

**Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Química**

Frederico Teixeira Silva

**Análise de algoritmos eficientes para uso em dinâmica
molecular *ab initio* direta**

**Belo Horizonte
2014**

UFMG/ICEx/DQ 1030^a
D 563^a

Frederico Teixeira Silva

Análise de algoritmos eficientes para uso em dinâmica molecular *ab initio* direta

Analysis of efficient algorithms for use in direct *ab initio* molecular dynamics

Dissertação apresentada ao Departamento de Química do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais com requisito parcial para a obtenção do grau de Mestre em Química – Físico Química.

**Belo Horizonte
2014**

S586a Silva, Frederico Teixeira
2014 Análise de algoritmos eficientes para uso em
D dinâmica molecular ab initio direta [manuscrito] /
Frederico Teixeira Silva. 2014.
[xv], 149 f. : il.

Orientador: Jadson Cláudio Belchior.

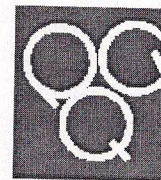
Dissertação (mestrado) - Universidade Federal de
Minas Gerais - Departamento de Química.
Inclui bibliografia.

1. Físico-química - Teses 2. Algoritmos - Teses 3.
Dinâmica molecular - Teses 4. Métodos de simulação -
Teses I. Belchior, Jadson Cláudio, Orientador II.
Título.

CDU 043

UFMG

PROGRAMA DE PÓS-GRADUAÇÃO EM QUÍMICA
DEPARTAMENTO DE QUÍMICA - ICEX
31270-901 - BELO HORIZONTE - MG
TEL.: 031 - 3409-5732
FAX: 031 - 3409-5711
E-MAIL: pgquimic@qui.ufmg.br

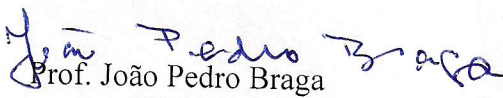


"Análise de Algoritmos Eficientes Para Uso em Dinâmica Molecular Ab Initio
Direta"

Frederico Teixeira Silva

Dissertação aprovada pela banca examinadora constituída pelos Professores:


Prof. Jadson Cláudio Belchior - Orientador
UFMG


Prof. João Pedro Braga
UFMG


Profa. Rita de Cássia de Oliveira Sebastião
UFMG

Belo Horizonte, 31 de julho de 2014.

Agradecimentos

A Deus.

Aos meus pais, Dalva e Eduardo. Pois contribuíram direta ou indiretamente na minha formação como pessoa.

Ao CNPq pelo auxílio financeiro. Se não fosse por essa instituição, nada disso seria possível.

Ao professor Jadson Cláudio Belchior, que há anos contribui na minha formação científica.

Aos demais professores da UFMG, pois foram responsáveis pela minha formação acadêmica.

À Lena, que foi compreensiva quando eu precisei de tempo para trabalhar.

Ao grupo de RPG Tormenta. Foram amigos importantes durante esse período.

Ao Flávio Haueisen pela consultoria gratuita na área de programação.

Ao laboratório de pesquisa LDAM, pelas várias contribuições, pequenas e grandes.

Aos demais colegas e amigos pelas críticas e apoio.

À FAPEMIG pelos recursos financeiros.

“Não entre em pânico”.

- Douglas Adams.

Resumo

Um software que integra as equações de movimento clássicas foi desenvolvido. O programa, de nome DRKAI (Dinâmica Runge Kutta *ab initio*) foi inicialmente testado em uma equação diferencial simples, a qual possui solução analítica, demonstrando grande precisão nesse caso. Verificada sua eficiência, foi aplicado em sistemas moleculares, abordagem conhecida como dinâmica molecular. Os testes da dinâmica resultaram em boa conservação de energia e preservação das características do sistema.

A segunda parte deste trabalho surgiu ao perceber que a dinâmica, na abordagem desejada, possui alto custo computacional, tornando imperativo o uso de algoritmos eficientes. Muitos algoritmos diferentes são usados em trabalhos atuais, tornando complexa a escolha do que deveria ser implementado. Nesta dissertação foi feito um estudo de cinco integradores. Dentre esses, os integradores Runge Kutta simétrico e Beeman-Verlet foram os mais eficientes, o algoritmo de Gauss Radau se mostrou inadequado para simulações longas e o algoritmo de Runge Kutta Gill revelou-se como o mais lento.

Palavras chave: Integração numérica, dinâmica molecular *ab initio*, análise de algoritmos.

Abstract

A software that integrates classical equations of motion was developed. The program named DRKAI (Dinâmica Runge Kutta *ab initio*) was first tested in a simple differential equation, which has analytical solution, showing great accuracy in this case. Verified its efficiency, DRKAI was applied in a molecular system, known as molecular dynamics approach. Dynamic results showed good energy conservation and preservation of major features of the system.

The second part of this paper came when high computational cost was perceived at desired approach. This made imperative the use of efficient algorithms. Many different algorithms are used in current works, making complex the choice of what should be implemented. In this thesis a study was made for five integrators. Among these, the symmetric Runge Kutta integrators and Beeman-Verlet were the most efficient, the algorithm of Gauss Radau proved inadequate for long simulations and the algorithm of Runge Kutta Gill proved to be the slowest.

Keywords: Numerical integration, *ab initio* molecular dynamics, algorithms analysis.

Índice de figuras

Figura 2.1 – Uma massa submetida a uma força de restauração com constante $k=1$ N/m.....	10
Figura 2.2 – Figura esquemática que apresenta uma sequência do integrador Gauss Radau.....	16
Figura 2.3 – Esquema de uma integração do tipo Runge Kutta.....	20
Figura 2.4 – Esquema de uma integrador do tipo Runge Kutta simétrico.....	21
Figura 4.1 - Fluxograma do software DRKAI. Mostra os principais passos da dinâmica molecular ab initio.....	36
Figura 4.2 – Representação do ponto de retorno clássico. Em ‘a’ têm-se o início da trajetória, em ‘b’ apresenta-se o ponto de retorno e em ‘c’ está representado alguns instantes após o ponto de retorno.....	39
Figura 4.3 – Resultado da integração numérica de duas partículas descritas por um potencial Morse realizando uma colisão frontal. Os parâmetros da simulação estão apresentados na tabela 4.2. O foco do gráfico está na região próxima ao ponto de retorno clássico.....	41
Figura 5.1 - Variação da energia em função do passo para cinco integradores diferentes. Estudo de uma partícula sob o efeito de uma força central.....	44
Figura 5.2. Variação da posição da partícula obtida pelos integradores em relação a posição obtida pela solução analítica da equação diferencial. Estudo de uma partícula sob o efeito de uma força central.....	46
Figura 5.3 - Variação da energia em função do passo para cinco integradores diferentes. Estudo da molécula de hidrogênio no nível MP2 e base TZV.....	48

