do modelo de Ising, tem-se a resolução númerica deste para sistemas com valores de dimensionalidade altos, ou com hamiltonianos com maior grau de complexidade.

De igual forma que para o modelo estocástico de Ising, a base teórica sobre a que se fundamenta a construção do programa de Monte Carlo, apresentado neste documento, é o conceito de *processo Markoviano*. Como casos particulares destes processos, serão apresentados nas próximas seções os tópicos: cadeias de Markov e algoritmo ou método de Metropolis.

Cadeias de Markov

Na teoria de probabilidade denomina-se *processo Markoviano* ou *processo aleatório Markoviano* aos processos estocásticos que satisfazem a propriedade de Markov, isto é, aqueles em que a probabilidade condicional de uma variável aleatória, $x_n = x(t_n)$, obter algum valor em um instante dado depende somente do valor que tenha obtido no instante anterior.[6,41,42] Em particular, aqueles que podem ocupar um número infinito numerável de estados $(e_1, e_2, e_3, ..., e_i, ...,)$, tal que a evolução do processo, dado um estado particular, depende só do estado presente, são denominados *cadeias de Markov*, e sua probabilidade de transição entre os estados e_i e e_j, está dada por,

$$p_{ij}(n,m) \triangleq P\{x_n = e_j | x_m = e_i\} \tag{2.19}$$

Se $p_{ij}(n,m)$ depende somente da diferença $n - m$, a cadeia de Markov é denominada *homogênea* no tempo, e a transição de probabilidade é denominada *estacionária*. Assim, um processo estocástico Markoviano pode ser escrito como,

$$p_j(n) = \sum_m p_{ij}(n,m) p_i(m) \tag{2.20}$$

onde $p_{ij}(n,m) = T(n,m)$ são os elementos de uma matriz, denominada *matriz estocástica*, tal que i) $T(n,m) \geq 0$ e ii) $\sum_n T(n,m) = 1$.

Um aspecto primordial destes processos é obter as condições ou propiedades de $T(n,m)$, tal que,

$$\lim_{j \to \infty} p_j = P \tag{2.21}$$

onde P é a solução estacionária, isto é, $\sum_m T(n,m)P(m) = P(n)$. Além do anterior, um processo Markoviano é reversível (reversibilidade microscópica) quando

$$T'(n,m) = T(m,n) = T(n,m) \tag{2.22}$$

onde $T^{'}(n,m)$ representa a matriz de transição para o processo reverso. Assim, a probabilidade estacionária, $P(n)$, pode ser escrita como,

$$[]\sum_{m}\{T(n,m)P(m) - T(m,n)P(n)\} = 0 \tag{2.23}$$

a expressão em (2.23) é denominada *balanceamento global*.

Um problema oposto à determinação da probabilidade estacionária, $P(n)$, dada uma matriz estocástica, $T(n,m)$, apresenta-se na área da física estatística, na determinação das propriedades de um sistema em equilibrio termodinâmico. De acordo com esta, tais propriedades são obtidas como uma medida de probabilidade, $P(n) \equiv \mu_{\Lambda\subset\mathbb{Z}^d}$ sobre $\Omega_\Lambda \subset \prod_{x\in\mathbb{Z}}\Omega_x$ ($\Omega_x \equiv \vec{x}$), dada por,

$$\mu_\Lambda\{\xi\} = Z_\Lambda^{-1}\exp[-U_\Lambda^\Phi(\xi)] \tag{2.24}$$

$$Z_\Lambda = \sum_{\xi\in\Omega_\Lambda}\exp[-U_\Lambda^\Phi(\xi)] \tag{2.25}$$

($U = \beta E(\xi)$, com $\beta = 1/kT$, tal que $E(\xi)$ é a energia da configuração ξ, T a temperatura absoluta e k uma constante (segundo a mecânica estatística, a constante de Boltzmann)).[13] μ_Λ é denominada *distribuição ou medida de Gibbs* e o campo correspondente, *estado de Gibbs*. Como exemplo de medida de Gibbs apresenta-se a análise do modelo de Ising exposto por Kindermann e Snell:[43] supondo que na equação 2.2 $H = 0$, e fazendo $x_k \equiv \sigma_k$, obtem-se a medida de Gibbs com a forma,

$$\mu_\Lambda(\xi) \equiv P(\xi) = Z^{-1}\exp\left[\beta J\sum_{i,j}\sigma_i(\xi)\sigma_j(\xi)\right] \tag{2.26}$$

que pode ser reescrita como,

$$[]P(\xi) = Z^{-1}\exp\left[\beta J(n_p(\xi) - n_i(\xi))\right] \tag{2.27}$$

onde n_u e n_v, são os números de pares adjacentes com igual sinal (+,+ ou -,-), e pares adjacentes com sinal distintos (+,- ou -,+), respectivamente. Se $n_t = n_u + n_v$, é o número total, e substituindo em 2.27 se obtém,

$$P(\xi) = Z^{-1}\exp\left[\beta J(n_t(\xi) - 2n_v(\xi))\right] \tag{2.28}$$

posto que n_t é constante, então,

$$P(\xi) = Z^{-1}\exp\left[-2\beta J n_v(\xi))\right] \tag{2.29}$$

se os valores (*spins*) foram gerados por uma cadeia de Markov de dois estados, $e_1 = +, e_2 = -$

e a matriz de transição, $T(n,m) \equiv T$,

$$T = \begin{array}{c} \\ - \\ + \end{array} \begin{array}{c} \begin{array}{cc} - & + \end{array} \\ \begin{pmatrix} p & 1-p \\ 1-p & p \end{pmatrix} \end{array} \tag{2.30}$$

então, a probabilidade está dada por,

$$P(\xi) = \frac{1}{2} p^{n_p(\xi)} (1-p)^{n_i(\xi)} \tag{2.31}$$

$$= \frac{1}{2} p^{n_t(\xi) - n_i(\xi)} (1-p)^{n_i(\xi)} \tag{2.32}$$

$$= \frac{1}{Z} \left(\frac{p}{1-p} \right)^{-n_i(\xi)} \tag{2.33}$$

Algoritmo de Metropolis

Em 1953, Metropolis et al.,[44] apresentaram uma metodologia para o cálculo computacional de propriedades de substâncias, como equações de estado de partículas individuais que interagem. Esse método, conhecido desde então como *algortimo de Metropolis*, é, de acordo com OLIVEIRA e TOME (2014), "[...] um dos algoritmos mais utilizados para construir a matriz estocástica [...]".[6]

Mediante o algoritmo de Metropolis, uma cadeia de Markov com matriz de transição ou kernel de transição é construída para uma distribuição de probabilidade dada. De acordo com LIANG et al. (2010), esse kernel somente é de utilidade práctica se a condição de reversibilidade é imposta. Isto é, se a condição de balanceamento, equação 2.23, é cumprida. E, o método de Metropolis considera uma aproximação de dois passos:
i) especifica uma distribuição simétrica proposta com função de distribuição de probabilidade $q(y|x) = q(x|y)$.
ii) um ajuste baseado em $q(y|x)$ visa aceite-rejeição, tal que a cadeia de Markov resultante é reversível.[45] No algoritmo 2.1 apresenta-se, esquemáticamente, a proposta original de Metropolis et al.:

os $\eta_i \in [-1,1]$ são números aleatórios (pseudoaleatórios), e E_i as energias de interação para as configurações ξ_i, em um retículo L.

Com uma distribuição asimétrica proposta, Hasting apresentou, em 1970, uma generalização do método de Metropolis. Esta é denominada *algoritmo de Metropolis-Hasting*, e está dada em dois passos:
1) ajuste de y baseado em $q(y|x)$

Entrada: $\xi \in \Omega,\ L \in \mathbb{Z}^2, N \in \mathbb{N}$
Saída: x_n, y_n, z_n
início
  $x_n = x_0 + \alpha\eta_1$
  $y_n = y_0 + \alpha\eta_2$
  $\Delta E = E_n - E_0$
  se $\Delta E < 0$ *ou* $exp(-\Delta E/kT) > \eta_3$ **então**
    $x_0 = x_n$; $y_0 = y_n$
  fim
fim

Algoritmo 2.1– pseudocódigo: Algoritmo de Metropolis

2) cálculo da razão de aceite

$$[]\alpha(x,y) = \min\left\{1, \frac{f(y)q(x|y)}{f(x)q(y|x)}\right\} \tag{2.34}$$

assim para $q(x|y) = q(y|x)$, obtem-se o algoritmo de Metropolis (amostrador de Metropolis).[45,46]

Tem-se, particularmente, para uma distribuição de probabilidades prosposta igual a:

$$P(n) = \frac{1}{Z}\exp[-\beta E(n)]\ (Boltzmann - Gibbs) \tag{2.35}$$

que a matriz estocástica é definida por:[6]

$$T(n,m) = \frac{1}{N}\exp[-\beta(E(n) - E(m))],\ \text{se } E(n) > E(m) \tag{2.36}$$

$$T(n,m) = \frac{1}{N},\ \text{se } E(n) \leq E(m) \tag{2.37}$$

e para elementos na diagonal dessa matriz,

$$T(n,n) = 1 - \sum_{n \neq m} T(n,m) \tag{2.38}$$

Métodos de Monte Carlo

O modelo de Ising, considerado como modelo estocástico, tem sido amplamente estudado como caso simples de sistemas com um grande número de configurações. Segundo MURTHY (2004), esse número é igual a 2^v, onde v é o número de *spins*.[47] Particularmente, simulações desse modelo, consistentes na amostragem aletória independente e com a mesma proabilidade dos estados, encontram-se nos métodos de Monte Carlo.

Em geral, o termo métodos de Monte Carlo refere-se a técnicas numéricas para a solução de algum problema, mediante os valores de uma variável aleatória.[21] Segundo KHOSHENEVISAN

(2007), o método Monte Carlo oferece uma forma para estimar o valor de alguma integral n-dimensional,

$$I(\phi) = \int_{[0,1]^n} \phi(x)\, dx \tag{2.39}$$

onde $\phi : \mathbb{R}^n \to \mathbb{R}$ é uma função Lebesgue integrável, mediante a seleção de variáveis aleatórias, X_j, independentes e identicamente distribuidas em um n-cubo $[0, 1]^n$, e a aplicação da lei de Kolmogorov dos grandes números.[48] Assim, obtem-se que

$$\lim_{N\to\infty} \frac{1}{N} \sum_{j=1}^{N} \phi(X_j) = I(\phi) \tag{2.40}$$

Esta técnica, utilizada inicialmente por E. Fermi nos estudos de difussão de nêutrons, e posteriomente por J. von Neummann et al.[?] foi denominada integração por método de Monte Carlo.

De acordo com HAM,[49] dois tipos de problemas são tratados com métodos de Monte Carlo: probabilísticos e determinísticos. Particularmente no caso probabilistico, números aleatórios são observados, e escolhidos de forma que estes simulam diretamente os processos físicos como processos aleatórios.[49] Os métodos de Monte Carlo têm sido utilizados em diversas áreas, por exemplo física estatística, solução de problemas de optimização, operações industriais, economia, busca de padrões, computação gráfica, linguistica, e estudos sociais.

Limita-se, neste documento, a discussão aos denominados métodos de Monte Carlo via cadeia de Markov (MCMC, por suas siglas em inglês). O termo MCMC refere-se a um tipo particular de modificação ao método convencional, incorporando a construção de uma cadeia Markoviana, como apresentado por Metropolis et al.[44] De acordo com RUBINSTEIN e KROESE (2008), os MCMC mais amplamente utilizados são o algoritmo de Metropolis-Hasting e o amostrador de Gibbs.[50] Uma descrição do primeiro foi dada na seção anterior. O segundo, diferentemente do algoritmo de Metropolis-Hasting, caracteriza-se pela aceitação de toda a amostragem.

Duas limitações no MCMC (algortimo de Metropolis-Hasting) são, segundo de acordo com LIAN et al.,[45] i) problemas com armadilhas locais, e ii) incapacidade para amostragens que utilizam distribuições com algums tipos de integrais. O primeiro caso refere-se a que a amostragem fica confinada em regiões de mínimo local de forma indefinida.[45]

Por fim, uma característica comúm aos métodos de Monte Carlo, como já foi dito, é o uso de números aleatórios ou, formalmente, pseudoaleatórios. Para a obtenção de tais números, se estes estão uniformemente distribuídos em $[0, 1]$ (ver equaçaõ 2.41) duas técnicas básicas são conhecidas: métodos congruentes e método de registro de troca de retorno,[51] ambos

caracterizados pelo uso de aritmética modular.

$$[]p(x) = \begin{cases} 1, & \text{se } 0 < x < 1 \\ 0, & \text{em outro caso} \end{cases} \tag{2.41}$$

Segundo MARSAGLIA e ZAMAN (1990) no seu trabalho sobre a construção de um gerador universal de números aleatórios, encontram-se dentre as propriedades desejáveis de tal gerador: i) aleatoriedade, ii) períodos longos, iii) eficiência, iv) repetitividade, v) portabilidade e vi) homogeneidade.[52]

Cálculo de momentos

Deriva-se do modelo de Ising o cálculo de algumas propriedades: magnetização média e variância da magnetização, por sitio. Ver equações 2.42 e 2.43, respetivamente.

$$[]m = \frac{1}{N}\langle \mathcal{M}^2 \rangle = \sum_i x_i \tag{2.42}$$

$$[]\chi = \frac{1}{N}\left\{ \langle \mathcal{M}^2 \rangle - \langle \mathcal{M} \rangle^2 \right\} \tag{2.43}$$

Derivam-se também, de forma análoga, os valores para a média e a variância da energia. Assim, considerando o modelo de Ising como estocástico e mediante o MCMC: algortimo de Metrópolis, com a distribuição de Boltzmann-Gibbs como probabilidade proposta, uma estimativa de tais propriedades pode ser obtida.[6] Em particular, de acordo com METROPOLIS et al. o cálculo de propriedades do sistema mediante o ensemble canônico está dado como:

$$[]\bar{F} = \frac{\left[\int F \exp(-E/kT) d^{2N}p d^{2N}q \right]}{\left[\int \exp(-E/kT) d^{2N}p d^{2N}q \right]} \tag{2.44}$$

onde $d^{2N}p d^{2N}q$ é um elemento de volume no espaço fase.[44]

Em teoria das probabilidades, média e variância pertencem a uma categoria de elementos que caracterizam a uma magnitude aleatória, são denominados, genericamente, *momentos*, e denotam-se como:

$$\alpha_r = \langle X^r \rangle = \int_{-\infty}^{\infty} x^r f(x) dx \tag{2.45}$$

Assim, o valor médio (ou esperança matématica) de uma magnitude aleatória, corresponde ao momento de ordem $r = 1$. As flutuações nessa magnitude aleatória são determinadas mediante

o momento de ordem $r = 2$, que é denominado *dispersão* ou variância, e está dado por,

$$D_x = \langle x^2 \rangle - \langle x \rangle^2 \tag{2.46}$$

Elementos de programação

Nesta seção, apresentam-se as noções, termos e métodos da área de computação necessarios no desenvolvimento do programa de simulação tratado neste documento. Para isto, divide-se estes temas em dois grupos principais. O primeiro refere-se aos elementos de programação essenciais para o desenvolvimento de um programa eficiente e portavel. O segundo trata sobre a construção de interpretes e compiladores, especicamente na definição de uma linguagem formal e de um autômata finito determinístico. Também, apresenta-se, de forma sucinta, o conceito de *interface*. Este embora não seja essencial para o desenvolvimento de um programa, melhora e possibita a comunicão entre este e otros programas. O que é util, por exemplo, na construção de ferramentas de análise de resultados ou de gráficos.

