UNIVERSIDADE FEDERAL DE JUIZ DE FORA

RENAN SILVEIRA SENA

SIMULAÇÃO COMPUTACIONAL DE LÍQUIDOS IÔNICOS

JUIZ DE FORA, 2017

RENAN SILVEIRA SENA

SIMULAÇÃO COMPUTACIONAL DE LÍQUIDOS IÔNICOS

Relatório de conclusão de iniciação científica apresentado a Universidade Federal de Juiz de fora como parte das exigências ao fim da iniciação científica.

Orientador: Prof. Sócrates de Oliveira Dantas

JUIZ DE FORA, 2017

Sena Renan Silveira
Simulação Computacional de Líquidos Iônicos/ Renan Silveira Sena -2017
17F: il.

Orientador: Sócrates de Oliveira Dantas

Relatório de Conclusão de Iniciação Científica (graduação) – Universidade Federal de Juiz de Fora, área de Física, 2017.

1. Líquidos Iônicos. 2. Simulação Computacional 3. Programação

SUMÁRIO

1 Introdução à Líquidos Iônicos (LI)

Os Líquidos Iônicos (Lis) são geralmente definidos como uma classe de sais orgânicos com um ponto de fusão até 100°C. Em sua composição possuem cátions orgânicos de baixa simetria e ânions orgânicos e inorgânicos, sendo assim amplamente estudados e buscando diversas propriedades físico-químicas desejadas.

Um dos principais interesses científicos nos líquidos iônicos baseia-se nas suas propriedades, como a baixa pressão de vapor, elevada estabilidade térmica e química, elevada condutividade iônica, entre outras propriedades. Uma das características mais relevantes dos LIs relaciona-se com a possibilidade de se conseguir modelar diferentes propriedades físicas, térmicas e químicas de acordo com a combinação adequada de cátion e ânion.

A área de pesquisa dos líquidos iônicos vem crescendo bastante nas últimas três décadas, porém existem referências a sais orgânicos de baixo ponto de fusão são enquadrados na definição de líquidos iônicos desde o século XX. Nesse contexto uma das primeiras referências surgiu em 1914 por P. Walden reportando um sal de nitrato de etilamônio com um ponto de fusão de 12°C e mais tarde em 1951 por Hurley e colaboradores apresentando sais de cloraluminatos de n-alquilpiridínio de baixo ponto de fusão.

Nessas últimas três décadas a evolução dos LIs pode ser divida em três fases de acordo com o potencial e interesse das suas aplicações finais. Na primeira fase os LIs foram preparados com o objetivo de serem usados como solventes alternativos, ou solventes verdes no lugar dos solventes convencionais (voláteis e tóxicos). Na segunda fase os LIs foram usados como materiais avançados para aplicações específicas nas áreas de engenharia química e ciência dos materiais. Na terceira e mais recente fase os LIs foram associados a áreas biológicas com potencial de aplicação na área de bioquímica e da farmacêutica.

As citações sobre Lis vem crescendo nos meios acadêmicos, através do seu elevado número de publicações nas últimas décadas ao contrário das aplicações a níveis industriais que ainda são reduzidas, existindo algumas aplicações que têm sido exploradas.