Figura 5.4 - Corpo girando em torno de \mathbf{w} . \mathbf{L} é o momento angular, sabe-se que nem sempre esse aponta na direção de \mathbf{w}	50
Figura 5.5 – Equações químicas dos sistemas estudados neste trabalho.....	52
Figura 5.6 – Representação de um parâmetro de impacto (\mathbf{b}) genérico. É definido como a componente de \mathbf{R} perpendicular a \mathbf{v}	53
Figura 5.7 – Exemplo de condição inicial. Ataque nucleófilo de um átomo de cloro à molécula CH_3Br . O parâmetro de impacto foi definido como $0,5 \text{ \AA}$	54
Figura 5.8 – Dinâmica direta da reação $\text{S}_{\text{N}}2$: $\text{CH}_3\text{Cl} + \text{F}^-$. O átomo no centro do tetraedro é o carbono, a esfera maior ligada ao carbono é o cloro e o nucleófilo, inicialmente distante é o fluoreto. As figuras estão em sequência com uma diferença de 10 fs entre cada trajetória.....	58
Figura 5.9 – Dinâmica direta da reação $\text{S}_{\text{N}}2$: $\text{CH}_3\text{Br} + \text{F}^-$. O átomo no centro do tetraedro é o carbono, a esfera maior ligada ao carbono é o bromo e o nucleófilo, inicialmente distante é o fluoreto. As figuras estão em sequência com uma diferença de 10 fs entre cada trajetória.....	60
Figura 5.10 – Dinâmica direta da reação $\text{S}_{\text{N}}2$: $\text{CH}_3\text{Br} + \text{Cl}^-$. O átomo no centro do tetraedro é o carbono, a esfera maior ligada ao carbono é o bromo e o nucleófilo, inicialmente distante é o cloro. As figuras estão em sequência com uma diferença de 25 fs entre cada trajetória.....	62
Figura A.1 – Fluxograma do algoritmo genético ab initio desenvolvido no período de mestrado.....	67

Índice de tabelas

Tabela 2.1 - Primeiros passos de uma integração numérica com o método de Euler. Sistema massa-mola com constante $k= 1$ N/m e $m=1$ Kg. O passo (h) foi 0,1 s. Destaque em duas atualizações de posição e velocidade respectivamente.....	11
Tabela 2.2 - Primeiros passos de uma integração numérica. O sistema é uma partícula sob um potencial harmônico. Foi integrado via método de Euler.....	12
Tabela 2.3 - Coeficientes das equações (2.54-2.72).....	23
Tabela 4.1 - Índice dos integradores no software DRKAI (Dinâmica Runge Kutta ab initio).....	37
Tabela 4.2 – Parâmetros referentes ao potencial Morse (4.1) e condições iniciais para uma colisão frontal entre duas partículas que interagem por esse potencial.....	40
Tabela 5.1 - Informações necessárias para a obtenção das condições iniciais. O sistema é uma molécula de hidrogênio no nível MP2 e base TZV.....	47
Tabela 5.2 - Conservação do centro de massa e do momento angular para vários integradores. A energia cinética translacional foi definida como $3kT/2$, para uma temperatura de 300K ($1,4 \times 10^{-3}$ Hartree).....	54
Tabela 5.3 - Resultados de uma dinâmica molecular direta do sistema $CH_3Cl + F$. A base utilizada foi a 3-21g, com o método MP2. A energia cinética translacional foi definida como $3kT/2$ para uma temperatura de 300K ($1,4 \times 10^{-3}$ Hartree).....	56
Tabela 5.4 - Erro na energia de um grupo de trajetórias para cinco integradores. Descrevem uma dinâmica direta da reação $CH_3Cl + F$	57
Tabela 5.5 - Erro na energia de um grupo de trajetórias para cinco integradores. Descrevem uma dinâmica direta da reação $CH_3Br + F$	59
Tabela 5.6 - Erro na energia de um grupo de trajetórias para cinco integradores. Descrevem uma dinâmica direta da reação $CH_3Br + Cl$	61

Abreviações

CNDO: Complete Neglect of Differential Overlap.

CPU: Central Processing Unit.

DRKAI: Dinâmica Runge Kutta *ab initio*.

MP2: Moller Plesset de segunda ordem.

PCM: Polarizable Continuum Model.

RRKM: Rice Ramsperger Kassel Marcus.

SCF: Self Consistent Field.

S_N2: Substituição nucleofílica bimolecular.

TZV: Triple zeta valence.

Sumário

Dedicatória.....	iii
Agradecimentos.....	iv
Epígrafe.....	v
Resumo.....	vi
Abstract.....	vii
Índice de figuras.....	viii
Índice de tabelas.....	x
Abreviações.....	xi
Capítulo 1: Introdução.....	1
1.1 - A dinâmica molecular direta.....	1
1.2 - Referências Bibliográficas.....	4
Capítulo 2: Dinâmica Molecular.....	7
2.1 - Introdução.....	7
2.2 - Equações de movimento.....	7
2.3 - Métodos numéricos.....	8
2.3.1 - Notação.....	8
2.3.2 - O método de Euler.....	9
2.4 – Algoritmos de integração numérica.....	13
2.4.1 - Adams Bashforth Moulton.....	13
2.4.2 – Beeman.....	14
2.4.3 – Gauss Radau.....	15

2.4.4 - Runge Kutta Gill.....	19
2.4.5 - Runge Kutta simétrico.....	21
2.5 – Referências Bibliográficas.....	24
Capítulo 3: Estrutura eletrônica.....	26
3.1 – Equação de Schrödinger.....	26
3.2 – Princípio variacional.....	27
3.3 – Método Hartree Fock.....	30
3.4 – Moller Plesset de segunda ordem (MP2).....	32
3.5 – Referências Bibliográficas.....	33
Capítulo 4: Dinâmica Runge Kutta <i>ab initio</i>	35
4.1 – O programa.....	35
4.2 – Teste do algoritmo: Colisão entre duas partículas.....	38
4.3 – Referências Bibliográficas.....	42
Capítulo 5: Sistemas modelo e resultados.....	43
5.1 – Introdução.....	43
5.2 – Planetário.....	44
5.3 – Vibração da molécula de H ₂	47
5.4 – Moléculas poliatômicas.....	48
5.4.1 – Amostragem de moléculas poliatômicas.....	49
5.5 – Reações estudadas.....	52
5.5.1 – CH ₃ Cl + F ⁻	55
5.5.2 – CH ₃ Br + F ⁻	58

5.5.3 – CH ₃ Br + Cl ⁻	60
5.6 – Considerações gerais sobre os resultados.....	62
5.7 – Referências Bibliográficas.....	63
Capítulo 6: Considerações finais.....	64
6.1 - Conclusões e perspectivas.....	64
6.2 – Referências Bibliográficas.....	65
Anexo A: Outros trabalhos desenvolvidos durante o mestrado.....	66
Anexo B: Código completo do DRKAI.....	69
Anexo C: Participação em artigo.....	122

Capítulo 1: Introdução

1.1 – A dinâmica molecular direta

A dinâmica molecular clássica concede uma compreensão atomística de uma grande quantidade de processos na natureza. Isso inclui transferência de energia nas reações químicas em fase gasosa [1], colisões de gases em uma superfície [2], decomposição unimolecular [3], trocas conformacionais [4] e mecanismos [6-7]. Cada trajetória é propagada realizando uma integração numérica das equações de movimento clássicas. A dinâmica molecular através dessa abordagem se concentra em três grandes grupos. O uso de campos de força [8-9], onde funções de energia potencial são parametrizadas com sistemas simples e extrapoladas para sistemas maiores. Campos de força são usados em grande parte quando não há quebra de ligação química [10]. As forças podem ser obtidas através de superfícies, onde todos os pontos relevantes são obtidos através de métodos *ab initio* de alto nível, ajustando por fim a uma função matemática de muitos corpos [11-13]. Por último, é possível calcular a força através de métodos *ab initio* a cada passo da simulação, abordagem conhecida como dinâmica molecular *ab initio* direta. No decorrer desse texto, a última abordagem será chamada apenas de dinâmica direta.

A primeira dinâmica direta foi feita por Wang e Karplus em 1973 [14]. Eles usaram o método semi-empírico CNDO[15-16] no estudo da colisão $\text{CH}_2 + \text{H}_2$. Essa abordagem levou a conclusões que não eram evidentes apenas com a análise estática. Concluíram que o método proposto é apenas um pequeno passo no estudo da dinâmica de reações orgânicas, mas que é factível devido ao avanço dos computadores. Como perspectivas, propõe que o método semi-empírico seja mantido, mas com parâmetros específicos para situações reativas.