Segundo ELLIS et al.,[53] o processo de escrever um programa pode ser dividido em três passos básicos: i) definir claramente o problema, ii) analisar o problema e dividi-o em seus elementos fundamentais, e iii) codificar o programa segundo o plano desenvolvido no passo ii. No entanto, estes autores discutem, também, as razões pelas que a escrita deve seguir um protocolo estrito na sua elaboração, ao que denominam *escrita elegante*. Isto e, o projeto de construção de um programa deve levar em conta os conceitos de eficiência, confiabilidade, manutenibilidade e portabilidade.

A seguir, apresentam-se as noções na área da computação para os temas de portabilidade e eficiência. Portabilidade apresenta-se como o procedimento de encapsulação (denominado *modularização*) de funções e rotinas e otros elementos, embora o termo seja também empregado para se referir a capacidade de um program ser exacutado independentemente da arquitetura de máquina. Eficiência e discutida mediante os conceitos de estruras dinâmicas de dados, principalmente com as denominadas *listas encadeadas*.

módulos e componentes

Em suas notas sobre programação estruturada, em 1972, E. Dijkstra propôs que para o desenvolvimento de um programa, sucessivos refinamentos das tarefas devem ser feitos até estas possam ser expresadas por operações básicas. Segundo DIJKSTRA,[54] a probabilidade P de exatidão de um programa pode ser escrita como $P = p^N$, onde p e N são a probabilidade de exatidão individual dos componentes e o número destes, respectivamente.

De acordo com SAFONOV, em 1970, Parnas e Myers, seguindo as ideas de programação estruturada, apresentaram o conceito de modularização no desenvolvimento de programas, como resposta alternativa ao problema dos denominados programas *spaghetti*[55‡]. Deriva-se, assim, que um módulo é uma unidade de programa que tem definição sintática e

‡Programa *spaghetti*: programas com estrutura de control de fluxo incomprensivel.

método de ativação explícita, realiza tarefas específicas, e apresenta um conjunto definido de contatos com as outras parte do programa.

Segundo ELLIS et al.[53] um módulo é apropriado para empacotar grupos de procedimentos ou operações básicas, e para facilitar sua manipulação. Além de permitir a construção de novos operadoes e exterder o significado intrinseco e fucioalidade dos exsitentes.Na figura 3.1 apresenta-se, esquematicamente, o processo de modularização de um programa. Os elementos que podem ser contidos em um módulo são: variáveis, tipos derivados, funções, subrotinas ou outros módulos.

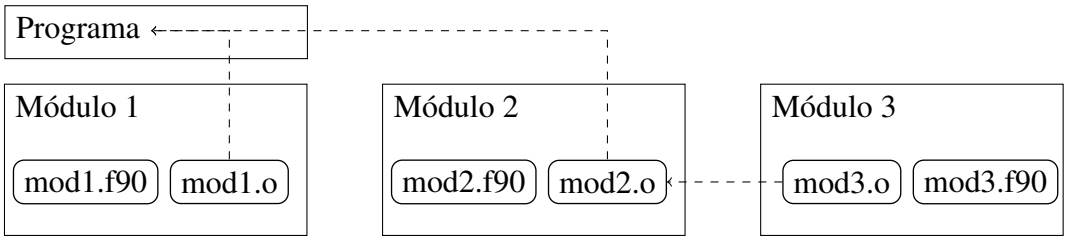


FIGURA 3.1 – Esquema de modularização proposto por Parnas e Myers

Listas encadeadas

Além dos procedimentos de modularização, um elemento importante para o desenvolvimento de algoritmos proposto por D. Knuth, em 1969, e que está relacionado com a organização dos dados, é o denominado esquema de *listas encadeadas*. A figura 3.2 mostra um diagrama para uma lista encadeada. Neste, o primeiro elemento na lista (1) é denominado *cabeça*(inicio), e o último *cauda*. Segundo KNUTH (1969), a informação em um programa refere-se a uma *relação estrutural* entre os dados e não somente a uma massa amorfa destes. E propôs que essa informação fosse armazenada em esquemas de memória flexível, no qual cada posição ou nó contem uma ligação ao próximo nó de uma lista de dados.[56]

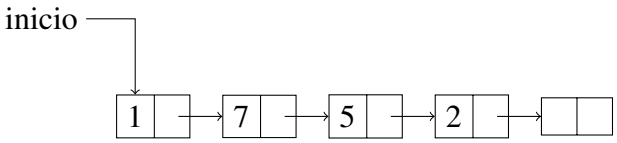


FIGURA 3.2 – Esquema de lista encadeada. A posição em cada parte da lista é dada por um ponteiro.

Segundo BROOKSHEAR,[57] a localização do inicio na lista é dada mediante o uso de um *ponteiro* [†] no endereço da primeira entrada dessa lista, e o final da mesma é dado por um *ponteiro* com atributo nulo ("null") ou zero.[57] Algumas caracteristicas que resultam da utilização do esquemas de listas encadeadas são: i) facilidade para inserir ou eliminar elementos da lista, e ii) buscas aleatórias em partes da lista são possíveis e empregam menos tempo que em

[†]Área da armazenamento que codifica os endereços de memória

uma lista sequêncial. Por fim, de acordo com ELLIS et al.[53] as listas encadeadas são utilizadas usualmente em inteligência artificial, simulação, redes neurais e na escrita de compiladores.

Compiladores

Um compilar é, segundo LOUDEN et al.[58] um programa que traduz de uma linguagem para outra. De acordo com GRUNE et al. este processo de conversão denomina-se *compilação*, e funciona a devido a dois fatores: i) a entrada é uma linguagem, assim apresenta estrutura, ii) a semântica da entrada é descrita em termos de essa estrutura.

Uma das funcionalidades do programa de simulação apresentado neste documento é sua aplicação na interpretação de funções matemáticas arbitrárias para o cálculo de energia de interação e posterior geração de um programa derivado que contem essa nova função. As ferramentas para realizar estes procedimentos encontram-se no tópicos da construção de compiladores ou de interpretadores.

Nesta seção apresentam-se dois conceitos fundamentais, que fazem parte dos etapas inicias na consttrução de um compilador. O de gramática livre de contexto e o de autômato finito determinístico.

Autômatos finitos determinísticos

Os autômatas finitos mostram-se como objetos matemáticos que descrevem alguns tipos de máquinas (algorítmos, no âmbito da computação).[58] Definem-se como uma 5-tupla, $(Q, \Sigma, \delta, q_0, F)$. Q, Σ, δ, q_0 e F, são um grupo de estados, um alfabeto, uma função de transição, um estado inicial e um conjunto de aceitação, respectivamente. Este autor, argumenta que um autômato finito representa um algoritmo que aceita cadeias de caracteres que casam com padrões.

Nas primeiras etapas de construção de um compilador estes autômatos empregam-se na construção de sistemas de varredura, isto é o algoritmo que lê um conjunto de caracteres, para uma análise léxica simultanea deste conjunto. de acordo com COOPER e TORCZON (2012), durante dita análise, um conjunto de caracteres é lido e um fluxo de palavras é produzido. Se uma palavra do fluxo é válida para o *compilador* (ou para o *intérprete*), então à esta é atribuida uma categoria o função gramatical (o par formado pela sequência de carateres e o seu valor o atributo associado denomina-se *lexema*).[59,60]

Em sua forma simples a análise léxica reconhece desde o código fonte do programa um caracter por vez, e para formas compostas compara-se, geralmente, com uma lista de palavras reservadas (como **if**, **for**, **while**, ou palavras que representam funções matemáticas). O processo de reconhecimento léxico é comumente representado por um diagrama denominado *de transição*. Neste, cada caracter representa um estado computacional, e se o último es-

tado é reconhecido, então o *lexema* será reconhecido. Segundo COOPER e TORCZON (2012) no reconhecimento da palavra **while** se produziría um diagrama semelante ao seguinte. Ver figura 3.3.

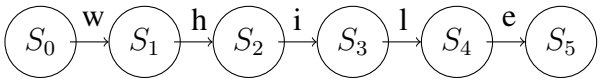


FIGURA 3.3 – Diagrama de transição

Ao finalizar a análise léxica, os lexemas produzidos são enviados a um processo de análise sintático, isto é, à parte onde cada palavra é organizada por sua função gramatical, e posteriormente aos procedimentos de interpretação ou geração de *código objeto* (compilação) no *intérprete* ou *compilador* respetivamente.

Gramáticas livres de contexto

Deve-se a Noah Chomsky a classificação das linguagems segundo sua complexidade gramatical.[58] Tal classificação têm sido empregada na construção de linguagems de programação.

De acordo com GRUNE et al.,[61] as gramáticas livres de contexto são essencias para descrever a estutura sintática e semântica de programas em linguagens de programação. Consistem de um conjunto de regras de produção e símbolos de inicio. Formalmente, definem-se como uma 4-tupla $G = (V_N, V_T, S, P)$. V_N e V_T são um conjunto de símbolos, e P um conjunto de regras de produção.

Como exemplo para explicar este conceito, Louden et al. apresentam o exemplo a seguir:

Considere a expresão aritmética

$$(34 - 3) * 42 \tag{3.1}$$

Segundo as regras gramáticais, uma **derivação** o sequênciade subtituições de nomes para dita cadeia é

$$(\text{número} - \text{número}) * \text{número} \tag{3.2}$$

Interfaces

Por fim, um procedimento prático para a construção de interfaces com outros programas, é o das denominadas chamadas ao sistema (*system call*). Isto refere-se a uma forma de comunicação com o núcleo (*kernel*) do sistema operacional, que permite o uso dos serviços

deste. Segundo VERMA,[62] quando uma chamada ao sistema é realizada, funções do *núcleo*, em UNIX, como *pipe*, *fork*, *wait*, etc., são utilizadas. A figura 3.4 mostra um diagrama do processo de chamada ao sistema.

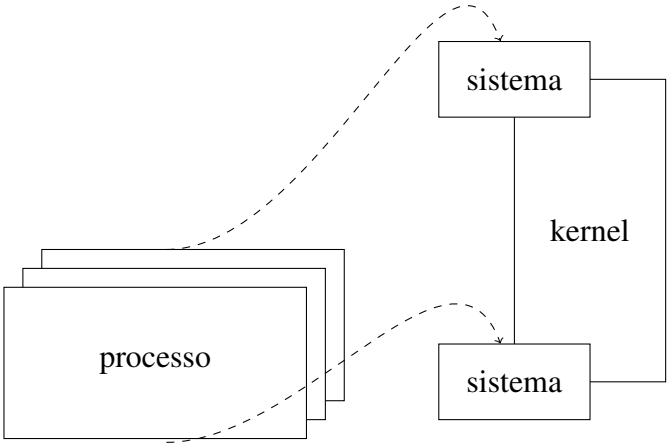


FIGURA 3.4 – Esquema de chamada ao sistema

Metodologia

Esta seção refere-se aos aspectos técnicos, isto é, ao processo de modularização e à definição de estructuras de dados, para a construção dos algoritmos que compõem o programa de Monte Carlo do que trata este documento.

A *linguagem de programação* na que os códigos fonte do programa estão escritos é **Fortran 90**, contidos na padronização **X3J3**[†] do Comité americano de padrões Fortran ("American Fortran Standards Committee"). Esta *linguagem*, que está catalogada na área de computação como de alto nivel e que foi desenvolvida, de acordo com JOSHI[?] (2003), para a avaliação numérica de fórmulas algébricas que requerem computação extensiva, tem sido de grande importância como ferramenta nas áreas da ciência.

O programa de *Monte Carlo* descrito neste documento, em adiante chamado de **MELQUIADES**[‡], divide-se na versão atual em 35 módulos (*module*) e apresenta dois tipos de variáveis derivadas (*derived type*), cada uma com um ponteiro (*pointer*) como atributo. O total de módulos pode ser classificado em uma de duas tarefas especificas executadas por **MELQUIADES**, a saber, uma seção para a inclução de novos tipos de potencial intermolecular e outra para o cálculo propriamente dito. O código fonte é executado serialmente e funciona de forma autônoma, isto é, não requere bibliotecas externas. Por fim, a documentação do código fonte está construida baseada na convenção do escritório de asimilação de dados (*DAO: Data Assimilation Office*), para os sistemas de documentação de códigos fonte.[63]

[†]US Fortran standards committee. J3. E inclui algumas modificações empregadas em *Fortran 2003* Disponivel em: <www.j3-fortran.org>

[‡]Personagem no romance de G. Garcia Marquez: *Cien años de soledad*

A seguir apresenta-se uma descrição, seguindo a ordem dada na seção anterior, da construção e forma de comunicação entre os módulos que compõem **MELQUIADES**: i) retículo, ii) configuração, iii) potencial, iv) MCMC, v) momentos e vi) análise léxica para novos potenciais.

A notação utilizada no código fonte para identificar os módulos e os procedimentos ao interior dos mesmos é:
i) nome de módulo : *m_<nome do módulo>*
ii) nome de subrotina : *r_<nome da subrotina>*
iii) nome de função: *f_<nome da função>*
O código de cada um destes módulos é apresentado na íntegra na seção dos apêndices.