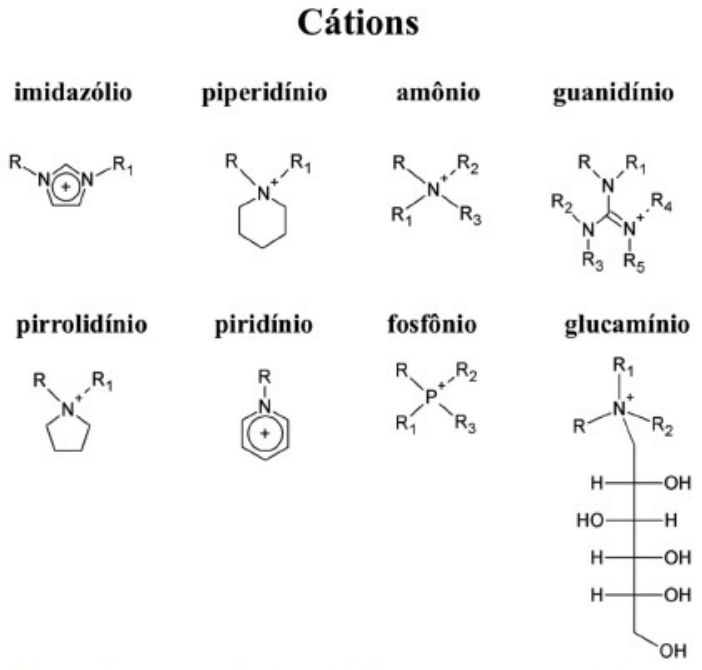


Figura 1. *Estrutura dos cátions mais utilizados no preparo de líquidos iônicos. Reimpresso de Ho. T. D.; Zhang, C.; Hantao, L. W.; Anderson, J. L. Anal. Chem.* ***2014*** *86 262. Copyright 2014 American Chemical Society*

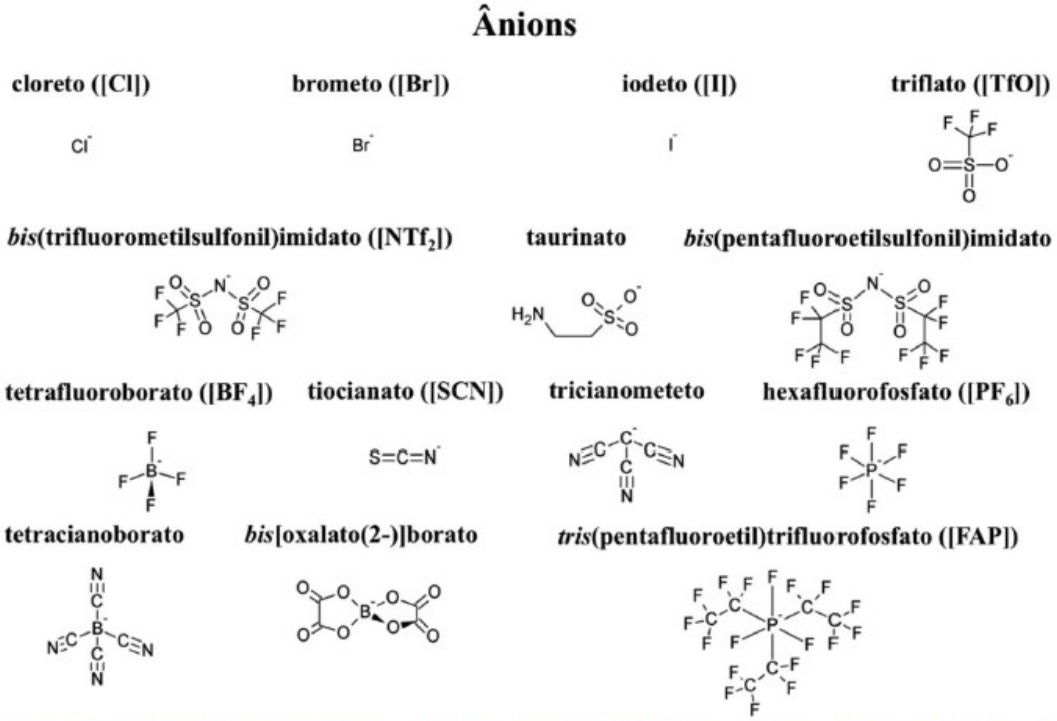


Figura 2. *Estrutura dos ânions mais utilizados no preparo de líquidos iônicos. Reimpresso de Ho. T. D.; Zhang, C.; Hantao, L. W.; Anderson, J. L.; Anal. Chem.* ***2014****, 86, 262. Copyright 2014 American Chemical Society*

2 Dinâmica Molecular(DM)

2.1 Introdução

É uma técnica computacional de resolver as equações de movimento de átomos e moléculas que interagem com os demais átomos ou moléculas da amostra(sistema), possivelmente com campos externos, e que o sistema apresente condições iniciais e de contorno.

À medida que os átomos vão variando seus estados, caracterizados pelas posições e velocidades, calculam-se as variáveis dinâmicas relevantes do sistema, como Temperatura ou densidade de energia, em função da posição, entre outras variáveis.