A primeira dinâmica direta usando métodos *ab initio* foi a substituição nucleofílica bimolecular (S_N2): $H + CH_4 \rightarrow CH_4 + H$ [17]. O autor conclui que graças ao novo método, foi possível realizar trajetórias em um espaço de coordenadas contendo 18 dimensões. Os resultados mostraram que a substituição do nucleófilo depende em grande parte da energia vibracional do metano. Foi observado que a dinâmica direta tem um alto custo, ou seja, requer grande tempo computacional. Mesmo assim, concluíram que essa técnica pode trazer mais informações em sistemas onde o estado de transição é conhecido. Os computadores sofreram grandes avanços nos últimos anos. Atualmente, a dinâmica molecular direta conquistou o seu espaço na ciência. A seguir, irei destacar alguns trabalhos recentes nessa área.

A conformação e a distribuição das cargas são determinantes nas propriedades catalíticas de um cluster de ouro. É possível, através da dinâmica molecular direta, obter informações sobre a estabilidade do cluster em função da temperatura. Através dessa abordagem, Manzoor et. al. [18] encontra uma troca conformacional dos compostos Au_6^- e Au_6^+ a 500K. Foi observado que o Au_6 neutro permaneceu estável até 1100K. Conclui que a carga ou um ligante pode ser usado como um ajuste fino para incrementar as propriedades químicas e físicas desse nanomaterial.

Outro resultado interessante é sobre o espectro de infravermelho. Para peptídeos grandes, o cálculo do espectro é obtido a partir da teoria do funcional da densidade junto com a aproximação harmônica. Sabe-se que essa aproximação falha no infravermelho distante ($<500\text{ cm}^{-1}$) porque há grande acoplamento dos modos vibracionais anarmônicos [19]. Esse problema é contornado pela dinâmica molecular direta. Jaeqx et. al. [20] mostrou que a dinâmica direta apresentou um excelente acordo com o experimento abaixo de 100 cm^{-1} para alguns peptídeos. Sugerindo que a combinação da dinâmica

direta com experimentos pode ser uma ferramenta poderosa na caracterização de peptídeos em fase gasosa.

Por último, ao estudar reações S_N2 do tipo $X^- + CH_3Y \rightarrow XCH_3 + Y^-$ com dinâmica direta, Hase et. al. [21] mostrou que esse sistema possui propriedades não estatísticas. Por exemplo, a decomposição do estado de transição não obedece a teoria RRKM (Rice–Ramsperger–Kassel–Marcus) devido à lenta redistribuição de energia intramolecular. Esse desvio é visto nos experimentos para algumas moléculas [22-23].

A dinâmica direta requer muito tempo de CPU, a referência [24] relata que algumas trajetórias levaram sete dias para se completar. Um algoritmo que pode contribuir para a redução do tempo de cálculo é o integrador. Integradores eficientes fornecem o mesmo resultado dos menos eficientes com um tempo de CPU menor. Muito trabalho é feito na melhoria desses algoritmos [25-26]. Entretanto, é evidente que não há um algoritmo que seja superior a todos os outros para qualquer tipo de sistema, dado que novos integradores são constantemente desenvolvidos [27-29]. Embora a referência [25] mencione que os integradores baseados na hessiana sejam superiores para cálculos de dinâmica direta, ainda se observa na literatura algoritmos que usam apenas o gradiente [21,30-31].

Há alguns software que realizam a dinâmica direta *ab initio*. Em especial, o VENUS96 tem sido usado por muitos anos com sucesso [11,21,32]. Outro software a ser mencionado é o Quickstep [33], o qual possui características inovadoras na obtenção do gradiente a partir da teoria do funcional da densidade. Por último, o Gaussian 09 [34] possui um integrador baseado em hessiana extremamente eficiente. O desenvolvimento de um software brasileiro que desempenha esse papel traz vantagens como formação de recursos humanos, permitindo o aprofundamento no conhecimento de algoritmos e equações envolvidas nesse processo. Por exemplo, tal estudo pode permitir a criação de

um núcleo de desenvolvimento completamente nacional, onde novos algoritmos e ideias pudessem ser criadas.

Neste trabalho, um software de dinâmica direta foi implementado em linguagem prolog. Cinco algoritmos de integração numérica disponíveis na literatura foram avaliados, de forma a encontrar o mais adequado para o uso em dinâmica direta.

No capítulo 2 será discutido um pouco da dinâmica molecular, os integradores e as equações correspondentes.

No capítulo 3 será apresentado uma breve revisão das teorias de estrutura eletrônica, importante principalmente para a dinâmica direta.

No capítulo 4 o algoritmo DRKAI será apresentado junto com um fluxograma e detalhes de sua implementação.

O capítulo 5 descreve os vários testes numéricos realizados, validando os algoritmos propostos. Isso foi feito tanto em sistemas clássicos onde a solução analítica era conhecida, como em dinâmica molecular direta *ab initio*.

No capítulo 6 estarão as considerações finais deste trabalho.

1.2 - Referências Bibliográficas

- [1] W.L. Hase, D.M. Ludlow, R.J. Wolf, T. Schlick. J. Phys. Chem. **1981**, 85, 958.
- [2] D.G. Schultz, S.B. Weinhaus, L. Hanley, P. de Sainte Claire, W.L. Hase. J. Chem. Phys. **1997**, 106, 10337.
- [3] W.L. Hase, R.J. Wolf, C.S. Sloane. J. Chem. Phys. **1979**, 71, 2911.
- [4] D.L. Bunker, W.L. Hase. J. Chem. Phys. **1973**, 59, 4621.
- [5] D.-H. Lu, W.L. Hase, R.J. Wolf. J. Chem. Phys. **1986**, 85, 4422.
- [6] P. de Sainte Claire, K.C. Hass, W.F. Schneider, W.L. Hase. J. Chem. Phys. **1997**, 106, 7331.

- [7] K. Bolton, W.L. Hase, C. Doubleday Jr. J. Phys. Chem. B. **1999**, 103, 3691.
- [8] D. S. Hou, Y. Zhu, Y. Y. Lu, Z. J. Li. Mat. Chem. Phys. **2014**, 503, 511.
- [9] M. Amani, S. Amjad-Iranagh, K. Golzar, G. M. M. Sadeghi, H. Modaress. J. Membr. Sci. **2014**, 462, 28.
- [10] U. Burkert, N.L. Allinger. Mol Mech, em: ACS Monograph, vol. 177, American Chemical Society, Washington, DC, **1982**.
- [11] B. R. L. Galvão, A. J. C. Varandas, J. P. Braga, J. C. Belchior. J. Phys. Chem. Lett. **2013**, 4, 2292.
- [12] P. J. S.B. Caridade, B. R. L. Galvão, A. J. C. Varandas. J. Phys Chem. A. **2010**, 114, 6063.
- [13] J. M.C. Marques, A. Riganelli, A. J. C. Varandas. Quim. Nova. **2003**, 26, 5, 769-778.
- [14] I.S.Y. Wang, M. Karplus. J. Amer. Chem. Soc. **1973**, 95, 8160.
- [15] J.A. Pople, D. P. Santry, G. A. Segal. J. Chem. Phys. **1965**, 4, 3, S129; J.A.Pople, G. A.Segal.J. Chem. Phys. **1965** 43,S136.
- [16] H. Fischer and H. Kollmar. Theor. Chim. Acta. **1969**, 13, 213.
- [17] C. Leforestier. J. Chem. Phys. **1978**, 68, 4406.
- [18] D. Manzoor, S. Pal. J. Phys. Chem. C. **2013**, 117, 20982-20990.
- [19] M.-P. Gaigeot. Phys. Chem. Chem. Phys. **2010**, 12, 3336.
- [20] S. Jaeqx, J. Oomens, A. Cimas, M. Gaigeot, A. M. Rijs. Angew. Chem. Int. Ed. **2014**, 53, 3663.
- [21] P. Manikandan, J. Zhang, W.L. Hase. J. Phys. Chem. A. **2012**, 116, 12, 3061.
- [22] D. S. Tonner, T. B. McMahon. J. Am. Chem. Soc. **2000**, 122, 8783.
- [23] R. Wester, A. E. Bragg, A. V. Davis, D. M. J. Neumark. Chem. Phys. **2003**, 119, 10032.
- [24] J. Zhang, U. Lourderaj, R. Sun, J. Mikosch, R. Wester, W. L. Hase. J. Chem. Phys. **2013**, 138, 114309.
- [25] H. B. Schlegel. J. Chem. Theory Comput. **2013**, 9, 3293-3298.
- [26] P. F. Tupper. IMA Journal of Numerical Analysis. **2005**, 25, 286-309.