Retículos

Definidos no espaço euclidiano $\mathbb{R}^3$, cada posição do retículo contém uma célula unitaria do sistema cúbico de faces centradas (**FCC**), figura 4.1. Na versão atual de **MELQUIADES** este tipo de distribuição geométrica é utilizada somente como uma condicição inicial, e é alterada durante a simulação.

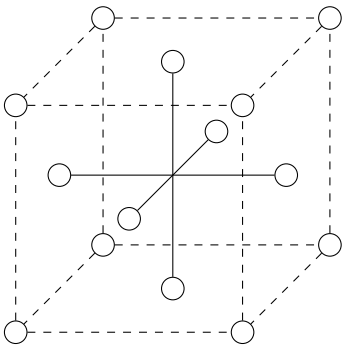


FIGURA 4.1 – Sistema cristalino cúbico de faces centradas ("FCC").

Em geral, outros tipos de modelos cristalinos ou distribuições aleatórias de *partículas* em $\mathbb{R}^3$ são aceitas. O formato para leitura dos arquivos que contém as coordenadas espaciais, *o retículo*, ao que se denominará *caixa de simulação*, é mostrado a seguir:

linha 1: n
linha 2: $nm_1\ nm_2\ ...\ nm_n \qquad ns_1\ ns_2\ ...\ ns_n \qquad m_1\ m_2\ ...\ m_n \qquad e_1\ e_2\ e_3$
linha 3: *seed*
linha 4: $scm_1 \quad 1 \quad xcm_1 \quad yc_1 \quad zcm_1$
linha 5: $ss_1 \quad n_1 \quad xs_1 \quad ys_2 \quad zs_3$
...
linha k+5: $ss_k \quad n_k \quad xs_k \quad ys_k \quad zs_k$
linha k+6: $scm_2 \quad 2 \quad xcm_2 \quad ycm_2 \quad zcm_2$

...
linha k+n: $scm_n \quad n \quad xcm_n \quad ycm_n \quad zcm_n$
linha k+n+1: $ss_{n+1} \quad n_{n+1} \quad xs_{n+1} \quad ys_{n+1} \quad zs_{n+1}$
...
onde $n \in \mathbb{N}^+$, é o número total de componentes distintas (líquidos, moléculas ou partículas), nm_i é o número de moléculas de cada tipo de componente, ns_j os sítios em cada tipo de componente, m_i as massas de cada molécula, e as e_i representam o tamanho de aresta de uma caixa cúbica. $seed \in \mathbb{N}$ é um valor para inicialização da função geradora de número aleatórios. Desde a linha quatro, os scm_i e ss_k referem-se ao símbolo ou rótulo que representa uma molécula ou grupo, e ao símbolo atômico de cada sítio nessa molécula, respectivamente. Por fim, os cm_i e s_i são as coordenadas cartesianas dos centros de massa das moléculas e dos sítios, respetivamente. Os valores inteiros representam um identificador sequencial para os centros de massa e o valor correspondente às entradas dos parâmetros de potencial para os sítios.

A seguir a definição das variáveis que representam às coordenadas cartesianas tanto dos centros de massa quanto dos sítios nas moléculas:

```
type box
 real(rkind), dimension(:,:),   pointer :: m_cmass
 real(rkind), dimension(:,:,:), pointer :: m_site
end type box
```

A definição como ponteiro (*pointer*), permite a alocação dinâmica de memória e o passo destas variáveis como argumentos para subrotinas e/ou funções. Os tamanhos destes arranjos numéricos são dados mediante:

```
type(box), pointer :: x
nullify(x)
 if(.not.associated(x)) then
  allocate(x,stat=error)
 end if


allocate(x%m_cmass(ndim,mxmol),stat=error)
allocate(x%m_site(ndim,mxns,mxmol),stat=error)
```

onde *ndim*, *mxmol* e *mxns*, são a dimensão no espaço euclidiano, o número máximo de moléculas e o número máximo de sítios por molécula, respectivamente. Igualmente o arranjo que define os símbolos atômicos está dado por:

```
character(len=2), dimension(:,:), pointer :: m_symbol
allocate(x%m_symbol(mxns,mxmol),stat=error)
```

Espaço de configuração

Define-se a dimensionalidade dos espaço de configuração como o conjunto formado pelos movimentos translacionais e rotacionais de uma molécula (Ver figuras 4.3 e 4.2.)

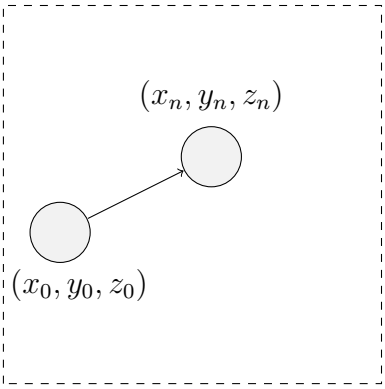


FIGURA 4.2 – Translação do centro de massa de uma molécula em $\mathbb{R}^3$.

o movimento translacional do ponto (x_0, y_0, z_0) ao ponto (x_n, y_n, z_n) é dado mediante adição (ou subtração) de uma quantidade aleatória, η, nas três coordenadas cartesianas. Para isso, o código implementado em **MELQUIADES** é o seguinte:

```
com_new(1) = (x%c_mass(1) + 2.0d0 * m_rand() - 1.0d0) * m_trans
com_new(2) = (x%c_mass(2) + 2.0d0 * m_rand() - 1.0d0) * m_trans
com_new(3) = (x%c_mass(3) + 2.0d0 * m_rand() - 1.0d0) * m_trans
```

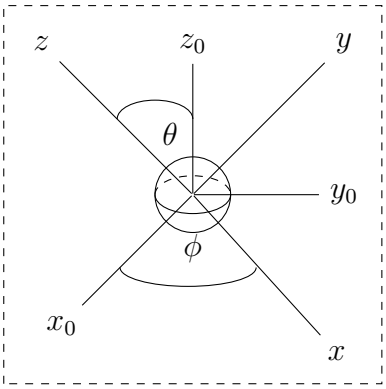


FIGURA 4.3 – Rotação da molécula em $\mathbb{R}^3$

De acordo com LANDAU e LIFSHITZ,[64] a orientação de um corpo em três dimensões pode ser descrita mediante os denominados *ângulos de Euler*. Estes ângulos mostram a relação entre um conjunto de vetores unitários no espaço inercial e um conjunto de vetores adjuntos ao corpo. A figura 4.3 mostra a rotação por um ângulo $\theta \in [0, \pi]$. Em notação matricial

as rotações podem ser escritas como[†]

$$\begin{bmatrix} \cos(\phi) & \sin(\phi) & 0 \\ -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) \end{bmatrix} \quad \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Da mesma forma que no caso da translação, a rotação é realizada mediante a adição de uma quantdade aleatória, η. O código correspondente no programa de *Monte Carlo* é:

```
dgamma = (2.0_rkind * m_rand() - 1.0) * dgamax
cosdg = dcos(dgamma)
...
if( iaxis == 1) then
 exinew = exiold
 eyinew = cosdg * eyiold + sindg * eziold
 ezinew = cosdg * eziold - sindg * eyiold
else if ...
...
```

Potencial de interação

A definição de potencias, no **MELQUIADES**, divide-se em duas partes básicas: Construção de funções (*function*) ou subrotinas (*subroutine*) e definição de *listas ligadas*.

Em **Fortran 90** refere-se aos subprogramas de procedimento como subrotinas. De acordo com ELLIS et al.[53] uma subrotina está definida como:

```
subroutine name (d1, d2, ...)
...
end subroutine name
```

onde os d_i representam argumentos da subrotina, tanto de entrada quanto de saída. Uma subrotina é acessada mediante uma chamada (*call*), a qual transfere o controle do programa a esse procedimento.

Na versão atual do **MELQUIADES** três subrotinas, representando formas de potencial de interação, estão implementadas: potencial de Lennard-Jones, de Buckingham e de Yukawa. Um fragmento do código para o potencial Lennard-Jones é mostrado a seguir:

```
subroutine p_ljones( rij, v, i ,j, x)
 type(box), pointer :: x
```

[†]Euler Angles. Wolfram. Disponivel em: <www.mathworld.wolfram.com>

```
 do m = 1, x%m_ns(i)
  alj_i = 4.0_rkind * t%m_eki * (t%m_ski ** 12)
  blj_i = 4.0_rkind * t%m_eki * (t%m_ski ** 6)
  ...
 end do
end subroutine p_ljones
```

O algoritmo para a construção do potencial de pares, isto é, a forma em que os pares de moléculas ou *partículas* interagem, está derivado do modelo de listas ligadas e denominado *método de células de listas ligadas*. De acordo com MATTSON e RICE,[65] neste método o espaço de simulação é dividido em células retangulares de forma que seus lados são maiores o iguais a um raio de corte do potencial de interação. A cada uma destas células é atribuída uma lista de *partículas* mediante uma ordenação dada pela posição no espaço de simulação.[14] A figura 4.4 mostra um esquema da divisão do espaço de simulação em células.

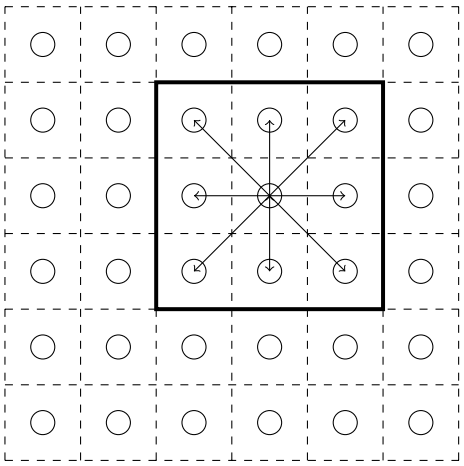


FIGURA 4.4 – Esquema de células de listas encadeadas.

O quadrado ao interior do retículo representa a adjacência ou primeira vizinhanza da célula central. Assim em $\mathbb{R}^3$ uma célula, na posicição i-ésima, apresenta N^3 células vizinhas, incluída ela propria. De acordo com AKL et al., dentre os modelos teóricos de complexidad ou de comportamento asimptótico deste tipo de algortimos, estão os tipos $\Theta(\log(n))$ ou $\Theta(n\log(n))$.[66] A seguir mostra-se o pseudocódigo 4.1 para o cálculo de energia de interação mediante o método de *células de listas ligadas*:

MCMC

Como mencionado na seção anterior, no método de *Metropolis-Hasting*, uma *cadeia de Markov*, com matriz de transição $T(n, m)$, é construída para uma medida de probabilidade proposta. Em **MELQUIADES** tal cadeia é construída, para o caso do *ensemble*

Entrada: $C_\alpha, C_\beta \in \mathbb{R}^3,\ P \in C_\alpha, Q \in C_\beta$
Saída: V_{ij}
início
  $r^2 = ||P - Q||$
  se $r^2 \leq r_c^2$ **então**
    $V_{ij} = f(P, Q)$
  fim
fim

Algoritmo 4.1– pseudocódigo Algoritmo de células de listas encadeadas

canônico, mediante o uso consecutivo de movimentos translacionais e rotacionais de cada uma das moléculas na *caixa de simulação*, e o cálculo da razão de aceite 2.34; e para o *ensemble isotérmico-isobárico*, mediante translações, rotações, modficação do volumen e o cálculo da razão de transição para cada uma destas variações. As equações 4.1 e 4.2, mostram a forma de cada caso:

$$[]\Delta E < 0 \quad \text{ou} \quad \exp(-\beta \Delta E) > \eta \tag{4.1}$$

$$[]\Delta E_V < 0 \quad \text{ou} \quad \exp\left[-\beta\left(\Delta E_V + p\Delta V - N\log\left(\frac{V_f}{V_i}\right)\right)\right] > \eta \tag{4.2}$$

onde $\eta \in [0, 1]$ representa uma numéro aleatório –em termos estrictos pseudoaleatório– uniformemente distribuído. Utiliza-se neste programa, para gerar os números aleatórios, a função proposta por Luscher e implementada na subroutina RANLUX[†] escrita em **Fortran77**. De acordo com JAMES,[67] o gerador de Lusher apresenta um período aproximado a 10^{171} e pode ser calculado em cinco (0 a 4) níveis distintos de qualidade. Onde o **nível 3** apresenta baixa probabilidades de correlação e o **nível 4** apresenta comportamento caótico. Em **MELQUIADES** a chamada a RANLUX é realizada mediante a subrotina $r\_rand()$ que está contida no modulo m_random.

Cada elemento na *cadeia de Markov*, está relacionado com um movimento que foi avaliado no cálculo da razão de aceite. Com isso, para cada *lista ligada*, os valores das coordenadas serão atualizados. Assim, uma molécula pertencente a uma dada célula i, pode permanecer nessa célula ou passar a outra, de acordo com os movimentos que realiza e são aceitos.

Cálculo de momentos

O aspecto metodológico desta seção refere-se aos processos de amostragem realizados ao final de cada *cadeia de Markov*. Assim, dois valores ou momentos são calculados: *média* e *variância*. De acordo com MEHROTRA et al. (1983), o cálculo de propriedades médias, para um conjunto de configurações X^N de um processo, está dado pela soma sobre as

[†] Descrita em *Comp. Phys. Comm.* ***79*** *(1994) 111-114*.

energias de cada configuração individual, como:

$$\langle E \rangle = \frac{1}{M} \sum_{t=1}^{M} E[X(t)] \tag{4.3}$$

onde $X(t)$ representa o estado de um sistema de N partículas no passo t-ésimo da cadeia de Markov.[68] Uma forma para o cálculo desses valores médios consiste em dividir o conjunto das realizações dos valores em blocos não superpostos de igual comprimento, e calcular a média para cada um.

Análise léxico

Refere-se esta seção a uma das partes essenciais da proposta do presente trabalho, isto é, em **MELQUIADES** novos tipos de potencial intermolecular de pares podem ser introduzidos mediante uma interface de usuário ou como texto (código ASCII) em um arquivo externo. Estas tarefas encontram-se contidas nos módulos: $m\_lexical$ e m_make. Nestes uma leitura (*scan*) e análise gramatical de uma cadeia de caracteres (que representa uma função de potencial intermolecular) são relizadas, e um fragmento de código é gerado, obtendo-se assim uma nova versão do programa. Com cada nova versão cálculos específicos para as funções de potencial introduzidas podem ser realizados. A parte inicial da análise léxica, isto é, o proceso de leitura junto com a comparção com uma lista de palavras reservadas, é também implementada no inicio do programa para a verficação dos arquivos de entrada *input* e *info* .