O método de dinâmica molecular foi originalmente desenvolvido na área da física teórica no final da década de 1950 mas atualmente é aplicada sobretudo a áreas como a químico-física, ciência dos materiais e à modelação de biomoléculas.

Como os sistemas moleculares possuem um vasto número de partículas, é impossível determinar analiticamente as propriedades de sistemas complexos. As simulações de DM contornam este problema pelo recurso a métodos numéricos. No entanto, simulações longas de DM tornam-se mal-acondicionadas, gerando erros cumulativo de integração numérica que podem ser minimizados pela escolha de parâmetros e algoritmos apropriados, mas estes erros não podem ser completamente eliminados.

A DM também é apelidada de "mecânica estatística numérica" e "Visão de Laplace da mecânica Newtoniana" para a previsão do futuro por animação das forças da natureza permitindo a visualização do movimento molecular à escala atômica.

É tentador, embora não inteiramente preciso, descrever a técnica como um "microscópio virtual" com alta resolução temporal e espacial, mas com vantagem de ver fenômenos físicos em escalas muito mais rápidas do que o olho humano pode perceber, como femtossegundos, ou mesmo muito menor do que microscópios podem ver, na escala de amgstrom; ou mesmo movimento de receptores em células dos seres vivos ocorrem em femtossegundos.

Escalas de tempo dos períodos de movimentos moleculares [7]

vibração de ligações químicas	10^{-14} a 10^{-13}s
rotação de cadeias laterais (à superfície da proteína)	10^{-11} a 10^{-10}s
movimentos do tipo "dobradiça" de pedaços da cadeia polipéptidica	10^{-11} a 10^{-7} s
transições alostéreas e desnaturação local	10^{-5} a 10 s

2.2 Etapas da Simulação por Dinâmica Molecular

Os cálculos de sistemas com centenas ou até milhares de átomos passam por alguns passos para a completa modelagem do sistema, como descrito a seguir:

1) Gerar as configurações iniciais, que podem ser obtidas ou geradas.

2) Cálculo das forças exercidas sobre cada partícula devido a interações interatômicas. Esta etapa é realizada em um campo de força.

3) Otimização da estrutura, realizada através de algoritmos.

4) Dinâmica da estrutura, realizada através da integração das equações de movimento de Newton, por métodos numéricos.

5) Análise dos resultados através das propriedades de equilíbrio.

Num sistema molecular é importante calcular a sua energia potencial. A energia potencial calculada através de uma função, essa função apresenta diversos termos com significados físicos mais ou menos preciso, em nosso estudo de dinâmica molecular

usamos a função de energia potencial abaixo.

$$V(\mathbf{r}_1,\ldots,\mathbf{r}_{Nat}) = \sum_{n=1}^{N_b} \frac{1}{2} K_{b_n}(b_n - b_{0_n})^2 + \sum_{n=1}^{N_\theta} \frac{1}{2} K_{\theta_n}(\theta_n - \theta_{0_n})^2 + \sum_{n=1}^{N_\xi} \frac{1}{2} K_{\xi_n}(\xi_n - \xi_{0_n})^2 +$$

$$+ \sum_{n=1}^{N_\phi} \left[1 + \cos(n_{n'}\phi_{n'} - \delta_{n'})\right] + \sum_{i<j}^{N_a} \left[\frac{C_{12}(i,j)}{r_{ij}^{12}} - \frac{C_6(i,j)}{r_{ij}^{6}} + \frac{q_i q_j}{4\pi\varepsilon_0 \varepsilon_r r_{ij}} \right]$$

Onde V é o potencial e essa função possui vários termos que serão explicados abaixo:

$$\sum_{n=1}^{N_b} \frac{1}{2} K_{b_n}(b_n - b_{0_n})^2$$

Esse termo representa a soma dos potenciais de todas as Nb ligações covalentes do sistema. A força que atua em cada átomo da ligação covalente pode ser modelada como na lei de Hooke como mostra a figura. Onde bn é a distância entre os átomos, Kbn é uma constante do tipo de ligação, b0n é a distância de equilíbrio.