- [27] M. Wandelt, M. Gunther, F. Knechtli, M. Striebel. Appl. Numer. Math. **2012**, 62, 1740-1748.
- [28] X. You, B. Chen. Mathematics and Computers in Simulation. **2013**, 94, 76-95.
- [29] S. Blanes, P. C. Moan. J. Comp. Appl. Math. **1994**, 51, 375-382.
- [30] J. Xie, S. C. Kohale, W. L. Hase, S. G. Ard, J. J. Melko, N. S. Shuman, A. A. Viggiano. J. Phys. Chem. A. **2013**, 117, 14019-14027.
- [31] W. L. Hase et. al. J. Phys. Chem. A. **2014**, 118, 2228-2236.
- [32] W. Lourderaj, R. Sun, S.C. Kohale, G.L. Barnes, W.A. de Jong, T.L. Windus, W.L. Hase. Comp. Phys. Comm. **2014**, 185, 1074.
- [33] J.V. Vondele, M. Krack, F. Mohamed, M. Parrinello, T. Chassaing, J. Hutter. Comp. Phys. Comm. **2005**, 167, 103.
- [34] Frisch, M. J. et al., GAUSSIAN 09 (Revision D.01), Gaussian Inv., Pittsburg, PA, **2009**.

Capítulo 2: Dinâmica Molecular

2.1 - Introdução

A simulação de uma reação química usando trajetórias clássicas permite a compreensão de vários processos. Nessa abordagem, cada trajetória é propagada realizando uma integração numérica das equações de movimento clássicas. Para isso é necessário avaliar o gradiente da energia em cada ponto, podendo ser obtido a partir de potenciais empíricos [1-2], superfícies [3-5] e de forma direta, calculando a força com métodos ab initio sempre que necessário. A dinâmica direta é o foco desse trabalho.

Nas seções seguintes será apresentado as equações de movimento clássicas e os integradores testados.

2.2 - Equações de movimento

Um sistema de N partículas com massas m_i , descritos pela posição \mathbf{x}_i e velocidade \mathbf{v}_i , interagem entre si por um potencial $u(\mathbf{X})$. Onde \mathbf{X} é um vetor contendo todas as coordenadas dos átomos. A função que descreve esse sistema é o hamiltoniano:

$$H = \sum_i \frac{1}{2} m_i v_i^2 + u(\mathbf{X}) \quad (2.1)$$

as forças entre as partículas são definidas pelo gradiente da função de energia potencial:

$$\mathbf{f} = -\nabla u(\mathbf{X}) \quad (2.2)$$

a equação de movimento pelo formalismo de Newton é expressa pela seguinte equação:

$$\mathbf{M} \frac{d^2 \mathbf{X}}{dt^2} = \mathbf{f} \quad (2.3)$$

onde M é uma matriz com as massas na diagonal. Há outro formalismo, a mecânica hamiltoniana, o qual transforma essa equação diferencial de segunda ordem em duas equações diferenciais de primeira ordem. Em coordenadas cartesianas, as equações de movimento podem ser reescritas como:

$$m_i \frac{dv_i}{dt} = -\frac{\partial H}{\partial x} = f_i \quad (2.4)$$

$$\frac{dx_i}{dt} = +\frac{1}{m_i} \frac{\partial H}{\partial v} = v_i \quad (2.5)$$

as equações (2.4) e (2.5) são válidas para todos os sistemas químicos estudados neste trabalho. Equações diferenciais de primeira ordem podem ser facilmente integradas pelos algoritmos tradicionais. Na próxima seção será apresentado um exemplo, para isso será usado um algoritmo simples: o método de Euler.

2.3 – Métodos numéricos

2.3.1 – Notação

No processo de integração numérica em coordenadas cartesianas, não há, e não pode haver um favorecimento de alguma das direções. Por isso, a mesma expressão usada em uma coordenada, deve ser usada em todas as demais. Isso também vale para as velocidades. Ao longo deste capítulo, as coordenadas serão representadas como: x_n , x_{n+1} , x_{n-1} correspondendo respectivamente a posição atual, a próxima posição, e uma posição anterior. Notação análoga será usada para velocidades. Quando for conveniente, a letra y representará um vetor que contém as duas grandezas:

$$y_n = [x_n \ v_n] \quad (2.6)$$

a aceleração do sistema é definida como a força dividida pela massa. Quando a forma vetorial (2.6) for necessária, as derivadas também estarão nesse formato:

$$y'_n = [v_n \ a_n] \quad (2.7)$$

$$a_n = \frac{f(x_n)}{m} \quad (2.8)$$

onde m é a massa correspondente à coordenada em questão. Apenas a expressão (2.8) será utilizada quando a notação vetorial não for relevante. O índice em subscrito tem o mesmo significado para todas as grandezas, então a_{n-1} corresponde à aceleração de uma iteração anterior.

O passo será representado pela letra h . Grandeza que define uma variação no tempo do sistema. Em geral, o passo está relacionado ao custo computacional da simulação. Passos longos têm um custo menor, mas provocam grandes saltos no sistema, podendo desprezar interações importantes que ocorreriam durante esse intervalo.

A seção seguinte terá como objetivo ilustrar uma integração numérica. Para isso será usado um dos integradores mais simples, o método de Euler, aplicado a um sistema do tipo massa-mola.

2.3.2 – O método de Euler

Seja um problema mecânico em coordenadas cartesianas, a seguinte notação será utilizada:

$$y(t) = [x(t) \ v(t)] \quad (2.9)$$

a derivada pode ser obtida usando a mecânica hamiltoniana (2.4) e (2.5).

$$y'(t) = [v(t) \ f(x(t))]$$
(2.10)

O método de Euler consiste em resolver as seguintes equações a cada iteração, que chamamos de passo:

$$y_{n+1} = y_n + hy'(t_n, y_n) \quad (2.11)$$

sendo y_{n+1} a nova configuração do sistema, h o passo e y' definido na expressão (2.10). Segue um exemplo de um sistema unidimensional submetido a um potencial harmônico do tipo massa-mola.

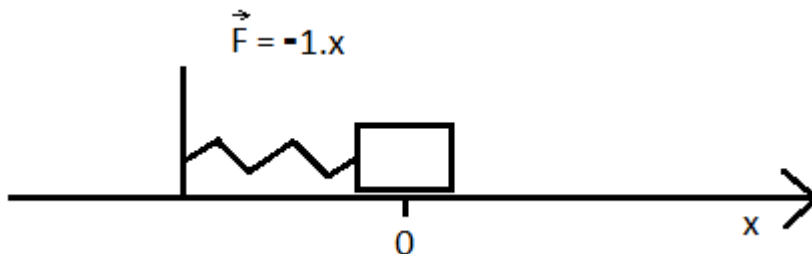


Figura 2.1 – Uma massa submetida a uma força de restauração com constante $k=1$ N/m.