Resultados

Esta seção refere-se, especificamente, à descrição do programa de simulação desenvolvido neste trabalho, **MELQUIADES**, e a suas formas de utilização. A ordem de apresentação está relacionada com aquela na que o programa pode ser executado, a saber, como analisador léxico para a introdução de funções de potenciais intermoleculares, ou como programa para cálculo mediante o algortimo de Metropolis.

A execução básica de **MELQUIADES** segue o padrão dado pelo **IEEE**[†] para a construção de interfaces portáveis, **POSIX**[‡]. Assim, o programa é posto em funcionamento na linha de comandos com a seguinte sintaxe:

```
melquiades -option [atribute]
```

As opções disponíveis, na versão atual, são: i) *h* (help), ii) *v* (version), iii)*m* (make), e iv) *i* (input) (somente esta opção requer a definição de *atribute*). Os items i e ii oferecem informação geral sobre **MELQUIADES**, por exemplo ao executar na linha de comando:

```
melquiades -h
```

obtem-se

```
MELQUIADES, version 1.0
[...]
Usage:
melquiades [OPTION] ...[FILE]

[...]
Options:

-h --help
-v --version
-i --input FILE Standard input file [...]
-m --make       Enable builder [...]
```

Estas opções são avaliadas mediante um processo de análise sintática (*parsing*) por comparação com uma lista de palavras reservadas num proceso de seleção de casos. Ver apêndice A

[†]The Institute and Electronics Engineers
[‡]Portable Operating System Interface

Análise léxica

O procedimento de análise léxica é utilizado em **MELQUIADES** para a introdução de equações matemáticas, que representam formas ou tipos de potenciais intermoleculares de pares. Como foi dito na seções anteriores, neste procedimento um conjunto de caracteres é analisado letra a letra ou como grupos específicos de caracteres, e uma caracteristica gramatical é atribuída. Esta opção é executada na linha de comanado com a sintaxe:

```
melquiades -make
```

Com as opções `-make` ou `-m` uma interface de usuario é executada para a construção de uma nova função de potencial intermolecular de pares.

A forma genérica para tal função é a de uma função definida por partes (*piecewise*), com o caso padrão representado em uma única parte ou função continua no dominio todo. Assim, por exemplo, para incorporar o potencial de Sutherland,[69] ver equação 5.1,

$$[]\Phi(r) = \begin{cases} \infty, & \text{se } r < \sigma \\ -\epsilon \left(\frac{\sigma}{r}\right)^6, & \text{se } r > \sigma \end{cases} \tag{5.1}$$

um conjunto de respostas à seguinte lista de perguntas, na interface de usuário ou mediante um arquivo de texto, devem ser introduzidas:

```
[...]

Building a new intermoluecular potential...
[...]
Pairwise potential function
Enter the number of parts to the piecewise function [Default 1] : 2
Enter the pairwise potential in the specified format.
Enter the 1st piece:"inf"
[...]
Enter the 2nd piece:"-cmn/r2mn^3"
[...]
Definig the intervals to piecewise potential function
Enter the pairwise potential in the specified format.
Enter the 1st interval:"r2mn <= val*val"
Enter the 2nd interval:"r2mn > val*val"
```

Lexemas e palavras reservadas

A seguir apresenta-se, sucintamente, uma lista das tarefas realizadas em **MELQUIADES** para a análise de cadeias de caracteres usadas para introdução de novos tipos de função de potencial intermolecular. Em ordem de execução, as tarefas realizadas são:
* troca de todos os caracteres alfabéticos, na codificão ASCII, do formato de maiúsculas para minúsculas.
* avaliação de símbolos, diferentes dos alfabéticos, permitidos e não permitidos pela sintaxe. Os operadores aritméticos $(+, *, ...)$ ou de comparação $(><, ...)$ são aceitos, enquanto que os símbolos como , $, #, etc, não são permitidos.
* eliminação de duplicações nos operadores aritméticos, por exemplo $++$ ou $--$
* análise de palavras reservadas, isto é, a comparção de conjuntos de caracteres com uma lista definida em **MELQUIADES**, que especifica funções matemáticas, como: **sin**, **cos**,**exp**, etc.
Como foi mostrado para o potencial de Shuterland, um conjunto de caracteres reservados que representa o valor infinito também encontra-se incorporado no programa: *inf* ou *infty*. Apresenta-se no apêndice B a lista completa dos símbolos permitidos em **MELQUIADES**.

Lexemas em funções de potencial. Devem ser seguidas, para a introdução de funções de potencial intermolecular em **MELQUIADES**, as seguintes regras de atribução para caracteres alfabéticos:
*os simbolos alfabéticos válidos para representar parâmetros nas funções de potencial intermolecular são os conjuntos de letras [a-h], [o] e [q-w]
*as letras r, x, y e z representam o vetor posição ou as coordenadas cartesianas de uma partícula
*as letras [i-n] estão reservadas para representar índices de párticulas que interagem
*a letra p reserva-se para representar o valor numérico π.

Assim, por exemplo, para um potencial tipo Lennard-Jones a sintaxe correta é a seguinte: `4.0 * eij * ( (sij / r2ij) ^ 12 - (sij / r2ij ) ^ 6 )`, onde `eij` e `sij` representam os valores de epsilon e sigma, respectivamente (valores conjuntos ou parâmetros de mistura das partículas *i* e *j*), e o símbolo `r2ij` representa o quadrado da distância da partícula *i* à partícula *j*.

A ordem de escrita das funções de potencial, isto é, de esquerda a direita, define a posição em que os parâmetros da função deverão ser escritos no arquivo de parâmetros. Considerando o exemplo acima, uma vez finalizada a análise léxica os parâmetros `eij` e `sij` serão atribuídos a posições de uma tabela na ordem crescente iniciando na coluna 1 (um), e sem repetições. Na próxima seção será estendida a discusão sobre este tópico.

Ao finalizar a rotina da interface de usuário, o módulo m_make, a funcão e os intervalos são escritos e avaliados como válidos, estes são introduzidos no módulo genérico $mpot$, e uma copia na íntegra do restante do código fonte é feita e armazenada em um novo

diretorio para posterior compilação. Fragmentos do código fonte para no módulo $m\_lexical$ são apresentados no apêndice C.

Programa de Monte Carlo

Seguindo a sequência de execução do programa **MELQUIADES** mediante a opção `--input`, apresentam-se os elementos do programa necessários para o cálculo do potential de interação mediante algoritmo de Metropolis.

Arquivos externos

Quatro arquivos externos em formato ASCII ou texto plano devem ser fornecidos para o funcionamento de **MELQUIADES**:
* arquivo de entrada (=*input*)
* arquivo de informação (=*info*)
* arquivo com coordenadas das partículas (ou caixa de simulação)
* arquivo de parâmetros

Arquivo de entrada (*input*). Este arquivo ASCII contém as palavras-chave válidas no programa, entre as que se encontram os nomes dos arquivos de parâmetros e das coordenadas das partículas. Também especifica-se no arquivo de entrada o tipo de ensemble termodinâmico, o potencial, e as condições para a simulação. A lista completa de palavras-chave é apresentada no apêndice D. De mesma forma que para as opções da linha de comandos, nas palavras-chave também é realizada uma análise sintática para verificação de validade.

O arquivo de entrada (*input*) está constituído por duas colunas, separadas por um ou mais espaços. Na primeira coluna escrevem-se as palavras-chave, e na segunda o valor de atributo que corresponde com cada palavra. Assim, por exemplo, a palavra-chave `ensem`, que representa o tipo de ensemble termodinâmico a ser utilizado, deve ir seguida de um espaço e um de três valores possíveis: `nvt`, `npt` ou `mvt`. O fragmento a seguir representa as condições típicas básicas para o cálculo em **MELQUIADES**:

```
title   Work_title
info    Information.dat
coor    Box_simulation.Box
pot     m_ljq
ensem   nvt
param   Parameters.dat
temp    25.0
press    1.0
dr       0.1
```

Arquivo de informação (*info*). Neste arquivo, dados sobre o tipo e quantidade de parâmetros a serem utilizados são especificados. Assim, define-se, por exemplo, se os parâmetros são de mistura (obtidos mediante uma regra de combinação) ou simples (cada partícula especifica o valor de seus parâmetros). Segue um fragmento para este tipo de arquivo:

```
single 2
O 4
H 5
file single.par
```

A palavra `single` especifica que cada partícula tem um valor para o parâmetro dado. Os números 4 e 5 no exemplo representam a posição (número de línea) do parâmetro no arquivo *single.par*.

Caixa de simulação. Como foi mencionado na seçaõ anterior, a caixa de simulação é construidacomo um retículo em $\mathbb{R}^3$, no qual cada nó representa a posição (em coordenadas cartesianas) de uma partícula. A distribuição neste tipo de caixa é somente um exemplo particular, e qualquer outro tipo também pode ser utilizado como caixa de simulacão, incluídas distribuições aleatórias. A seguir apresenta-se o formato no qual **MELQUIADES** lê tais caixas:

```
2
1000 3000 4 3 18.0 32.0 61.51 61.51 61.51
314159265
OH2     1  0.000  0.000 0.000
O       6  0.000  0.000 0.000
H       7  0.759  0.586 0.000
H       7 -0.759  0.586 0.000
X       8  0.000  0.155 0.000
...
CH3OH   2 -15.37 -24.60 -9.23
CH      9   0.00   0.00  0.00
OM     10  -1.43   0.00  0.00
HM     11   0.30   0.89  0.00
```

Na sequência de aparição no texto os números representam: * [2] a quantidade de tipos diferentes de partículas (ou moléculas)
* [1000, 3000] número total de tais tipos de partículas
* [4, 3] número de sitios em cada tipo de moléculas
* as massas das moléculas e as dimensões da caixa
* o número que inicia o gerador de números pseudaleatórios
* um rótulo para as moléculas e as coordenadas cartesianas dos centros de massa
* o símbolos atómicos para os sitios e suas coordenadas cartesianas.

No apêndice E apresenta-se a rotina para a leitura da caixa de simulação.

Arquivo de parâmetros. No caso geral, este arquivo está conformado por um número equivalente à quantidade de parâmetros diferentes que possa ter uma função de potencial intermolecular específica. Assim, a ordem na que a função de potencial foi escrita determina aquela em que os parâmetros deven ser introduzidos neste arquivo. Para o potencial *12-6-1* ou potencial de Lennard-Jones e coulombico:

```
4.0 * eij * ( (sij / r2ij)^12 - (sij / r2ij)^6) + (qij)/rij
```

o arquivo de parâmetros tem o seguinte formato

```
# symbol     eij      sij        qij
    O      0.1521   3.15061    -0.834
    H      0.0000   0.00000     0.417
```

No caso de parâmetros simples, isto é, valores definidos para cada sitio (átomo), o módulo m_rules contém uma rotina para o cálculo de parâmetros conjuntos ou de mistura, ver apêndice F. Na versão atual de **MELQUIADES** as regras para mistura de parâmetros são as seguintes: i) *Good-Hope, GH*(*J. Chem. Phys.* 53, 540(1970)), ii) *Lorentz-Berthelot, LB*(*Der Annalen Physics 12, 127(1880)*), iii) *Fender-Halsey*

Células de listas encadeadas

Para a implementação do método de células de lista encadeadas um valor de raio de corte deve ser especificado no arquivo *input*, mediante a palavra chave `rcut`. O código fonte correspondente encontra-se no módulo $m\_precells$, ver apêndice G, onde o número total de células vizinhas à célula avaliada é calculado, e pode ser 1, 8 ou 27, sendo 27 o máximo para primeiros vizinhos.

O procedimento fundamental para construção das listas encadeadas é a atribuição de um índice de célula a cada partícula, segundo sua posição na caixa de simulação (em coordenadas cartesianas). Este índice é representado por um número entero e calculado mediante uma função (*function procedure*) contida no módulo $m\_precells$. Segue um fragmento de código fonte na *function procedure*:

```
f_idcell= ix+ iy*div(k)%cells(1)+iz*div(k)%cells(1)*div(k)%cells(2)
```

os valores ix, iy, iz, representam a parte entera dos valores nas coordenadas cartesianas das partículas, e os valores `div(k)%cells` os tamanhos das aresta das células (em $\mathbb{R}^3$)

Cálculo de interação intermolecular

O procedimento de avaliação de energia de interação é realizado posteriormente à definição das listas encadeadas. Assim, para uma partícula dada, um cálculo de distância com as células vizinas é realizado, no espaço euclideo. Posteriormente, com estas distâncias são

calculados os valores para a energia potencial de interação. Ver o módulo $m\_interaction$ no apêndice H.

Nesta rotina, inicialmente é calculada a interação de uma partícula com aquelas que têm o mesmo indice de célula, e posteriormente com as restantes células vizinhas. O tipo de potencial de interação intermolecular é escolhido mediante um procedimento de seleção de casos (*select case*). Na versão atual de **MELQUIADES** quatro opções estão implementadas: i) *Lennard-Jones*, ii) *Buckhingham*, iii) *Yukawa* e, iv) *pot* (função de potencial definida pelo usuario).

O procedimento contido no módulo $m\_interaction$ é aplicado tanto para o cálculo de energía configuracional (cálculo sequencial com todas as partículas na caixa de simulação), quanto na construção da cadeia de Markov, no algoritmo de Metropolis.
As palavras-chave correspondentes no arquivo de entrada (*input*) são:
* Lennard-Jones : *m_ljq*
* Buckihma : *m_bq*
* Yukawa : *m_ljy*
* pot : *m_pot*

Cadeia de Markov

Divide-se a rotina para construção de uma cadeia de Markov nos seguintes passos: i) escolha aleatória de uma partícula na caixa de simulação, ii) escolha de um grau de liberdade, isto é, movimento das partículas no espaço euclideo, iii) avaliação do criterio de Metropolis, iv) atualização das listas encadeadas.