$$\sum_{n=1}^{N_\theta} \frac{1}{2} K_{\theta_n}(\theta_n - \theta_{0_n})^2,$$

Esse termo de potencial possui características muito semelhante ao termo de ligação covalente pode ser aplicado aos Nθ ângulos de ligação do sistema, onde Kθn é a constante de força, θ0n o ângulo de equilíbrio e o ângulo θn formado entre três átomos.

$$\sum_{n=1}^{N_\xi} \frac{1}{2} K_{\xi_n}(\xi_n - \xi_{0_n})^2,$$

Esse termo de potencial pode ser aplicado aos ângulos diedros (impróprios), isto é, as interações do tipo ligante definidas entre quatro átomos. Existem dois tipos de diedros os próprios(geometria sem transição) e os impróprios(geometria com transição). Onde Kξn é constante de força, ξ0n é o ângulo de equilíbrio.

Esse tipo de diedros serve, por exemplo para definir quiralidades em tetraedros ou planaridade em anéis, tal como nas figuras abaixo:

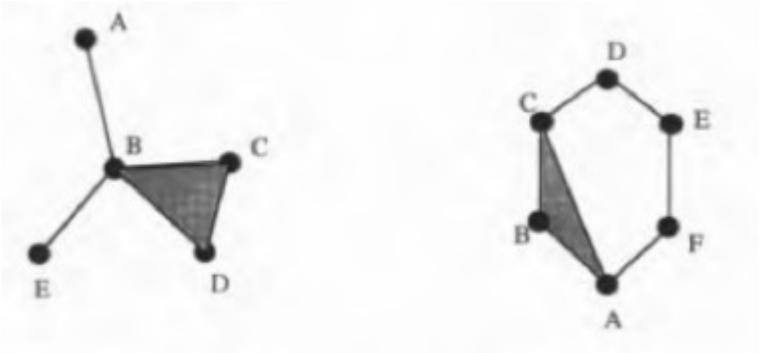


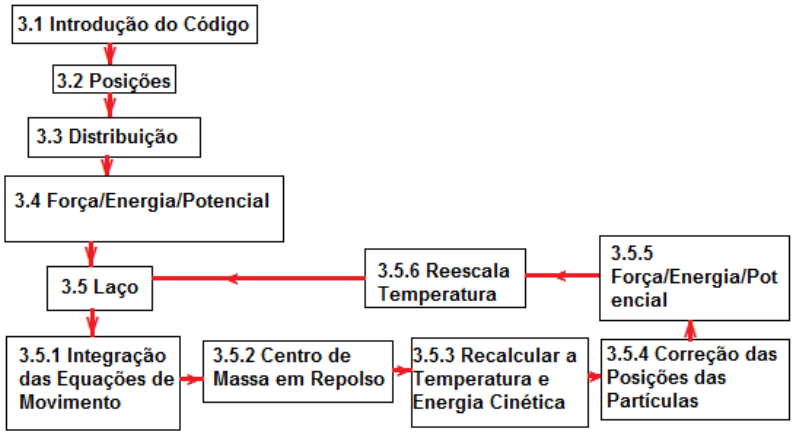
Fig. 1 - **Uso de diedros impróprios. No caso da esquerda, o potencial de diedro impróprio serve para manter o ângulo ξ_n formado pela ligação AB com o plano BCD (o mesmo se aplicada a BE). Assim a quiralidade é mantida. No caso da direita, ilustra-se o uso do potencial de diedros impróprios para manter a ligação CD no plano ABC. Neste caso o ângulo de equilíbrio será zero para manter a planaridade.**

$$\sum_{n'=1}^{N_\phi} \left[1 + \cos(n_{n'}\phi_{n'} - \delta_{n'})\right].$$

Esse termo de potencial pode ser aplicado aos ângulos diedros (próprios), onde se aplica a todos os Nϕ diedros do sistema, contém uma constante de força Kϕn (somente para além do ângulo de equilíbrio ∂n, existe também um outro termo nn que é designado por multiplicidade do diedro.

3 O Código de Dinâmica Molecular em Python

Realizamos uma simulação computacional de átomos de Argônio em um sistema fechado. O programa segue o fluxograma abaixo.