A derivada desse sistema toma o seguinte formato:

$$y'(t_n, y_n) = [v_n \quad -x_n] \quad (2.12)$$

seja a constante da mola $k=1$ N/m, posição de equilíbrio na origem e massa $m=1$ Kg. Se o sistema for iniciado no ponto $x=2$ m, com velocidade igual a zero, a solução exata é obtida a partir da solução da equação de Newton (2.2). Para as condições iniciais definidas nesse problema, a fase do sistema é igual a zero. A velocidade angular é $\omega=1$ s⁻¹ nessas condições. As equações (2.13) e (2.14) são, portanto, a solução analítica desse problema.

$$x(t) = 2 \cos(\omega t) \quad (2.13)$$

$$v(t) = -2 \text{sen}(\omega t) \quad (2.14)$$

Na tabela seguinte, os primeiros passos da integração numérica estão apresentados. É um sistema com potencial harmônico e foi integrado pelo método de Euler.

Tabela 2.1 - Primeiros passos de uma integração numérica com o método de Euler. Sistema massa-mola com constante $k= 1 \text{ N/m}$ e $m=1 \text{ Kg}$. O passo (h) foi $0,1 \text{ s}$. Destaque em duas atualizações de posição e velocidade respectivamente

$t \text{ (s)}$	$x_{\text{exato}} \text{ (m)}$	$v_{\text{exato}} \text{ m/s}$	$x_{n+1} = x_n + hv_n$	$v_{n+1} = v_n + h(-x_n)$	Erro em $x \text{ (m)}$	Erro em $v \text{ (m)}$
0,0000	2,0000	0,0000	2,0000	0,0000	0,0000	0,0000
0,1000	1,9900	-0,1997	2,0000	-0,2000	0,0100	0,0003
			x_n	$+hv_n$		
0,2000	1,9601	-0,3973	1,9800	-0,4000	0,0199	0,0027
			$+h(-x_n)$	v_n		
0,3000	1,9107	-0,5910	1,9400	-0,5980	0,0293	0,0070
0,4000	1,8421	-0,7788	1,8802	-0,7920	0,0381	0,0132
0,5000	1,7552	-0,9589	1,8010	-0,9800	0,0458	0,0212

A tabela abaixo mostra o mesmo sistema da tabela 2.1 com um passo menor ($0,01 \text{ s}$).

Tabela 2.2: Primeiros passos de uma integração numérica. O sistema é uma partícula sob um potencial harmônico. Foi integrado via método de Euler

t (s)	x_{exato} (m)	v_{exato} (m/s)	$x_{n+1} = x_n + hv_n$	$v_{n+1} = v_n + h(-x_n)$	Erro em x (m)	Erro em v (m)
0,0000	2,0000	0,0000	2,0000	0,0000	0,0000	0,0000
0,0100	1,9999	-0,0200	2,0000	-0,0200	0,0001	0,0000
0,0200	1,9996	-0,0400	1,9998	-0,0400	0,0002	0,0000
0,0300	1,9991	-0,0600	1,9994	-0,0600	0,0003	0,0000
0,0400	1,9984	-0,0800	1,9988	-0,0800	0,0004	0,0000
0,0500	1,9975	-0,1000	1,9980	-0,1000	0,0005	0,0000
0,0600	1,9964	-0,1199	1,9970	-0,1200	0,0006	0,0000
0,0700	1,9951	-0,1399	1,9958	-0,1399	0,0007	0,0000
0,0800	1,9936	-0,1598	1,9944	-0,1599	0,0008	0,0001
0,0900	1,9919	-0,1798	1,9928	-0,1798	0,0009	0,0001
0,1000	1,9900	-0,1997	1,9910	-0,1998	0,0010	0,0001

Observa-se que quanto menor o passo maior a precisão da integração numérica. Por exemplo, para o sistema massa-mola, usando um passo de **0,1 s**, obtêm-se **$x=2,000$ m** em **$t=0,1$ s** (tabela 2.1). Usando um passo dez vezes menor e quando o sistema atinge esse mesmo tempo (**$t=0,1$ s**), o valor de **x** é muito mais preciso: **$x=1,991$ m**, se comparado ao exato **$x_{\text{exato}} = 1,990$ m** (tabela 2.2). Isso mostra que o passo está intimamente relacionado à precisão do sistema.

Na seção seguinte, os integradores analisados nesse trabalho serão apresentados com as respectivas equações.

2.4 – Algoritmos de integração numérica

2.4.1 - Adams Bashforth Moulton

O procedimento de Adams Moulton pertence a um conjunto de métodos conhecidos como lineares de passo múltiplo (*Linear multistep*). Do teorema fundamental do cálculo [6] têm-se:

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} y'(t) dt \quad (2.15)$$

Os métodos de Adams consistem em aproximar o integrando da equação (2.15) por um polinômio. Isso é feito através de um formalismo chamado interpolação polinomial na forma lagrangiana, sendo um procedimento análogo à série de Taylor. Usando esse procedimento e aplicando à integral da equação (2.15), se obtêm os coeficientes de Adams Moulton. Para se obter um integrador de ordem k basta utilizar como aproximador um polinômio de ordem $k-1$.

O método Adams Bashforth Moulton implementado neste trabalho é dividido em duas regiões assim como na referência [7]. A primeira parte é uma aproximação Adams Moulton de quarta ordem. Isso significa que o polinômio que aproxima a função possui ordem três. Utilizando um polinômio de ordem três na expressão (2.15) se obtêm a seguinte equação:

$$y_{n+1} = y_n + \frac{h}{720} [1901y'_n - 2774y'_{n-1} + 2616y'_{n-2} - 1274y'_{n-3} + 251y'_{n-4}] \quad (2.16)$$

onde, nesse procedimento, o integrador usa informação de até quatro iterações anteriores, a saber, as derivadas y'_{n-1}, y'_{n-2} , etc. A segunda equação é chamada corretor, porque realiza uma pequena mudança na posição do previsor, melhorando o resultado. Na atual implementação usou-se o método de Adams Bashforth de ordem cinco, através de um polinômio de ordem quatro na expressão (2.15) se obtêm a equação (2.17).

$$y_{n+1} = \frac{h}{1440} [475y'_{n+1} + 1427y'_n - 798y'_{n-1} + 482y'_{n-2} - 173y'_{n-3} + 27y'_{n-4}] \quad (2.17)$$

O Adams Bashforth Moulton é um algoritmo muito usado no estudo da dinâmica com superfícies [8-11]. Sabe-se que possui grande estabilidade e precisão. O algoritmo de Adams Bashforth Moulton não é auto-inicializável. Ou seja, os primeiros cinco passos precisam ser realizados com outro integrador. Nesse trabalho, assim como na ref. [7], a inicialização foi feita usando o algoritmo Runge Kutta Gill.