Como foi dito acima, em **MELQUIADES** implementa-se o gerador de número pseduoaleatório RANLUX, proposto por Lusher. O generador é utilizado no nível 3, com um periodo 10^{171}. O código fonte, escrito originalmente em Fortran 77, foi adaptado para funcionar ao interior do módulo m_random em Fortran 90, e o valor aleatório é obtido com a função (*function procedure*) $m\_rand()$. Segue o fragmento de código fonte para gerar o índice de partícula (número inteiro com valor máximo igual ao total de partículas na caixa de simulação):

```
do
  xi = m_rand()
  imol = int(max_mol * xi) + 1
  [...]
end do
```

Em $\mathbb{R}^3$, dois tipos de movimentos estão implementados em **MELQUIADES**: translação e rotação, com as palavras-chave *dr* e *dangle*, respectivamente, onde o valor de potencial de interação é calculado para cada movimento, e um criterio de aceite é avaliado, segundo o formalismo

do algoritmo de Metropolis. O fragmeto de código fonte correspondente é mostrado a seguir (ver código na íntegra no apêndice I):

```
! Transformation matrix in R3
com_new(1) = (com_old(1)+(2.0_rkind*m_rand()-1.0_rkind)*sim%m_dr)
com_new(2) = (com_old(2)+(2.0_rkind*m_rand()-1.0_rkind)*sim%m_dr)
com_new(3) = (com_old(3)+(2.0_rkind*m_rand()-1.0_rkind)*sim%m_dr)
```

O criterio de aceite, avaliado segundo o algoritmo de Metropolis é construído desde uma medida de probabilidade dada pela denominada distribuição de Boltzmann, no formalismo de Gibbs, como ensemble canônico ou isotérmico-isobárico. A palavra-chave correspondente é *ensem*, com atributos possíveis *nvt*, *npt* e *mvt*. O código para esta rotina é mostrado a seguir:

```
[...]
deltv  = e_new - e_old
deltvb = detltv * y%m_beta
[...]
if(deltv < 10.0d0) then
  if(deltv <= 0.0d0) then
   accept = 1
   t%m_move = t%m_move + 1
  else if(exp(-deltvb) > m_rand()) then
    accept = 1
    t%m_move = t%m_move + 1
  end if
end if
```

Por fim, se o movimento de uma partícula é aceito e o índice da célula muda, as listas encadeadas são atualizadas. Ver código fonte no apêndice J.

Os procedimentos acima mencionados são realizados para o total de números de passos (elementos na cadeia de Markov). No arquivo *input* a palavra-chave correspondente é *nstep*

Cálculo de momentos

A média dos valores de energía de interação e sua correspondente variância, ou primeiro e segundo momentos de acordo com a teória das probabilidades, são calculados em **MELQUIADES** mediante a rotina $m\_average$. Este cálculo é executado com atributo *.true.* da palavra chave *aver* no arquivo de entrada (*input*). A seguir apresentam-se os fragmentos para o cálculo do primeiro e segundo momentos:

* primeiro momento: energia por molécula
* segundo momento :
capacidade calorífica específica, C_p
compressibilidade isotérmica, κ
coeficiente de expansão térmica, α

Cálculo de médias

```
aeng    = t%m_aeng / real(t%m_amv, rkind)
aengsq  = t%m_aens / real(t%m_amv, rkind)

aetot   = t%m_aeto / real(t%m_amv, rkind)
aetotsq = t%m_aets / real(t%m_amv, rkind)
[...]
write(*,'(" Total energy in this run :  ",e20.10," kcal/mol ")') aetot
write(*,'(" Total energy per molecule:  ",e20.10," kcal/mol ")') aeng
```

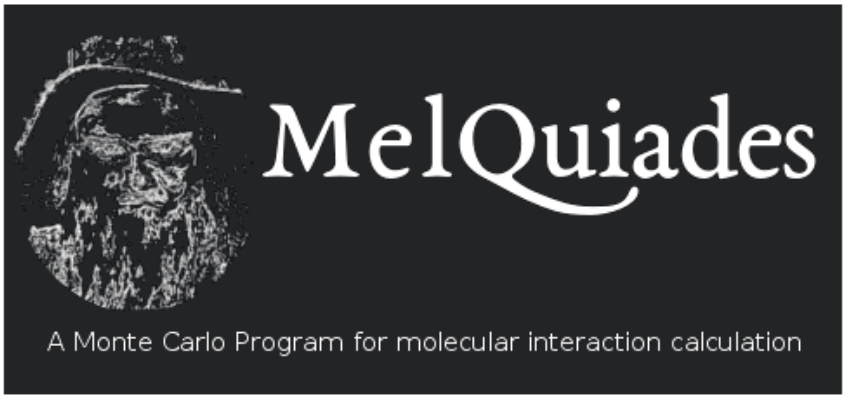
Cálculo de variâncias (flutuções)

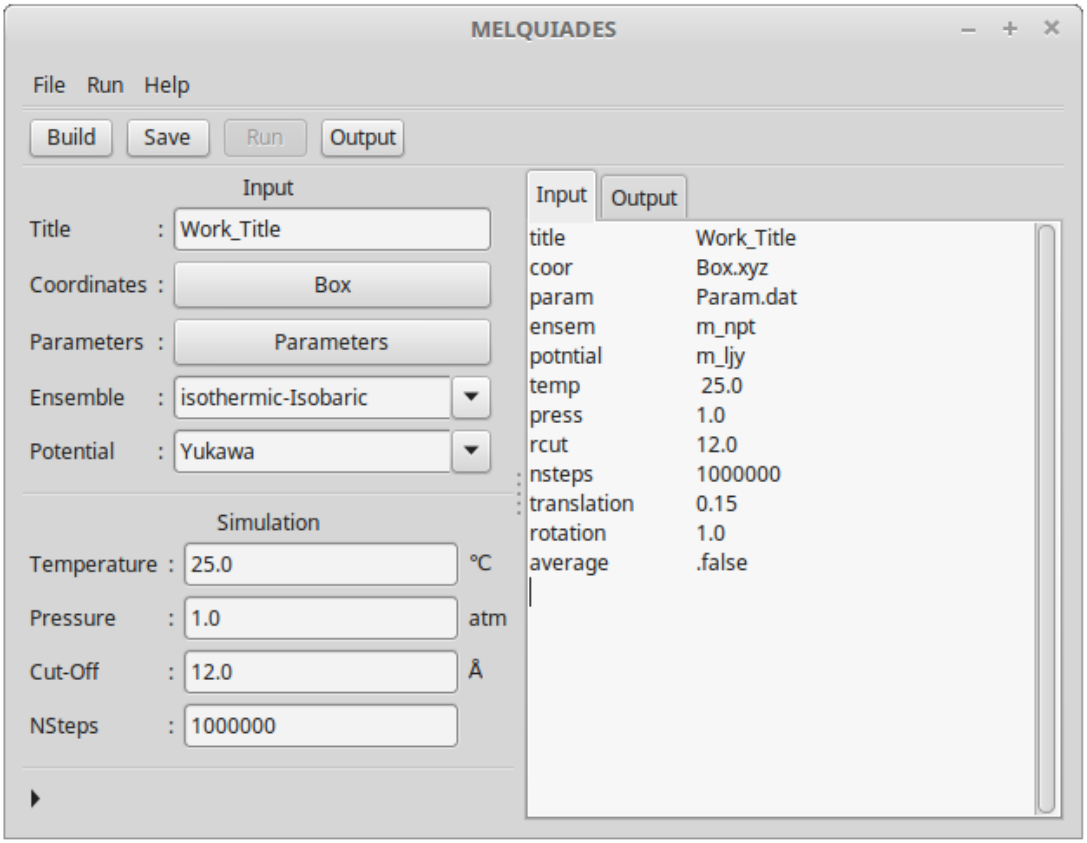
```
cp=(t%m_aens/real(t%m_amv, rkind))-((t%m_aent/real(t%m_amv, rkind))**2)
cp=cp / (kB * y%m_temper * y%m_temper * real(y%m_mxmol))
cp=cp * 1000.0_rkind

kappa=(t%m_avos/real(t%m_amv,rkind))-((t%m_avol/real(t%m_amv,rkind))**2)
kappa=kappa/(kB*y%m_temper*(t%m_avol/real(t%m_amv,rkind)))
kappa=kappa*1.45836_rkind*0.00001_rkind

tmp1= t%m_avol / real(t%m_amv, rkind)
tmp2= t%m_aent / real(t%m_amv, rkind)

alpha = (t%m_aalp / real(t%m_amv, rkind)) - (tmp1 * tmp2)
alpha = alpha / (kB * y%m_temper * y%m_temper * tmp1)

write(*,'(" Specific heat capacity [Cp]   :  ",e20.10," cal/mol K")') cp
write(*,'(" Isothermal compressibility[] :  ",e20.10," atm")') kappa
write(*,'(" Thermal expansion coeff.[]   :  ",e20.10," K")') alpha
```

Interface gráfica de usuário

Como ferramenta alternativa para a construção do arquivo de entrada (*input*) uma interface gráfica de usuario, (*GUI, Graphical User Interface*), também foi desenvolvida neste trabalho. O código fonte desta interface está escrito na linguagem C e utiliza as bibliotecas gráficas GTK+2 (*Gimp tools kits*). Nesta interface as palavras chaves são representadas como

rótulos (*label*) e seus atributos podem ser introduzidos ou escolhidos desde uma lista. Assim que as opções para o cálculo são especificadas, um arquivo de texto com as características do *input* é gerado. A seguir apresentam-se capturas de tela que mostram as janelas que constituem a interface.





Teste de desempenho

Nesta seção apresentam-se os testes realizados para avaliação do desempenho de **MELQUIADES**, específicamente no relativo à escalabilidade representada no tamanho do

sistema, isto é, o número de moléculas na caixa de simulação.

O teste que segue foi realizado para o cálculo do tempo gasto no cálculo da energia potencial de interação de um fluido de Lennard-Jones. A tabela 5.1 mostra o valor médio para uma amostragem de trinta (30) repetições e um total de $10^5 + 10 * nmol$ configurações em cada caso. O teste foi realizado sobre uma arquitetura Intel(R)Core(TM) i7 CPU 870

TABELA 5.1 – Tempo médio em segundos gasto para cálculo de um fluido de Lennard-Jones

nmol	média	desvio	células
864	13.1741	0.0337	64
1372	13.1852	0.0375	125
2048	13.7268	0.0266	216
2916	16.2656	0.0901	216
4000	16.7198	0.0382	343

os valores correspondentes aos números de moles, *nmols*, 2048 e 2916, representam o comportamento esperado no algoritmo de células de listas encadeadas, com comportamento asintótico $n \log n$. Segundo o qual o rendimento é melhorado com o aumento do número de células em que a caixa de simulação pode ser dividida.

A figura 5.1 mostra um teste de desempehno do **MELQUIADES** como função do número de sitios em três moléculas.

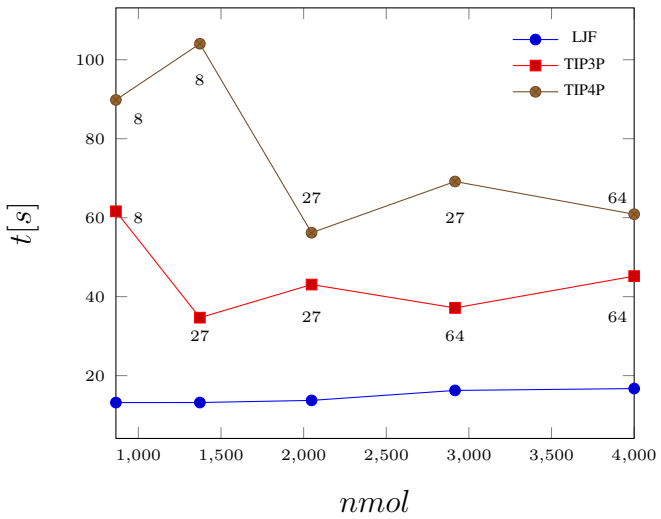
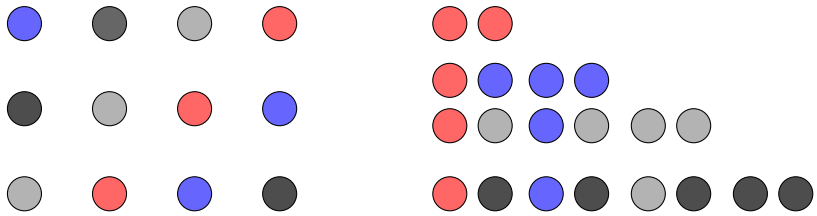


FIGURA 5.1 – Tempo médio gasto como função do número de sitios

O decrecimento inicial no tempo de cálculo para os potenciais **TIP3P** e **TIP4P**, pode ser associado ao incremento no número de células no esquema de listas encadeadas. Assim, e de acordo o modelo de comportamento asintótico do algoritmo, o tempo gasto para números pequenos de celulas é proporcional a $O(N^2)$, enquanto para números maiores tal comportamento é $O(N \log N)$

Avaliação da capacidade multicomponente por pares

Caso: Mistura de quatro (4) componentes para o potencial transferível **TIP3P**. O diagrama a seguir representa duas configurações diferentes para uma caixa de simulação contendo quatro componentes.



A tabela 5.2 mostra o valor para a energia média, em $kcal/mol$, por componente, para a mistura acima.

TABELA 5.2 – Energia de interação média, em kcal/mol, para um mistura de quatro componentes

	L1	L2	L3	L4
L1	-2524.745			
L2	-4714.939	-2646.673		
L3	-5106.870	-4619.187	-2624.911	
L4	-4719.420	-5054.152	-4603.876	-2601.192
$\langle E \rangle$	**-9.803**	Ref0*. -9.75		

Para a análise da capacidade multicomponente, uma caixa de simulação com 4000 moléculas de agua (potencial **TIP3P**) foi construida definindo um total de quatro tipos de componentes, cada um com mil (1000) moléculas. Os simbolos Ln na tabela acima representam cada um dos componentes. Os valores consignados na tabela representam as energías potenciais de interação por pares, assim, por exemplo, a energia de mistura para os componentes L3-L2 é igual a -4619.187.

O valor médio, $\langle E \rangle$, por molécula para a mistura é **-9.803** kcal/mol. De acordo com VAN der SPOLE et al.(1998), o valor de energia potencial correspondente para uma caixa de simulação com 820 moléculas é -9.75 kcal/mol (Ref0)

Por fim, um teste de validação para o cálculo de propriedades termodinâmicas, foi realizado. Neste, uma caixa de simulação com 4000 moléculas TIP3P (e um teste similar para o potencial TIP4P), a 25C de temperatura e 1atm de pressão. As tabelas 5.3 e 5.4 mostram os valores de energía de interação média por molécula, densidade e calor de vaporização para os potencias TIP3P e TIP4P, respectivamente.
O teste acima mencionado foi realizado no ensemble isotérmico-isobárico (npt), com um total de 10^9 configurações. Pode ser observado nas tabelas anteriores que as propriedades termodinâmicas calculadas estão em concordância com os valores experimentais reportados (J. Chem.