3.1 Introdução do Código

```
# -*- coding: latin-1 -*-
import math                              #IREMOS USAR ESSAS BIBLIOTECAS NO CÓDIGO
import numpy as np
#import pylab as plote                   #CASO QUEIRA PLOTAR GRÁFICOS, USE pylab


TABELA1 = open ('configura.txt', 'w') #Tabela 1 contendo colunas: Elemento qui,
                                         # posições e numero de indices
TABELA2 = open ('proprieda.txt', 'w') #Tabela 2 contendo colunas: energia po-
                                         # têncial, temperatura e energia cinética
                                         # retornar tabela 1 para .xyz e tabela 2
                                         #para .dat para uso em programas como o
                                         #VMD

TABELA2.writelines(['#Elem\t' , 'X\t' , 'Y\t' , 'Z\n'])
TABELA2.writelines(['#ENER. POTENCIAL(eV)\t' , 'ENER. CINÉTICA(eV)\t' ,
'TEMPERATURA(K)\n'])
                                         #Acima temos a escrita dos pilares das
                                         #informações que serão preenchidas nas
                                         #tabelas

print("PROGRAMA DE DINÃMICA ÃTOMICA\n")  #O usuário colocará as informações
print("VALORES DEFINIDOS:\n")                # Iniciais do Sistema
print("E=0.997 kJ/mol , s=3.4 A, V0 definida pela temperatura inicial\n")
print("Distancia de equilibrio: 3.81 A \n")
                                          #Dados usados na Equação de Lennard-
                                          #Jones
```

$$V(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right]$$

```
# Onde V(r) Potêncial de Lennard-Jones
#Ɛ é o poço de potencial
#σ é a distância de equilíbrio
#r distância entre os átomos

num = int(input("Informe o numero de particulas do sistema:\n"))
T = float(input("Informe a temperatura em Kelvin:\n"))
tmax = float(input("Informe o tempo maximo de simulacao(tmax):\n"))
dt = float(input("Informe o passo (dt) :\n"))
L = float(input("Informe o comprimento da caixa:\n"))

Talvo = T                                 # temperatura ideal
r = np.float(3.81e0)                      #distancia de equilibrio
rc = r * np.float(2.5e0)                  #Raio de corte
aresta_cubo = L * rc

#Rmax = float(input("Entre com Rmax:\n"))  #CASO QUEIRA PLOTAR GRÁFICOS
#dr = float(input("Entre com dr:\n"))
```

3.2 Posições

```
r0 = posicoes(num,aresta_cubo)  #Subrotína de distribuição dos átomos na caixa

def posicoes(num,aresta_cubo):
    l = aresta_cubo / math.pow(num,(1.0/3.0))
    nterco = np.int(math.pow(num,(1.0/3.0)))
    if num != np.int(math.pow(nterco,3)):
       nterco = np.int(nterco) + np.int(1)
    r0 = np.zeros((num,3),dtype=float)
    npart = 0
    for n in range(nterco):
        for m in range(nterco):
            for w in range(nterco):
                r0[npart][0] = np.float(n) * l
                r0[npart][1] = np.float(m) * l
                r0[npart][2] = np.float(w) * l
                npart = npart + 1

     return r0
```

```
aresta_cubo2 = aresta_cubo / np.float(2.0e0)

TABELA1.writelines([str(num), '\n', '\n'])
for i in range (num):
    TABELA1.writelines(['Ar  \t' , str(r0[i][0]), '\t' , str(r0[i][1]), '\t' ,
str(r0[i][2]), '\n' ])
                            #Acima está sendo escrito na tabela 1 as posições
                            #de cada átomo

t=0

uma = 1.66e-27              #unidade de massa atômica
massa = 39.94 * uma         #massa do átomo de argônio

K = 3 / 2 * num / 6.022e23 * 8.31 * T
                            #K é a energia cinética
                            # num/6.022e23 = numero de moles
                            #8.31 J/mol*K e T = K

v0 = distribuicao_aleatoria(massa,num,K)
```

3.3 Distribuição

```
def distribuicao_aleatoria(massa,num,K):
                                          #Calcula a velocidade inicial
                                          # das moléculas
    V0 = math.sqrt(2 * K / (num * massa))  #Apatir da energia cinética K,
                                          #da massa e do numéro de particulas


   v0 = np.random.random((num,3)) - 0.5e0   #normaliza vetores

    for i in range(num):
        modulo = np.sqrt(v0[i][0]*v0[i][0] + v0[i][1]*v0[i][1] +
        v0[i][2]*v0[i][2])
        if modulo != 0.0e0:
           v0[i] = v0[i] / modulo
    v0 = v0 * V0
    return v0
```