2.4.2 - Beeman

No método de Verlet a coordenada \mathbf{x} e a velocidade \mathbf{v} são obtidas pelas seguintes expressões:

$$x_{n+1} = 2x_n - x_{n-1} + h^2 a_n \quad (2.18)$$

$$v_n = h \frac{x_{n+1} - x_{n-1}}{2} \quad (2.19)$$

sendo h o passo da simulação. A velocidade é assumida como constante entre \mathbf{x}_{n+1} e \mathbf{x}_{n-1} . Essa expressão pode ser aprimorada considerando uma mudança na velocidade ao longo desse intervalo. Na abordagem proposta por beeman, considera-se que a aceleração varie de acordo com a seguinte expressão [12]:

$$a(t) = a_n + t \frac{a_n - a_{n-1}}{h} \quad (2.20)$$

após integrar (2.19) e realizando um pequeno rearranjo algébrico obtêm-se (2.21) e (2.22).

$$x_{n+1} = x_n + hv_n + \frac{1}{6}(4a_n - a_{n-1})h^2 \quad (2.21)$$

$$v_{n+1} = v_n + \frac{1}{6}(2a_{n+1} + 5a_n - a_{n-1})h \quad (2.22)$$

Sabe-se que a equação de Beeman é equivalente a de Verlet com algumas vantagens, por exemplo, há uma dependência da velocidade na posição, facilitando o controle da temperatura, em geral, feito através das velocidades. A formulação de beeman retira a

dependência direta da posição na velocidade, evitando operações entre grandes números, e consequentemente erros de arredondamento.

O Verlet é largamente utilizado na literatura [13-16]. A modificação proposta por Beeman foi escolhida porque é ligeiramente mais eficiente do que o Verlet tradicional. Porém, tem um maior custo em memória. Isso pode ser um problema para sistemas grandes, como uma simulação de líquidos ou um sistema biológico, mas no caso da dinâmica direta o aumento é negligível.

2.4.3 - Gauss Radau

A força a qual a partícula está sujeita é uma função das posições dos átomos. Expandindo a força em série de Taylor obteríamos:

$$m\ddot{x} = f = f_1 + A_1 t + A_2 t^2 + A_3 t^3 + \dots + A_N t^N \quad (2.23)$$

onde f_1 é a força no tempo $t=0$. Integrando a equação (2.23) duas vezes é possível obter as posições e velocidades (2.24) e (2.25).

$$mx = x_1 + \dot{x}t + f_1 \frac{t^2}{2} + A_1 \frac{t^3}{6} + \dots + A_N \frac{t^{N+2}}{(N+1)(N+2)} \quad (2.24)$$

$$m\dot{x} = \dot{x}_1 + f_1 t + A_1 \frac{t^2}{2} + A_2 \frac{t^3}{3} + \dots + A_N \frac{t^{N+1}}{N+1} \quad (2.25)$$

No método de Gauss Radau, procura-se os valores A_1 , A_2 e A_3 que produzam o melhor resultado na dinâmica. Os números A_1 , A_2 e A_3 serão ajustados a cada passo da simulação e seu resultado, pode, inclusive, ser melhor do que o obtido apenas com a expansão em série de Taylor [17].

O ajuste é feito através de um uma sequência de tamanho T , com os tempos $t_1=0, t_2, t_3, \dots$ (figura 2.2). Nesse contexto, o comprimento da sequência é o passo da simulação ($T=h$). Ao longo do passo, a força é calculada várias vezes, com o objetivo de ajustar os parâmetros A 's.

Por fim, os mais adequados são aplicados na equação (2.24). Nesta seção, a notação do artigo será mantida, ou seja, a letra T representa o comprimento da sequência onde os parâmetros são ajustados. Essa grandeza, também é o passo da simulação.

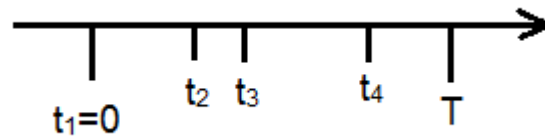


Figura 2.2 – Figura esquemática que apresenta uma sequência do integrador Gauss Radau.

A força é calculada em cada um desses pontos. É possível aproximar uma expansão em série de Taylor com um método chamado diferenças divididas. As expressões abaixo ilustram o processo. A força pode ser inteligentemente reescrita como:

$$f = f_1 + \alpha_1 t + \alpha_2 t(t - t_2) + \alpha_3 t(t - t_2)(t - t_3) + \dots \quad (2.26)$$

com uma leve inspeção é possível observar que os coeficientes α_n podem ser determinados por:

$$t = 0 \quad ; \quad f = f_1 \quad (2.27)$$

$$t = t_2 \quad ; \quad f = f_1 + \alpha_1 t_2 \quad (2.28)$$

e assim por diante. Como mostrado na figura 2.1, as forças são calculadas em cada ponto do intervalo. Isolando α_n e usando a seguinte notação: $t_{nj} = t_n - t_j$, têm-se:

$$\alpha_1 = \frac{f(t_2) - f(t_1)}{t_2} \quad (2.29)$$

$$\alpha_2 = \left(\frac{f(t_3) - f(t_1)}{t_3} - \alpha_1 \right) / t_{32} \quad (2.30)$$

$$\alpha_3 = \left[\left(\frac{f(t_4) - f(t_1)}{t_4} - \alpha_1 \right) / t_{42} - \alpha_2 \right] / t_{43} \quad (2.31)$$

Comparando a expressão (2.26) com (2.23) pode-se obter os coeficientes \mathbf{A}_1 , \mathbf{A}_2 e etc.

Igualando obtêm-se:

$$A_1 = \alpha_1 - \alpha_2 t_2 + t_2 t_3 \alpha_3 \quad (2.32)$$

$$A_2 = \alpha_2 + (-t_2 - t_3) \alpha_3 \quad (2.33)$$

$$A_3 = \alpha_3 \quad (2.34)$$

na atual implementação a expansão ocorre apenas até o terceiro termo (\mathbf{A}_3). Obtidos seus valores, usa-se as expressões (2.35) e (2.36) para atualizar as posições e velocidades.

$$x(T) = x_1 + v_1 T + f_1 \frac{T^2}{2} + A_1 \frac{T^3}{6} + A_2 \frac{T^4}{12} + A_3 \frac{T^5}{20} \quad (2.35)$$

$$v(T) = v_1 + f_1 T + A_1 \frac{T^2}{2} + A_2 \frac{T^3}{3} + A_3 \frac{T^4}{4} \quad (2.36)$$

O seguinte modelo esquemático da implementação presente no DRKAI é mostrado abaixo:

1. *Definição de todos os A's como zero.*
2. *Para $t < t_{\text{máximo}}$ faça:*
 - 2.1 *Repetição do refinamento dos parâmetros e obtenção das novas coordenadas e velocidades. **
 - 2.1.1 *Definição dos espaçamentos. t_1, t_2, t_3, t_4^{**} .*
 - 2.1.2 *Calculo do ponto $\mathbf{x}(t_2)$ usando os A's anteriores*** a partir da seguinte expressão:*

$$x(t_2) = x_1 + v_1 t_2 + f_1 \frac{t_2^2}{2} + \left[A_1 \frac{t_2^3}{6} + A_2 \frac{t_2^4}{12} + A_3 \frac{t_2^5}{20} \right] \quad (2.37)$$
 - 2.1.3 *Calculo da força $\mathbf{f}(\mathbf{x}(t_2))$.*
 - 2.1.4 *Obtenção do α_1 a partir da expressão (2.29).*
 - 2.1.5 *Usando o novo α_1 , e os α_2 e α_3 antigos, é possível corrigir o \mathbf{A}_1 usando a expressão (2.32).*

2.1.6 *Calculo do ponto $\mathbf{x}(t_3)$ usando os \mathbf{A} 's anteriores a partir da expressão seguinte:*

$$x(t_3) = x_1 + v_1 t_3 + f_1 \frac{t_3^2}{2} + A_1 \frac{t_3^3}{6} + \left[A_2 \frac{t_3^4}{12} + A_3 \frac{t_3^5}{20} \right] \quad (2.38)$$