TABELA 5.3 – Propriedades termodinâmicas para modelo TIP3P

	MELQUIADES	Ref.1 *	Exp. †
$\langle E \rangle$ kcal mol^{-1}	-9.8742	-9.86	-9.92
ρ g cm^{-3}	1.0044	0.982	0.997
ΔH_v kcal mol^{-1}	10.4671	10.45	10.51

TABELA 5.4 – Propriedades termodinâmicas para modelo TIP4P

	MELQUIADES	Ref.1	Exp.
$\langle E \rangle$ kcal mol^{-1}	-9.9979	-10.07	-9.92
ρ g cm^{-3}	0.9988	0.999	0.997
ΔH_v kcal mol^{-1}	10.5909	10.66	10.51

Phys., 79, 926, 1983).

Conclusões

Deriva-se do desenvolvimento deste trabalho um programa de simulação computacional, **MELQUIADES**, para o cálculo de energía potencial de interação em sistemas multicomponentes, mediante o método de Monte Carlo via cadeia de Markov, e para a avaliação de modelos arbitrários de potencial.

MELQUIADES está composto por um total de 8235 linhas de código fonte, distribuídas em 35 modulos de procedimento, escritas na linguagem Fortran 90 e modificações de **Fortran 2003**. É executado em forma serial e de manera autônoma, sem uso de bibliotecas externas. A codificação segue padrão para o desenvolvimento de programas **POSIX** e implementa como procedimento geral para o cálculo de energia potencial de interação, mediante o método Monte Carlo via cadeia de Markov, o algoritmo de células de listas encadeadas.

A definição das condições para a simulação é dada mediante um arquivo externo, denomindo archivo de entrada, em codificação ASCII. De forma alternativa para a construção deste archivo foi desenvolvida uma interface grafica de usuário, na linguagem C e a ferramenta gráfica GTK+2.

Em **MELQUIADES**, equações matemáticas arbitrárias podem ser introduzidas como modelos de potencial intermolecular de pares. Isto é facilitado mediante um procedimento de análise léxico, tal que não é necessaria a escrita de código fonte (programção).

Os testes de desempenho realizados a **MELQUIADES** mostram o comportamento descrito teóricamente pelo algoritmo de células de listas encadeadas, e a escalabilidade com o número de componentes (número de tipos de moléculas). Adicionalmente, o teste de validação mostrou que os valores de energía potencial de interação e das propriedades termodinâmicas concordam com dados experimentais.

É importante destacar a funcionalidade de **MELQUIADES** como ferramenta para a avaliação de equações matemáticas como modelos de potencial de interação. O esquema de introdução de conjuntos de caracteres em texto simples elimina a escritura de código fonte e, portanto, de programação para o usuário final. Com isto copias do código com as variações introduzidas são obtidas, e um novo código objeto ou executável é gerado (tal que o programa original não é modificado, e sua estrutura de dados permanece fixa em cada uma das novas versões). Adicionalmente, o esquema de modularização no que o programa está construído permite a portabilidad e a implementação de ferramentas para a análise de resultados, com um mínimo de codificação (como é o caso dos denominados procedimentos de *chamada ao sistema*).

Finalmente, avanços para a execução em paralelo, desde a versão atual, estão

sendo realizados no esquema de memoria compartilhada (=*shared memory multiprocessing, OpenMP*).

* **MELQUIADES** foi apresentado no XVIII Simpósio Brasileiro de Química Téorica - SBQT 2015 (Pirenópolis-GO, 22-25 de novembro de 2015) e reconhecido como melhor trabalho

Referências bibliográficas

1. PIA, M. G.; BASAGLIA, T.; BELL, Z. W. & DRESSENDORFER, O. V. "The impact of Monte Carlo simulation: a sicentometric analysis of scholarity literatiure". *JOint International Conference on Supercomputing in Nuclear Applications ans Monte Carlo 2010*, : n/a–n/a, 2010.

2. MARTIN, M. G. "MCCCS Towhee: a tool for Monte Carlo molecular simulation". *Molecular Simulation*, **39**: 1212, 2013.

3. SHAH, J. K. & ET AL., E. M.-R. "Cassandra: An open source Monte Carlo package for molecular simulation". *Journal of Computational Chemistry*, : n/a–n/a, 2017.

4. PURTON, J. A.; CRABTREE, J. C. & PARKER, S. C. "DL_MONTE: A general purpose program for parallel Monte Carlo simulation". *Molecular Simulation*, **39**: 1240, 2013.

5. FREITAS, L. C. G. "DIADORIM: a Monte Carlo Program for liquid simulations including quantum mechanics and molecular mechanics (QM/MM) facilities: Applications to liquid ethanol". *Journal of the Brazilian Chemical Society*, **20**: 1541, 2009.

6. TOME, T. & DE OLIVEIRA, M. J. Dinâmica Estocástica e Irreversibilidade. São Paulo: edusp, 2014.

7. LANDAU, D. P. & BINDER, K. A Guide to Monte Carlo Simulation in Statistical Physics. United Kingdom: Cambridge University Press, 2000.

8. ISING, E. "Beitrag zur Theorie des Ferromagnetismus". *Zeitschrift für Physik*, **31**: 253, 1925.

9. ONSAGER, L. "Crystal Statistics. I. A Two-Dimensional Model with an Order-Disorder Transition". *Phys. Rev.*, **65**: 117, 1944.

10. DOVE, M. T. Introduction to Lattice Dynamics. New York: Cambridge University Press, 1993.

11. MICCIANCIO, D. & GOLDWASSER, S. Complexity of Lattice Problems: A Cryptographic Perspective. Boston: Kluber Academic Publisher, 2002.

12. LAVIS, D. A. & BELL, G. M. Statistical mechanics of lattice systems. Berlin: Springer-Verlag, 1999.

13. RUELLE, D. Thermodynamic Formalism: The Mathematical Structure of Equilibrium Statistical Mechanics. New York: Cambridge University Press, 2004.

14. ALLEN, M. P. & TILDESLEY, D. J. Computer Simulation of Liquids. Oxford University Press, 1989.

15. LENZ, W. “Beiträge zum Verständnis der magnetischen Eigenschaften in festen Körpern”. *Physikalische Zeitschrift*, **21**: 613, 1920.

16. BRUSH, S. “History of the Lenz-Ising Model”. *Rev. Mod. Phys.*, **39**: 883, 1967.

17. CHEN, M. F. From Markov chains to non-equilibrium particle systems. Singapore: World Scientific, 2004.

18. SUSSMAN, G. J. & WISDOM, J. Structure ans Interpretation of Classical Mechanics. Cambrigde: The MIT Press, 2000.

19. ARNOLD, V. I. Mathematical Methods of Classical Mechanics. USA: Springer-Verlag, 1989.

20. FRENKEL, D. & SMIT, B. (Eds.). Understanding Molecular Simulation: From Algorithms to Applications. Orlando, FL, USA: Academic Press, Inc., 1996.

21. MACKEOWN, P. K. & NEWMAN, D. J. Computational Techniques in Physics. Bristol: Adam Hilger, 1987.

22. GEORGII, H. O. Gibbs Measures and Phase Transition. Berlin: De Gruyter, 2011.

23. SINAI, Y. G. Dynamical Systems II: Ergodic Theory with Applications to Dynamical Systems and Statistical Mechanics. Berlin: Springer-Verlag, 1989.

24. LENNARD-JONES, J. E. “Cohesion”. *Pro. Phys. Soc.*, **43**: 461, 1931.

25. SLATER, J. C. & KIRKWOOD, J. G. “The van der Waal forces in gases”. *Physical Review*, **37**: 682, 1931.

26. OREA, P.; REYES-MERCADO, Y. & DUDA, Y. “Some universal trends of the Mie(n,m) fluid thermodynamics”. *Physics Letters A*, **372**: 7024, 2008.

27. DELHOMMELLE, J. & MILLIÉ, P. “Inadequacy of the Lorentz-Berthelot combinig rules for accurate preditions of equilbrium properties by molecular simulation”. *Molecular Physics*, **99**: 619, 2001.

28. KULINSKII, V. L. “Global isomorphism betweenn the Lennard-Jones fluids and the Ising model”. *J. Chem. Phys.*, **133**: 034121, 2010.

29. FREITAS, L. C. G. & BOTELHO, L. F. “Cálculo de propriedades termodinâmicas de líquidos via simulação computacional”. *Química Nova*, **17**: 489, 1994.

30. JORGENSEN, W. L. “Transferable Intermolecular Potential Functions for Water, Alcohols, and Ethers. Application to Liquid Water”. *J. Am. Chem. Soc.*, **103**: 335, 1981.

31. YUKAWA, H. “On the Interaction of Elemtary Particles”. *PTP*, **17**: 48, 1935.

32. PRIGOGINE, I. & RICE, S. A. Advences in Chemical Physics. New York: John Wiley & Sons, 2009.

33. OMELYAN, I. P.; FENZ, W.; FOLK, R. & MRYGLOD, I. M. "Phase diagrams of Ising fluids Yukawa-Lennard-Jones interactions form an intergral equation approach". *Eur. Phys. J. B.*, **51**: 101, 2006.

34. KREJCÍ, J.; NEZBEDA, I.; MELNYK, R. & TROKHYMCHUK, A. "Mean-spherical approximation for the Lennard-Jones-like two Yukawa model: Comparision againts Monte Carlo data". *Condense Matter Physics*, **14**: 1, 2011.

35. BUCKINGHAM, R. A. "The classical equation of state of gaseous helium, neon and argon". *Proc. R. Soc. A*, **168**: 264, 1938.

36. GALE, J. D. "Empirical potential derivation for ionic materials". *Philosophical Magazine B*, **73**: 3, 1996.

37. ERRINTONG, J. R. & PANAGIOTOPOULOS, A. "Phase equilibria of modi ed Buckingham exponential-6 potential from Hamiltonial scalinggrand canonical Monte Carlo". *J. Chem. Phys.*, **109**: 1093, 1998.

38. CHEN, J. & LEE, J. D. "The Buckinham Catastrophe in multiscale modelling of fracture". *Int. J. Theoretical and Applied Multiscale Mechanics*, **2**: 3, 2011.

39. GLAUBER, R. J. "Time-Depent Statistics of the Ising Model". *Journal of Mathematical Physics*, **4**: 294, 1963.

40. LIGGETT, T. M. Interacting Particle Systems. Bristol: Springer-Verlag, 1985.

41. PUGACHEV, V. S. Introducción a la Teoría de las Probabilidades. Moscu: Editorial MIR, 1973.

42. PAPOULIS, A. & PILLAI, S. U. Probability, Random Variables and Stochastic Process. Singapore: Mc Graw Hill, 2002.

43. KINDERMANN, R. & SNELL, J. L. Contemporary Mathematicals: Markov Field and Their Applications. Amercan Mathematical Society, 1980.

44. METROPOLIS, N.; ROSENBLUTH, A. W.; ROSENBLUTH, M. N.; TELLER, A. H. & TELLER, E. "Equation of State Calculations by Fast Computing Machines". *The Journal of Chemical Physics*, **21**: 1087–1092, 1953.

45. LIANG, F.; LIU, C. & CARROLL, R. J. Advanced Markov Chain Monte Carlo Methods: Learning from past samples. United Kingdom: Wiley, 2010.

46. HASTING, W. K. "Monte Carlo sampling methods using Markov chains and their applications". *Biometrika*, **57**: 97, 1970.

47. MURTHY, K. P. N. Monte Carlo Methods in Statistic Physics. India: Universities Press, 2004.

48. KHOSHNEVISAN, D. Probability. Rhode Island: American Mathematical Society, 2007.

49. HAMMERSLEY, J. Monte Carlo Methods. London: Springer Science & Business Media, 2013.

50. RUBINSTEIN, R. Y. & KROESE, D. P. Simulation and the Monte Carlo Method. United States of America: Wiley-Interscience, 2008.

51. GENTLE, J. E. Random Number Generation and Monte Carlo Metohds: Statistic and Computing. New York: Springer Science & Bussines Media, 1998.

52. MARSAGLIA, G. & ZAMAN, A. "Toward a Universal Random Number Generator". *Statistics Probability Letters*, **8**: 35, 1990.

53. ELLIS, T. M. R.; PHILIPS, I. R. & LAHEY, T. M. Fortran 90 Programing. Harlow: Addison-Wesley Publisher Ltd., 1994.

54. DAHL, O. J.; DIJKSTRA, E. W. & HOARE, C. A. R. Structure Programming. London: Academic Press, 1972.

55. SAFONOV, V. Using Aspect-Oriented Programming for Trustworthy Software Development. United States of America: Wiley-Interscience, 2008.

56. KNUTH, D. E. The Art of Computer Programming. Massachusetts: Addison-Wesley, 1969.

57. BROOKSHEAR, J. G. Ciência da computação: uma visão abrangente. Brazil: Bookmad Editors, 2012.

58. LOUDEN, K. Compiladores: Principios e Prácticas. Brazil: Thompson, 2004.

59. COOPER, K. D. & TORCZON, L. Engineering a Compiler. Amsterdam: Morgan Kaufmann, 2012.

60. AHO, A. V.; LAM, M. S.; SETHI, R. & ULLMAN, J. D. Compilers: Principles, Techniques, Tools. Boston: Pearson, 2007.

61. GRUNE, D. & ET AL., H. E. B. Moder Compiler Desing. New York: WILEY, 2000.

62. VERMA, A. Unix and Shell Programming. India: Laxmi Publications, 2008.

63. SAWYER, W. & DA SILVA, A. ProTeX: A Sample Fortran 90 Source Code Documentation System. Maryland: Goddard Space Flight Center, 1997.

64. LANDAU, L. D. & LIFSHITZ, E. M. Mechanics. Course of Theoretical Physics. Great Britain: Pergamon Press, 1976.

65. MATTSON, W. & RICE, B. M. Near-neighbor calculation usng a modified cell-linked list method. 1999. 119.

66. AKL, S. G.; FIALA, F. & KOCZKODAJ, W. W. Advances in Computing and Informational - ICCI 90. Canada: Springer, 1990.

67. JAMES, F. "RANLUX: A Fortran implementation of the high-quality pseudorandom number generator of Luscher". *Computer Physics Communications*, **79**: 111, 1994.