3.4 Força/Energia/Potencial

```
  #chama rotina para calcular forcas e energia potencial do sistema
  F, U = forcas_energia_pot(r0,aresta_cubo,rc,num)

  def forcas_energia_pot(r0,aresta,corte,num):
      E = np.float(0.997e0)                    # E da equacao de Lennard-Jones
                                               #em Kj/mol
      s = np.float(3.4e0)                      # sigma da equaÃ§Ã£o de
                                               #Lennard-Jones
      U = np.float(0.0e0)                      #energia potencial

      F = np.zeros((num,3),float)              #Força
      unitario = np.zeros((3),float)           #Vetor de zeros que se tornará
      for i in range(num-1):                   #um vetor unitário
          for j in range (i+1,num):
              rx = r0[j][0] - r0[i][0]         #Calcula a norma ou Distância
              if np.abs(rx) > (aresta/2.0e0):  #entre ascpartículas
                 rx = np.abs(rx) - aresta
              ry = r0[j][1] - r0[i][1]
              if np.abs(ry) > (aresta/2.0e0):
                 ry = np.abs(ry) - aresta
              rz = r0[j][2] - r0[i][2]
              if np.abs(rz) > (aresta/2.0e0):
                 rz = np.abs(rz) - aresta
              r = math.sqrt(rx*rx + ry*ry + rz*rz) # distancia entre 2 atomos,
              unitario[0] = rx / r                 # dada em A[Angtrom] = e-10 m
              unitario[1] = ry / r                 #vetor unitário
              unitario[2] = rz / r
              if r <= corte:                       #raio de corte
                 f1 = s / r                        # 3.4/[A ou e-10m]
                 f2 = math.pow(f1,12) - math.pow(f1,6)
                 #Lennard-Jones
                 f3 = np.float(48.0e0) * (E / s) * (math.pow(f1,13) -
                 np.float(0.5e0) * math.pow(f1,7))
                 #Derivada da equação da energia Potencial
                 U = U + np.float(4.0e0) * E * f2 * 1.04e-2
                 #Energia Potencial total do processo,e a soma de todas as
                 #energias potenciais (em eV)
                 F[i] = F[i] + f3 * unitario * 1.0e-10 / 6.022
                 #Calculos de forças
                 F[j] = F[j] - f3 * unitario * 1.0e-10 / 6.022
```

```
    return F, U
```

Saindo da sub-rotina temos

```
#determina o numero de iteracoeses totais, tendo conhecimento do intervalo de
#tempo e do tempo maximo
itmax = int (tmax/dt)
dt = dt * np.float(1.0e-15) #em fs ou e^(-15)s
dt2 = dt * dt
mdt2 = 0.5 * dt2
numesc = 10
```

3.5 Laço

Agora o programa entrará em um laço, chamando diversas sub-rotinas novas e já vistas antes nesse artigo

```
#laço da dinÃ¢mica molecular
for i in range(itmax):
    #integra as equacoes de movimento
    r0, v = integracao_das_equacoes_de_movimento(F,r0,v0,massa,dt,mdt2)

    #agora vamos fazer o centro de massa do sistema ficar em repouso
    v0 = centro_de_massa_em_repolso(v,num,massa)

    #Recalcular a temperatura e energia cinética
    T, C = recalcular_temperatura(v0,num,massa)
    C = C * 6.242e+18

    #verifica se a particula sai do cubo e faz correcoes
    r0 = correcao_posicao_particulas(num,r0,aresta_cubo)

    #chama rotina para calcular forcas e energia potencial do sistema
    F, U = forcas_energia_pot(r0,aresta_cubo,rc,num)

    #reescala velocidades caso temperatura atual seja distinta da temperatura
    #alvo
    v0 = reescala_veloci(v0,Talvo,T)

    esc = int(i / numesc)
    if ((i - esc*numesc) == 0):
       TABELA2.writelines([str(i), '\t', str(U) , '\t' , str(C) , '\t\t' ,
       str(T) , '\n'])
       TABELA1.writelines([str(num), '\n', '\n'])
       for j in range(num):
           TABELA1.writelines(['Ar  \t' , str(r0[j][0]) , '\t' ,
           str(r0[j][1]) , '\t' , str(r0[j][2]) , '\n' ])

TABELA1.close()
TABELA2.close()
```