2.1.7 *Calculo da força $\mathbf{f}(\mathbf{x}(t_3))$.*

2.1.8 *Obtenção do α_2 a partir da expressão (2.30).*

2.1.9 *Usando o novo α_2 , o α_1 obtido em 2.1.4, e o α_3 antigo, é possível corrigir o \mathbf{A}_1 novamente com a expressão (2.32), e corrigir o \mathbf{A}_2 com a expressão (2.33).*

2.1.6 *Calculo do ponto $\mathbf{x}(t_4)$ usando os \mathbf{A} 's anteriores a partir da seguinte expressão:*

$$x(t_4) = x_1 + v_1 t_4 + f_1 \frac{t_4^2}{2} + A_1 \frac{t_4^3}{6} + \left[A_2 \frac{t_4^4}{12} + A_3 \frac{t_4^5}{20} \right] \quad (2.39)$$

2.1.7 *Calculo da força $\mathbf{f}(\mathbf{x}(t_4))$.*

2.1.8 *Obtenção do α_3 a partir da expressão (2.31).*

2.1.9 *Usando o novo α_3 , o α_1 obtido em 2.1.4 e o α_2 obtido em 2.1.8, é possível corrigir o \mathbf{A}_1 novamente com a expressão (2.32), corrigir o \mathbf{A}_2 com a expressão (2.33) e corrigir o \mathbf{A}_3 com a expressão (2.34).*

2.2 *A partir do processo 2.1 obtêm-se os \mathbf{A} 's. Sabe-se que o primeiro valor obtido nesse processo ainda não é adequado para a dinâmica, então precisa-se repetir o procedimento para refiná-los. Na primeira iteração o processo de obtenção dos \mathbf{A} 's 2.1 é repetido seis vezes. Nas iterações seguintes o processo é repetido duas vezes [18].*

2.3 *Usando as expressões (2.35-2.36) obtêm-se as novas coordenadas e velocidades.*

2.4 *Impressão do espaço de fase com as novas coordenadas.*

3. Fim do programa.

* Na primeira iteração, todos os A's são zero, então é necessário muito mais cuidado para inicializá-los. A ref [18] sugere que haja uma repetição do refinamento por seis vezes antes que eles atinjam um valor adequado.

** Os espaçamentos são a princípio arbitrários, mas sabe-se que com uma escolha adequada é possível aprimorar o resultado. Neste trabalho, assim como na ref [18], foram utilizados o espaçamento de Gauss-Radau:

$$t_2 = T * 0,212340538239 \quad (2.40)$$

$$t_3 = T * 0,590533135559 \quad (2.41)$$

$$t_4 = T * 0,911412040488 \quad (2.42)$$

o comprimento do intervalo é igual ao passo da simulação nesse contexto.

*** São definidos como zero no início da simulação.

O algoritmo de Gauss Radau [17] é um integrador de ordem quatro, mas graças aos espaçamentos de Gauss Radau pôde ser aprimorado para um integrador de ordem sete. Por isso pode ter grande utilidade em sistemas químicos.

2.4.4 - Runge Kutta Gill

Algoritmos do tipo Runge Kutta são muito utilizados na literatura em diversos problemas [19-21]. São métodos que, em geral, avaliam as derivadas em pontos intermediários antes do passo ser completado. A partir da inclinação do ponto intermediário é possível realizar uma melhor previsão do próximo ponto da trajetória. A figura 2.2 ilustra esse processo.

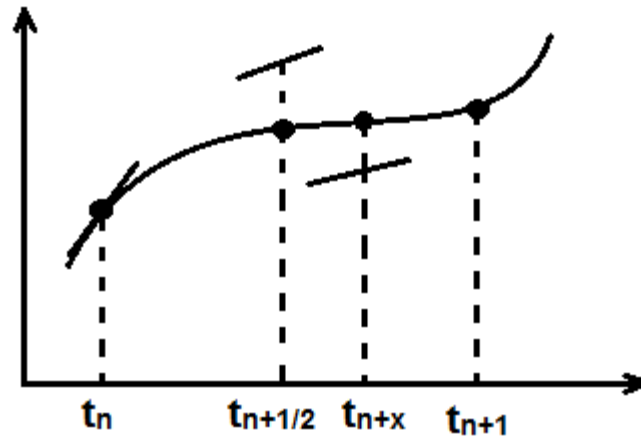


Figura 2.3 - Esquema de uma integração do tipo Runge Kutta.

A principal diferença entre os métodos Runge Kutta é o número de pontos escolhidos durante o intervalo e qual será o peso de cada um no resultado final. Neste trabalho, foi utilizado uma quantidade de pontos e pesos geradas conforme foi proposto por Gill e apresentado na referência [7]. As equações 2.43-2.54 mostram o processo.

$$k_1 = hk'_n \quad (2.43)$$

$$y_1 = y_n + \frac{1}{2}k_1 \quad (2.44)$$

$$q_1 = k_1 \quad (2.45)$$

$$k_2 = hy'_{x_1} \quad (2.46)$$

$$y_2 = y_1 + \left(1 - \frac{\sqrt{2}}{2}\right)(k_2 - q_1) \quad (2.47)$$

$$q_2 = (2 - \sqrt{2})k_2 - \left(2 - \frac{3\sqrt{2}}{2}\right)q_1 \quad (2.48)$$

$$k_3 = hy'_{x_2} \quad (2.49)$$

$$y_3 = y_2 + \left(1 + \frac{\sqrt{2}}{2}\right)(k_3 - q_2) \quad (2.50)$$

$$q_3 = (2 + \sqrt{2})k_3 - \left(2 + \frac{3\sqrt{2}}{2}\right)q_2 \quad (2.51)$$

$$k_4 = hy'_{x_3} \quad (2.52)$$

$$y_4 = y_3 + \frac{1}{6}(k_4 - 2q_3) \quad (2.53)$$

$$y_{n+1} = y_4 \quad (2.54)$$

O Runge Kutta Gill tem sido utilizado em conjunto com o Adams Bashforth Moulton por muitos anos [8-11]. Para uma dinâmica usando superfícies essa abordagem parece ser a mais eficiente. Entretanto, a dinâmica direta tem limitações e necessidades diferentes. Nesse trabalho será avaliada a performance desse algoritmo sem o acoplamento com outros métodos.

2.4.5 - Runge Kutta simétrico

O integrador aqui chamado Runge Kutta simétrico é um método de sexta ordem e está desenvolvido na ref. [22]. O integrador é simétrico por construção e as inclinações calculadas durante o passo de integração são espelhadas. A figura 2.3 ilustra esse processo.

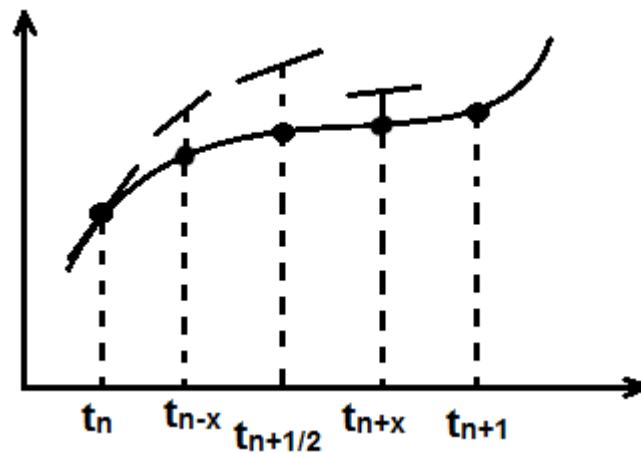


Figura 2.4 - Esquema de um integrador do tipo Runge Kutta simétrico.