68. MEHROTRA, P. K.; MEZEI, M. & BEVERIDGE, D. L. "Convergence accelaration in Monte Carlo computer simulation on water and aqueous solutions". *J. Chem. Phys.*, **78**: 3156, 1983.

69. GRABEN, H. W. & PRESENT, R. D. "Third Virial Coeffient for the Sutherland Potential". *Reviews of Modern Physics*, New York, **36**: 1025, 1964.

Apêndice A

Código fonte para construção de opções desde linha de comandos, mediante sistema de seleção de casos (=*select case*)

```
!-----------------------------------------------------------------------
! MELQUIADES, A Monte Carlo program for molecular interaction calculation.
! Copyright (C) 2015 Asdrubal Lozada
!-----------------------------------------------------------------------
!> @author Asdrubal Lozada
!> @brief Contains a routine for get arguments from cmd line and to
request by files.
!> @date 10 Jun 2016
!-----------------------------------------------------------------------
module m_inquire
   use m_units, only : input_file, name_ifile
   use m_help
   use m_make
   !
   implicit none
   !
   logical :: call_make = .false.
   !
contains
!-----------------------------------------------------------------------
!> @author Asdrubal Lozada
!> @brief This routine parsing over command arguments.
!> @date 10 Jun 2016
!-----------------------------------------------------------------------
   subroutine r_cmd_line
      implicit none
      !
      ! Local variables:
      integer                         :: ncmd, len_cmd, i, j, stat
      character(len=20)               :: cmd
      character(len=3), parameter     :: version = '1.0'

      stat = 0
      ncmd = command_argument_count()

      if ( ncmd > 0 ) then
         i = 1
```

```
call get_command_argument( i, cmd, status=stat )

if ( stat /= 0 ) then
   write(*,'(a)') 'Reading arguments in command line failed.'
   stop
end if

cmd = trim(adjustl(cmd))

select case(cmd)
case('-h','--help')

    if ( ncmd > 1 ) then
      write(*,'(a)')' Number of arguments not allowed.'
      stop
    end if

    call r_cmd_help
    stop
case('-i','--input')

    if ( ncmd > 2 ) then
      write(*,'(a)')' Number of arguments not allowed.'
      stop
    end if

    j = i + 1
    call get_command_argument( j, cmd, status=stat )

    if ( stat /= 0 ) then
       write(*,'(a)') 'Read arguments in command line failed.'
       stop
    end if

    len_cmd = len_trim(cmd)

    if ( len_cmd < 1 ) then
       call r_input_help
       stop
    elseif (cmd(1:1) == '-') then
         write(*,'(a)') 'Invalid character in name input file.'
         stop
```

```
            end if

            name_ifile = trim(adjustl(cmd))
            call r_request( input_file, name_ifile )
         case('-m','--make')

            if ( ncmd > 1 ) then
              write(*,'(a)')' Number of arguments not allowed.'
              stop
            end if
            call r_make
! Builds a new potential
            call_make = .true.
         case('-v','--version')

            if ( ncmd > 1 ) then
              write(*,'(a)')' Number of arguments not allowed.'
              stop
            end if

            write(*,'(a,a)') 'MELQUIADES v ',version
            write(*,'(a)') 'Copyrigth (C) 2015 Asdrubal Lozada'
            write(*,'(a)')
            write(*,'(a)') 'Written by Asdrubal Lozada Blanco'
            write(*,'(a)') 'see: <http://www.lqt.dq.ufscar.br>.'
            stop
         case default
            call r_input_help
            stop
         end select
      else
         call r_cmd_help
         stop
      end if
   end subroutine r_cmd_line
```

Apêndice B

Operadores aritméticos e lógicos válidos em **MELQUIADES** para a construção de novos potenciais:

operador	uso	descrição
$+$	$+$	soma
$-$	$-$	substração
$*$	$*$	multiplacação
$=$	$==$	igual ao
$<>$	$/=$	diferente de
$>$	$>$	maior que
$<$	$<$	menor que
$>=$	$>=$	maior ou igual que
$<=$	$<=$	menor ou igual que

Apêndice C

Rotina para análise léxico dos conjuntos de caracteres introducidos como novas funções de potencial.

```
!------------------------------------------------------------------------
! MELQUIADES, A Monte Carlo program for molecular interaction calculation.
! Copyright (C) 2015 Asdrubal Lozada
!------------------------------------------------------------------------
!> @author Asdrubal Lozada
!> @brief Parse the user defined function potential
!> @date 10 Jun 2016
!------------------------------------------------------------------------
module m_lexical
   use m_buffer
   use m_globals
   use m_error

contains

   subroutine r_parsing( pot, id, fun, wrong )
    implicit none
    type(ptr_string), pointer :: pot
    integer, intent(in) :: id, fun
    type(globals), pointer :: pop
! Local variables
    character(len=length_string) :: arg
    integer :: lstring
    integer :: k, j
    logical :: wrong
! Symbols list
    character(len=1), dimension(21) :: sym
    integer, parameter :: isym = 21
! Stranger expressions
    character(len=2), dimension(8) :: strg
    integer, parameter :: istrg = 8

    sym = (/'!','@','#','$','%','_',';',',','{','}','[','}',&
          & ':','"','?','\','=','&','|','\','~'/)

    strg = (/'++','--','^^','//','()','&&','||','~~'/)

    wrong = .false.
```

```
      select case(fun)
      case(1) ! Parse potential
       arg = trim(adjustl(pot%pstr(id)%str))
      case(2) ! Parse domine
      end select

      lstring = len(trim(adjustl(arg)))


! Shift case upper to lower
     do k = 1, lstring
       if( lge(arg(k:k),'A') .and. lle(arg(k:k),'Z') ) then
         arg(k:k) = achar(iachar(arg(k:k)) + 32 )
       end if ! up > lw

      do j = 1, isym
        if( arg(k:k) == sym(j) ) then
          call bad_msg( arg, k, wrong)
        end if ! sym
      end do ! j
     end do ! k

! Stranger expressions
     do k = 1, lstring
      do j = 1, istrg
        if( arg(k:k+1) == strg(j) ) then
         call bad_msg( arg, k, wrong )
        end if ! stranger
      end do ! j
     end do ! k

    call trim_inside(arg)
    call layout_final( arg, pop )

    pot%pstr(id)%str = arg


   end subroutine r_parsing
```

Apêndice D

Palavras chaves usadas no arquivo *input*

Palavra chave	exemplo	tipo de variavel	Descrição
title	title_work	character	
coor	file_coord	character	name file of the box simulation
info	file_info	character	name file of information
ensem	nvt	character	thermodynamic ensemble type
pot	m_ljc	character	type of interaction potential
temp	25.00	real	value in centigrades
press	1.00	real	value in atmospheres
rcut	8.50	real	value in angstroms
pbc	.true.	logical	periodical boundaries conditions
nstep	10000	integer	number of steps
dr	0.10	real	value in angstroms
dang	10.00	real	value in degree
solt	.true.	logical	active use the second cut-off
aver	.true.	logical	active the averagee calculation
mxvol	200.	real	maximum value to change volume

Apêndice E

Rotina para leitura de caixa de simulação

```
subroutine r_sim_box( box, sim )
    implicit none
    type(sim_box),    pointer :: box
    type(simulation)          :: sim
    !
    ! Local variables
    integer           :: ios = 1, id = 1
    integer           :: k, mt, mm, ms
    integer           :: imol = 0, isite = 0
    character(len=len_file)  :: error_box

    error_box = trim(adjustl(name_ibox))
    error_label = 'error_alloc_box'

    rewind(input_box)
    read( input_box, *, iostat = ios ) max_types
    call r_error_read( ios, error_box, id, 1 )

    if ( max_types < min_type ) then
       write(*,'(a)')' Undefined number of components.'
       stop
    end if

    nullify(box)

    call r_alloc_box( box, 0 )

    id = id + 1; ios = 1
    read( input_box, *, iostat = ios ) (box%m_nmol(k), k = 1, max_types),&
        & (box%m_nsite(k), k = 1, max_types), (box%m_mass(k),
        k = 1, max_types),&
        & (box%m_edge(k), k = 1, max_dim )
        call r_error_read( ios, error_box, id, 1 )

    id = id + 1
    read( input_box, *, iostat = ios ) m_seed
    call r_error_read( ios, error_box, id, 1 )
    if( .not. sim%m_average ) m_seed = 314159265
```

```
   max_mol    = sum(box%m_nmol)
   max_sites  = sum(box%m_nsite)
   call r_alloc_box( box, 1 )

   do mt = 1, max_types
      do mm = 1, box%m_nmol(mt)
         imol = imol + 1
         read( input_box, *, iostat = ios ) box%m_mol_label(imol),&
            & box%m_mol_id(imol),box%m_com_xyz(:,imol)
            id = id + 1
            call r_error_read( ios, error_box, id, 1 )
            do ms = 1, box%m_nsite(mt)
               isite = isite + 1
               read( input_box, *, iostat = ios )
               box%m_site_label(isite,imol),&
                  & box%m_site_id(isite,imol),
                  box%m_site_xyz(:,isite,imol)
                  id = id + 1
                  call r_error_read( ios, error_box, id, 1 )
            end do
            isite = 0
      end do
   end do

end subroutine r_sim_box
```

Apêndice F

Código fonte com as regras para a construção de parâmetros de mistura:

```
    subroutine r_rules( box, par, sim, np, iskip, jskip )
      implicit none
      type(sim_box),    pointer :: box
      type(sim_par)             :: par
      type(simulation)          :: sim
      integer, intent(in)       :: np
      integer, intent(in)       :: iskip, jskip
      !
      ! Local variables
      integer :: k
      real(rkind) :: pi, pj, ti, tj

      select case(sim%m_rules)
      case('r_goodhope','GH')
! Good-Hope rule
! J. Chem. Phys., 53, 540 (1970)
         pi = box%m_one_par(1,iskip)
         pj = box%m_one_par(2,jskip)
         !
         if ( pi*pj /= 0.0_rkind ) then
            box%m_two_par(1,jskip,iskip) = dsqrt(pi*pj)
         else
            box%m_two_par(1,jskip,iskip) = 0.0_rkind
         end if
         !
         pi = box%m_one_par(2,iskip)
         pj = box%m_one_par(2,jskip)
         !
         if ( pi*pj /= 0.0_rkind ) then
            box%m_two_par(2,jskip,iskip) = dsqrt(pi*pj)
         else
            box%m_two_par(2,jskip,iskip) = 0.0_rkind
         end if
         !
      case('r_lorberth','LB')
! Lorentz-Berthelot
! Der Annalen Physik, 12, 127 (1881)
      pi = box%m_one_par(1,iskip)
      pj = box%m_one_par(1,jskip)
```

```
      if ( pi*pj /= 0.0_rkind ) then
         box%m_two_par(1,jskip,iskip) = (pi*pj)/2.0_rkind
      else
         box%m_two_par(1,jskip,iskip) = 0.0_rkind
      end if
      !
      pi = box%m_one_par(2,iskip)
      pj = box%m_one_par(2,jskip)

      if ( pi*pj /= 0.0_rkind ) then
         box%m_two_par(2,jskip,iskip) = dsqrt(pi*pj)
      else
         box%m_two_par(2,jskip,iskip) = 0.0_rkind
      end if
   case('r_fenderh','FH')
! Fender-Halsey
      pi = box%m_one_par(1,iskip)
      pj = box%m_one_par(1,jskip)

      if ( pi*pj /= 0.0_rkind ) then
         box%m_two_par(1,jskip,iskip) = (pi*pj)/2.0_rkind
      else
         box%m_two_par(1,jskip,iskip) = 0.0_rkind
      end if
      !
      pi = box%m_one_par(2,iskip)
      pj = box%m_one_par(2,jskip)

      if ( (pi+pj) == 0.0_rkind ) then
         write(*,'(a)') ' Warning: Division by zero.'
         write(*,'(a)') ' Check value epsilon in parameters file.'
         stop
      end if

      if ( pi*pj /= 0.0_rkind ) then
         box%m_two_par(2,jskip,iskip) = (2.0_rkind*pi*pj)/(pi+pj)
      else
         box%m_two_par(2,jskip,iskip) = 0.0_rkind
      end if
   case('r_halgren','HHG')
! Halgren
```

```
      pi = box%m_one_par(1,iskip)
      pj = box%m_one_par(1,jskip)

      if ( (pi+pj) == 0.0_rkind ) then
         write(*,'(a)') ' Warning: Division by zero.'
         write(*,'(a)') ' Check value sigma in parameters file.'
         stop
      end if

      if ( pi*pj /= 0.0_rkind ) then
         box%m_two_par(1,jskip,iskip) = ((pi*pi*pi)+(pj*pj*pj))/&
                                        &((pi*pi)+(pj*pj))
      else
         box%m_two_par(1,jskip,iskip) = 0.0_rkind
      end if

      pi = box%m_one_par(2,iskip)
      pj = box%m_one_par(2,jskip)

      if ( (pi+pj) == 0.0_rkind ) then
         write(*,'(a)') ' Warning: Division by zero.'
         write(*,'(a)') ' Check value epsilon in parameters file.'
         stop
      end if

      if ( pi*pj /= 0.0_rkind ) then
         box%m_two_par(2,jskip,iskip) = (4.0_rkind*pi*pj)/&
                                        & (dsqrt(pi)+dsqrt(pj))
      else
         box%m_two_par(2,jskip,iskip) = 0.0_rkind
      end if
   case('r_whagler','WH')
      pi = box%m_one_par(1,iskip)
      pj = box%m_one_par(1,jskip)

      if ( (pi+pj) == 0.0_rkind ) then
         write(*,'(a)') ' Warning: Division by zero.'
         write(*,'(a)') ' Check value sigma in parameters file.'
         stop
      end if

      if ( pi*pj /= 0.0_rkind ) then
```

```
         box%m_two_par(1,jskip,iskip) = (((pi*pi)**3 + (pj*pj)**3)/&
                                        &2.0_rkind)**(1./6.)
      else
         box%m_two_par(1,jskip,iskip) = 0.0_rkind
      end if

      ti = pi; tj = pj

      pi = box%m_one_par(2,iskip)
      pj = box%m_one_par(2,jskip)

      if ( pi*pj /= 0.0_rkind ) then
         box%m_two_par(2,jskip,iskip) = (2.0_rkind*(ti*ti*ti)*&
                           &(tj*tj*tj)*dsqrt(pi*pj))/(ti+tj)
      else
         box%m_two_par(2,jskip,iskip) = 0.0_rkind
      end if
   case default
      if ( sim%m_potential == 'm_ljc'.or. sim%m_potential == 'm_ljy' ) then
! Good-Hope rule
! J. Chem. Phys., 53, 540 (1970)
         pi = box%m_one_par(1,iskip)
         pj = box%m_one_par(2,jskip)
         !
         if ( pi*pj /= 0.0_rkind ) then
            box%m_two_par(1,jskip,iskip) = dsqrt(pi*pj)
         else
            box%m_two_par(1,jskip,iskip) = 0.0_rkind
         end if
         !
         pi = box%m_one_par(2,iskip)
         pj = box%m_one_par(2,jskip)
         !
         if ( pi*pj /= 0.0_rkind ) then
            box%m_two_par(2,jskip,iskip) = dsqrt(pi*pj)
         else
            box%m_two_par(2,jskip,iskip) = 0.0_rkind
         end if
      end if
      !
      if ( sim%m_potential == 'm_bc' ) then
! Good-Hope rule
```

```
! J. Chem. Phys., 53, 540 (1970)
         pi = box%m_one_par(1,iskip)
         pj = box%m_one_par(1,jskip)

         if ( (pi*pj) == 0.0_rkind ) then
            write(*,'(a)') ' Warning: Division by zero.'
            write(*,'(a)') ' check value sigma in parameters file.'
            stop
         end if

         if ( pi*pj /= 0.0_rkind ) then
            box%m_two_par(1,jskip,iskip) = dsqrt(pi*pj)
         end if
         !
         pi = box%m_one_par(2,iskip)
         pj = box%m_one_par(2,jskip)

         if ( pi*pj /= 0.0_rkind ) then
            box%m_two_par(2,jskip,iskip) = dsqrt(pi*pj)
         else
            box%m_two_par(2,jskip,iskip) = 0.0_rkind
         end if
         !
         pi = box%m_one_par(3,iskip)
         pj = box%m_one_par(3,jskip)

         if( (pi-6.0_rkind) == 0.0_rkind&
            &.or.(pj-6.0_rkind)==0.0_rkind) then
            write(*,'(a)') ' Warning: Division by zero.'
            write(*,'(a)') ' Check value alpha in parameters file.'
            stop
         end if

         if ( pi*pj /= 0.0_rkind ) then
            box%m_two_par(3,jskip,iskip) = dsqrt(pi*pj)
         else
            box%m_two_par(3,jskip,iskip ) = 0.0_rkind
         end if
         !
       end if
         !
         if ( sim%m_potential == 'm_pot' ) then
```

```
! Good-Hope rule
! J. Chem. Phys., 53, 540 (1970)

            do k = 1, np
               pi = box%m_one_par(k,iskip)
               pj = box%m_one_par(k,jskip)

               if ( pi*pj /= 0.0_rkind ) then
                  box%m_two_par(k,jskip,iskip) = dsqrt(pi*pj)
               else
                  box%m_two_par(k,jskip,iskip) = 0.0_rkind
               end if
            end do
         end if

      end select

   end subroutine r_rules
```