3.5.1 Integração das Equações de Movimento

```
def integracao_das_equacoes_de_movimento(F,r0,v0,massa,dt,mdt2):
    v = np.zeros((num,3),float)
    r = np.zeros((num,3),float) #Nessa subrotina o programa vai calcular as
                                #novas posições das partículas e velocidades

    for j in range(num):
        r[j] = r0[j] + (v0[j] * dt + F[j] * mdt2 / massa) * np.float(1.0e10)
        v[j] = v0[j] + F[j] * dt / massa

    return r, v
```

3.5.2 Centro de Massa em Repouso

```
def centro_de_massa_em_repolso(v, num , massa):
#Como um programa computacional pode apresentar pequenos erros nos
#calculos esses erros podem se propagar levando a situações não
#desejadas como por exemplo deslocar o centro de massa.
#numa experiência real, o centro de massa está em repolso.
#Então temos que ajustar o centro de massa de forma que ele fique
#em repolso

    pcm = np.zeros((3),float) #momento linear do centro de massa

    for j in range(num):
        pcm = pcm + v[j] * massa

    vcm = pcm / (num * massa)

    for j in range(num):
        v[j] = v[j] - vcm

    v0 = v

    return v0
```

3.5.3 Recalcular a Temperatura e Energia Cinética

```
def recalcular_temperatura(v0,num,massa):
    K = np.float(0.0e0)
    Rb = np.float(1.38064852e-23)    #constante de Boltzmann =
                                      #=1,38064852x10-23 (m2 kg s-2 K-1)

    for i in range(num):
        V2 = math.sqrt (v0[i][0] * v0[i][0] + v0[i][1] * v0[i][1] +
        v0[i][2] * v0[i][2] )              #velocidade em modulo de cada átomo
        K = K + 1.0e0/2.0e0 * massa * V2   #energia cinética de cada partícula
    T = 2 * K /(3 * num * Rb)              #recalcula a temperatura
    return T, K
```

3.5.4 Correção das Posições das Partículas

```
def correcao_posicao_particulas(num, r0, aresta_cubo):
#Essa subrotina tem a função de não deixar que a partícula
#saia do cubo de analíse do programa, fazendo com que a
#posição da partícula que saiu do cubo seja corrigida para
#uma posição relativa dela no cubo, ou seja as partículas
#que saem, entram simultâneas sem alterar propriedades
#do sistema
    for j in range (num):
        if r0[j][0] >= aresta_cubo:
            r0[j][0] = r0[j][0] - aresta_cubo
        else:
            if r0[j][0] < 0.0e0:
               r0[j][0] = r0[j][0] + aresta_cubo

        if r0[j][1] >= aresta_cubo:
            r0[j][1] = r0[j][1] - aresta_cubo
        else:
            if r0[j][1] < 0.0e0:
               r0[j][1] = r0[j][1] + aresta_cubo

        if r0[j][2] >= aresta_cubo:
            r0[j][2] = r0[j][2] - aresta_cubo
        else:
            if r0[j][2] < 0.0e0:
               r0[j][2] = r0[j][2] + aresta_cubo
    return r0
```

3.5.5 Força/Energia/Potencial

É a mesma sub-rotina do item 3.4

3.5.6 Reescala Temperatura

```
#reescala velocidades caso temperatura atual seja distinta da temperatura alvo
def reescala_veloci(v0,Talvo,T):
    v0 = v0 * np.sqrt(Talvo/T)
    return v0
```

4 Simulação de uma caixa de água usando Moltemplate e Lammps

Esse exemplo foi estudado para a compreensão do Moltemplate e Lammps que são auxiliadores muito importantes na simulação computacional.

```
#
#    H1     H2
#      \   /
#        O
#

SPCE {
# Propriedades do átomo e topologia molecular vão nas várias
#seções "Data ..."
# Nós selecionamos "atom_style full". Isso significa que
# usamos esse formato de coluna: 13


# atomID     molID      atomType    charge    coordX     coordY    coordZ
write("Data Atoms") {
$atom:o      $mol:.       @atom:O  -0.8476  0.0000000   0.000000   0.00000
$atom:h1     $mol:.       @atom:H   0.4238  0.8164904   0.5773590  0.00000
$atom:h2     $mol:.       @atom:H   0.4238  -0.8164904  0.5773590  0.00000
}


# As variáveis que começam com $ ou @ serão substituídas por números
# que o LAMMPS irá
# Eventualmente ler. Cada um dos três átomos "será atribuído
# atomIDs (denotado aqui por "$ átomo: o", "$ átomo: h1", "$ átomo: h2"),
# mesmo se eles pertencem a diferentes moléculas. No entanto, os tipos
#de átomos (designado por  "@atom:O", "@atom:H") são partilhados para
#átomos em todas as moléculas. Todos os 3 átomos compartilham o mesmo
# número de modelo (representado aqui por "$ mol :.")
# Porém esse número é diferente para diferentes moléculas de água.


write_once("Data Masses") {
# átomo    massa
@atom:O    15.9994
@atom:H    1.008
}
write("Data Bonds") {
#ligaçãoID  tipo de ligação  atomID1  atomID2
$bond:oh1      @bond:OH      $atom:o  $atom:h1
$bond:oh2      @bond:OH      $atom:o  $atom:h2
}
write("Data Angles") {
# anguloID  tipo de angulo     atomID1    atomID2 atomID3
$angle:hoh   @angle:HOH       $atom:h1    $atom:o $atom:h2
}
# --- Os parâmetros de campo de força vão na seção "In Settings": ---
write_once("In Settings") {
# -- interações não ligantes (Par) --
#          tipo 1 de átomo   tipo 2 de átomo  lista de parame. (epsilon, sigma)
pair_coeff  @atom:O           @atom:O           0.1553 3.166
pair_coeff  @atom:H           @atom:H           0.0    2.058
```

```
# (Regras de mistura determinam interações entre os tipos @atom: O e @atom: H)
# -- Interações Ligantes --
#             tipo de ligação   lista de parâmetros (k_bond, r0)
bond_coeff   @bond:OH             1000.00 1.0
#             tipo de ângulo    lista de parâmetros (k_theta, theta0)
angle_coeff  @angle:HOH           1000.0   109.47

# Definições de grupo e restrições também podem ir na seção "In Settings"
group spce type  @atom:O  @atom:H
fix fSHAKE spce shake 0.0001 10 100 b @bond:OH a @angle:HOH
# (Lammps quirk: Lembre-se de "unfix fSHAKE" durante a minimização.)
}
# LAMMPS suporta um grande número de estilos de força-campo. Devemos selecionar
# Quais nós precisamos. Essas informações pertencem à seção "In Init".
write_once("In Init") {
units        real                 # angstroms, kCal/mole, Daltons, Kelvin
atom_style   full                 40/5000 # Escolha um formato de coluna para
                                          # a secção átomo
pair_style   lj/charmm/coul/long 9.0 10.0 10  # Parâmetros: epsilon sigma
bond_style   harmonic             # Parâmetros: k_bond, r0
angle_style  harmonic             # Parâmetros: k_theta, theta0
kspace_style pppm 0.0001          # Método de soma eletrostática
                                  # de longo alcance
pair_modify  mix arithmetic       # Usando as regras de mistura
                                  # de Lorenz-Berthelot
}
```

5 Referências

1 – SILVA, THIAGO B, LÍQUIDOS IÔNICOS – ALGUNS ASPECTOS SOBRE AS PROPRIEDADES, PREPARAÇÃO E APLICAÇÕES, 2004

2 – BRANCO, LUÍS C., LÍQUIDOS IÔNICOS – APLICAÇÕES E PERSPECTIVAS FUTURAS, 2015

3 – SOARES, CLÁUDIO M. - SIMULAÇÃO DE PROTEÍNAS USANDO MÉTODOS DE MECÂNICA/DINÂMICA MOLECULAR

4 – RAPORTE, DENNIS – ART OF MOLECULAR DYNAMICS SIMULATION, 2004