Apêndice G

Rotina para divisão da caixa de simulação em células e construção de listas ligadas.

```
module m_precells
   use m_kind
   use m_error
   use m_boxtype
   use m_simtype
   !
   implicit none

   type linked_list
      integer, dimension(:), pointer :: head
      integer, dimension(:), pointer :: object
   end type linked_list
   !
   type domain
      integer,      dimension(3)    ::  cells
      real(rkind),  dimension(3)    ::  inv_cells
      integer                       ::  ncells
      integer                       ::  neigh_cells
   end type domain
   !
   type neighbors
      integer, dimension(:), pointer :: neigh
   end type neighbors

   !
   type(linked_list),    allocatable,   dimension(:),   save    ::  linked
   type(domain),         allocatable,   dimension(:),   save    ::  div
   type(neighbors),      allocatable,   dimension(:),   save    ::  border
contains
   subroutine r_create_domain( box, sim, edge, par )
      implicit none
      type(simulation)  :: sim
      type(sim_box),    pointer :: box
      type(sim_par)             :: par
      real(rkind),  dimension(:),   intent(inout)   :: edge
      !
      ! Local variables
      integer :: i, k
```

```
error_label = 'error_alloc_dom'

if ( .not. sim%m_solute ) then
   allocate( div(1), stat = error_alloc )
   call r_error_alloc( error_label, error_alloc )

   box%m_cutoff_mol(1) = sim%m_rcut
   div(1)%cells = int(edge)

   if ( box%m_cutoff_mol(1) > edge(1) .or.&
      & box%m_cutoff_mol(1) > edge(2) .or.&
      & box%m_cutoff_mol(1) > edge(3) ) then
      write(*,*)' The cut-off value ', box%m_cutoff_mol(1),' exceded to
      stop
   end if
   !
   do
      if ( (edge(1)/div(1)%cells(1)) <= box%m_cutoff_mol(1) &
         & .and. (edge(2)/div(1)%cells(2)) <= box%m_cutoff_mol(1) &
         & .and. (edge(3)/div(1)%cells(3)) <= box%m_cutoff_mol(1)) then
         div(1)%cells = div(1)%cells - 1
         cycle
      else
         exit
      end if
   end do
   div(1)%inv_cells = div(1)%cells/edge
   div(1)%ncells = div(1)%cells(1) * div(1)%cells(2) * div(1)%cells(3)
   if( div(1)%ncells > max_mol ) then
      write(*,'(a)')' The number of division of box simulation exceded t
      write(*,'(a)')' Check the cut-off radius.'
      stop
   end if
   !
   select case(div(1)%ncells)
   case(1)
      div(1)%neigh_cells = 1
   case(8)
      div(1)%neigh_cells = 8
   case default
      div(1)%neigh_cells = 27
```

```
   end select
   !
end if
!
if ( sim%m_solute ) then
   allocate( div(par%cut_type), stat = error_alloc )
   call r_error_alloc( error_label, error_alloc )

   do k = 1, par%cut_type
      div(k)%cells = int(edge)
   end do

   do k = 1, par%cut_type
      if ( box%m_cutoff_mol(k) > edge(1) .or.&
         & box%m_cutoff_mol(k) > edge(2) .or.&
         & box%m_cutoff_mol(k) > edge(3) ) then
         write(*,*)' The cut-off value ', box%m_cutoff_mol(k),' exceeded
         stop
      end if
   end do
   !
   do k = 1, par%cut_type
      do
         if( (edge(1)/div(k)%cells(1) <= box%m_cutoff_mol(k)) &
            & .and. (edge(2)/div(k)%cells(2) <= box%m_cutoff_mol(k)) &
            & .and. (edge(3)/div(k)%cells(3) <= box%m_cutoff_mol(k)) ) t
            div(k)%cells = div(k)%cells - 1
            cycle
         else
            exit
         end if
      end do
      div(k)%inv_cells = div(k)%cells/edge
      div(k)%ncells = div(k)%cells(1) * div(k)%cells(2) * div(k)%cells(3
      if( div(1)%ncells > max_mol ) then
         write(*,'(a)')' The number of division of box simulation excede
         write(*,'(a)')' Check the cut-off radii.'
         stop
      end if
      !
      select case(div(k)%ncells)
      case(1)
```

```
            div(k)%neigh_cells = 1
         case(8)
            div(k)%neigh_cells = 8
         case default
            div(k)%neigh_cells = 27
         end select
         !
      end do
   end if
end subroutine r_create_domain
!
```

Apêndice H

```
module m_interaction
   use m_kind
   use m_simtype
   use m_boxtype
   use m_precells
   use m_pbcs
   use m_cutoffs
   !
contains
   subroutine r_interaction( com, edge, box, sim, par, i, id )
      implicit none
      type(simulation)                        :: sim
      type(sim_box), pointer                  :: box
      type(sim_par)                           :: par
      real(rkind), dimension(:), intent(inout) :: com
      real(rkind), dimension(:), intent(in) :: edge
      integer, intent(in)                     :: id, i
      !
      ! Local variables
      real(rkind)               :: rix, riy, riz
      real(rkind)               :: rjx, rjy, rjz, r2ij
      real(rkind), dimension(3) :: rij
      integer                   :: idcell, j, k, iter
      !

      rix = com(1); riy = com(2); riz = com(3)

      if ( .not. sim%m_solute ) then
         idcell = f_idcell( com, edge, sim, 0 )
         call r_create_neigh( idcell, sim, par, 0 )
      end if

      if( sim%m_solute ) then
         idcell = f_idcell( com, edge, sim,  id )
         call r_create_neigh( idcell, sim, par, id )
      end if

      if ( .not. sim%m_solute ) then
         do k = 1, div(1)%neigh_cells
```

```
      iter = border(1)%neigh(k)
      j = linked(1)%head(iter)
      !
      do
         if ( j == 0 ) exit

         if ( j /= i ) then

            rjx = box%m_com_xyz(1,j)
            rjy = box%m_com_xyz(2,j)
            rjz = box%m_com_xyz(3,j)
            !
            rij(1) = rix - rjx
            rij(2) = riy - rjy
            rij(3) = riz - rjz

            if ( sim%m_pbcs ) then
               call r_pbc( rij, edge )
            end if
            !
            r2ij = rij(1) * rij(1) + rij(2) * rij(2) + rij(3) * rij(3)

            if ( r2ij <= cut(1)%rcut_pair ) then
            end if
         end if

         j = linked(1)%object(j)
     end do
   end do
end if
!
if ( sim%m_solute ) then
   do k = 1, div(id)%neigh_cells

      iter = border(id)%neigh(k)
      j = linked(id)%head(iter)
      !
      do
         if ( j == 0 ) exit

         if ( j /= i ) then
```

```
                    rjx = box%m_com_xyz(1,j)
                    rjy = box%m_com_xyz(2,j)
                    rjz = box%m_com_xyz(3,j)
                    !
                    rij(1) = rix - rjx
                    rij(2) = riy - rjy
                    rij(3) = riz - rjz

                    r2ij = rij(1) * rij(1) + rij(2) * rij(2) + rij(3) * rij(3)

                    if( r2ij <= cut(id)%rcut_pair ) then
                    end if
                 end if

                 j = linked(id)%object(j)
              end do
           end do

        end if

     end subroutine r_interaction
  end module m_interaction
```

Apêndice I

```
 module m_markov
   use m_kind
   use m_boxtype
   use m_simtype
   use m_random
   use m_interaction
   use m_pbcs
   use m_rotation
   !
contains
   subroutine r_markov( box, sim, par )
      implicit none
      type(sim_box),    pointer :: box
      type(sim_par)             :: par
      type(simulation)          :: sim
      !
      ! Local variables
      real(rkind),  dimension(3)    :: com_old
      real(rkind),  dimension(3)    :: com_new
      real(rkind)                   :: xi
      integer                       :: imol = 1
      real(rkind),  dimension(3)    :: edge
      integer :: id = 1, j

      do
         xi    = m_rand()
         imol  = int(max_mol * xi ) + 1
         if ( imol > max_mol ) then
            cycle
         else
            exit
         end if
      end do

      com_old(1) = box%m_com_xyz(1,imol)
      com_old(2) = box%m_com_xyz(2,imol)
      com_old(3) = box%m_com_xyz(3,imol)

      edge(1) = box%m_edge(1)
      edge(2) = box%m_edge(2)
```

```
      edge(3) = box%m_edge(3)

      call r_interaction( com_old, edge, box, sim, par, imol, id )
      !
! Transformation matrix in R3
  com_new(1) = (com_old(1) + (2.0_rkind * m_rand() - 1.0_rkind) * sim%m_dr)
  com_new(2) = (com_old(2) + (2.0_rkind * m_rand() - 1.0_rkind) * sim%m_dr)
  com_new(3) = (com_old(3) + (2.0_rkind * m_rand() - 1.0_rkind) * sim%m_dr)

  if ( sim%m_pbcs ) then
   call r_pbc(com_new, edge)
  end if
      !
  call r_interaction( com_new, edge, box, sim, par, imol, id )

      j = 1

      call r_euler( box, sim, id, j )

   end subroutine r_markov
end module m_markov
```

Apêndice J

Procedimento para atualização de listas ligadas.

```
   subroutine r_update( com_new, i_d, a, y, x )
!
  implicit none
!
! !Input parameters:
  type(simulation), intent(inout) :: y
  type(box), pointer :: x
  integer,intent(in) :: a
  integer,intent(in) :: i_d
  real(rkind), dimension(:),intent(in) :: com_new
!
! !Revision history:
! 08Aug 2015 Asdrubal Lozada
!
!eop
!-----------------------------------------------------------------------
! Local variables
  real(rkind), dimension(3) :: com_old
  integer :: jold, jnew, i

  if( a == 1 ) then
    com_old(1) = x%m_cmass(1,i_d)
    com_old(2) = x%m_cmass(2,i_d)
    com_old(3) = x%m_cmass(3,i_d)

    x%m_cmass(1,i_d) = com_new(1)
    x%m_cmass(2,i_d) = com_new(2)
    x%m_cmass(3,i_d) = com_new(3)

    do i=1,x%m_ns(i_d)
     x%m_site(1,i,i_d) = x%m_rot(1,i)
     x%m_site(2,i,i_d) = x%m_rot(2,i)
     x%m_site(3,i,i_d) = x%m_rot(3,i)
    end do

    jold = f_idcell(com_old,y,x)
    jnew = f_idcell(com_new,y,x)

    if( jnew /= jold ) then
```

```
    call r_headinit(y,x)
    call r_linkscell(y,x)
  end if ! index cell

 else
   x%m_cmass(1,i_d) = x%m_rcom(1,i_d)
   x%m_cmass(2,i_d) = x%m_rcom(2,i_d)
   x%m_cmass(3,i_d) = x%m_rcom(3,i_d)

  do i=1,x%m_ns(i_d)
   x%m_site(1,i,i_d) = x%m_rsite(1,i)
   x%m_site(2,i,i_d) = x%m_rsite(2,i)
   x%m_site(3,i,i_d) = x%m_rsite(3,i)
  end do
end if ! a

end subroutine r_update
```