Esse procedimento possui uma propriedade chamada integração reversa. Ou seja, caso se inverta as velocidades no último ponto da trajetória, o ponto inicial é obtido. Isso nem sempre acontece, em outros Runge Kutta, por exemplo; em que os pontos intermediários escolhidos não são simétricos. Se a velocidade for invertida no fim da trajetória, novos pontos são escolhidos, o que causará um desvio do ponto inicial esperado.

O Runge Kutta simétrico é válido apenas quando o hamiltoniano puder ser particionado: $H = T(\mathbf{p}) + V(\mathbf{q})$. Condição presente em todos os testes deste trabalho. Não seria válido, por exemplo, para uma partícula carregada sob o efeito de um campo magnético, pois nesse caso o potencial iria depender tanto da posição quanto da velocidade. As equações 2.55-2.73 mostram esse procedimento.

$$x_1 = x_n + ha(0)v_n \quad (2.55)$$

$$v_1 = v_n + ha(1)a_1 \quad (2.56)$$

$$x_1 = x_1 + ha(2)v_1 \quad (2.55)$$

$$v_1 = v_1 + ha(3)a_2 \quad (2.56)$$

$$x_1 = x_2 + ha(4)v_2 \quad (2.55)$$

$$v_1 = v_2 + ha(5)a_3 \quad (2.56)$$

$$x_1 = x_3 + ha(6)v_3 \quad (2.55)$$

$$v_1 = v_3 + ha(7)a_4 \quad (2.56)$$

$$x_1 = x_4 + ha(8)v_4 \quad (2.55)$$

$$v_1 = v_4 + ha(9)a_5 \quad (2.56)$$

$$x_1 = x_5 + ha(10)v_5 \quad (2.55)$$

$$v_1 = v_5 + ha(11)a_6 \quad (2.56)$$

$$x_1 = x_6 + ha(12)v_6 \quad (2.55)$$

$$v_1 = v_6 + ha(13)a_7 \quad (2.56)$$

$$x_1 = x_7 + ha(14)v_7 \quad (2.55)$$

$$v_1 = v_7 + ha(15)a_8 \quad (2.56)$$

$$x_1 = x_8 + ha(16)v_7 \quad (2.55)$$

$$v_1 = v_8 + ha(17)a_9 \quad (2.56)$$

$$x_{n+1} = x_9 + ha(18)v_9 \quad (2.55)$$

Na tabela (era para colocar parágrafo aqui) abaixo estão os coeficientes das equações (2.55-2.73) com uma precisão de 20 casas decimais.

Tabela 2.3: Coeficientes das equações (2.55-2.73)

Coeficientes	Valores
a(0)=a(18)	0,095176255
a(1)=a(17)	0,666296894
a(2)=a(16)	-0,127950286
a(3)=a(15)	0,024618901
a(4)=a(14)	0,105972953
a(5)=a(13)	-0,410725534
a(6)=a(12)	0,448222277
a(7)=a(11)	0,657729262
a(8)=a(10)	-0,021421199
a(9)	2,0x(-0,43791952338277493384)

Esse algoritmo foi usado em uma simulação de dinâmica direta recentemente [3]. Mesmo necessitando calcular a força 10 vezes a cada passo, possui grande eficiência e um custo benefício favorável [22]. Adicionalmente, é um algoritmo simplético, ou seja, conserva o volume do espaço de fase durante toda a simulação.

2.5 – Referencias Bibliográficas

- [1] D. S. Hou, Y. Zhu, Y. Y. Lu, Z. J. Li. *Mat. Chem. Phys.* **2014**, 503, 511.
- [2] M. Amani, S. Amjad-Iranagh, K. Golzar, G. M. M. Sadeghi, H. Modaress. *J. Membr. Sci.* **2014**, 462, 28.
- [3] B. R. L. Galvão, A. J. C. Varandas, J. P. Braga, J. C. Belchior. *J. Phys. Chem. Lett.* **2013**, 4, 2292-2297.
- [4] P. J. S. B. Caridade, B. R. L. Galvão, A. J. C. Varandas. *J. Phys Chem. A.* **2010**, 114, 6063-6070.
- [5] J. M. C. Marques, A. Riganelli, A. J. C. Varandas. *Quim. Nova.* **2003**, 26, 5, 769-778.
- [6] E. Swokowski. *Cálculo com geometria analítica: Volume 1.* **1994.** Makron Books.
- [7] *Atom-molecule collision theory.* Editado por R. B. Bernstein. Plenum Press, Nova Yorque. **1979.**
- [8] B. R. L. Galvão, A. J. C. Varandas, J. P. Braga, J. C. Belchior. *J. Phys. Chem. Lett.* **2013**, 4, 2292-2297.
- [9] P. J. S. B. Caridade, B. R. L. Galvão, A. J. C. Varandas. *J. Phys Chem. A.* **2010**, 114, 6063-6070.
- [10] J. M. C. Marques, A. Riganelli, A. J. C. Varandas. *Quim. Nova.* **2003**, 26, 5, 769-778.
- [11] W. L. Hase, R. J. Wolf, C. S. Sloane. *J. Chem. Phys.* **1979**, 71, 7, 2910-2928.
- [12] D. Beeman. *J. Comp. Phys.* **1976**, 20, 130-139.
- [13] M. Pu, T. Privalov. *J. Chem Phys.* **2013**, 138, 154305.
- [14] W. L. Hase et. al. *J. Phys. Chem. A.* **2014**, 118, 2228-2236.
- [15] D. A. Bonhommeau, M. P. Gageot. *Comp. Phys. Com.* **2014**, 185, 684-694.
- [16] S. Toxvaerd. *J. Chem Phys.* **2013**, 139, 224106.
- [17] E. Everhart. *Cel. Mech.* **1974**, 10, 35-55.
- [18] K. Bolton, S. Nordholm. *J. Comp. Phys.* **1994**, 113, 320.

- [19] I. Alolyan, T. E. Simos. *J. Math. Chem.* **2014**, 52, 917-947.
- [20] Q. Li, L. Pareschi. *J. Comp. Phys.* **2014**, 259, 402-420.
- [21] V. T. Luan, A. Ostermann. *J. Comp. Appl. Math.* **2014**, 256, 168-179.
- [22] C. Schlier, A. Seiter. *J. Phys. Chem. A.* **1998**, 102, 9399-9404.
- [23] J. Xie, S. C. Kohale, W. L. Hase, S. G. Ard, J. J. Melko, N. S. Shuman, A. A. Viggiano. *J. Phys. Chem. A.* **2013**, 117, 14019-14027.

Capítulo 3: Estrutura eletrônica

3.1 - Equação de Schrödinger

Todas as propriedades de um dado sistema são descritas pela sua função de onda. Realizar uma descrição teórica de muitos problemas em química é equivalente a resolver a equação de Schrödinger não relativística e independente do tempo:

$$\hat{H}\psi = E\psi \quad (3.1)$$

onde \hat{H} é o hamiltoniano de um sistema contendo M núcleos e N elétrons, os quais são descritos pelos vetores posição \mathbf{R}_A e \mathbf{r}_i respectivamente. A distância entre o elétron i e o núcleo A pode ser obtida por $r_{iA}=|\mathbf{r}_i - \mathbf{r}_A|$. A distância entre o elétron i e o elétron j é definida como $r_{ij}=|\mathbf{r}_i-\mathbf{r}_j|$, e a distância entre o núcleo A e o núcleo B é $R_{AB}=|\mathbf{R}_A-\mathbf{R}_B|$. Em unidades atômicas o hamiltoniano pode ser apresentado como:

$$\hat{H} = - \sum_{i=1}^N \frac{1}{2} \nabla_i^2 - \sum_{A=1}^M \frac{1}{2M_A} \nabla_A^2 - \sum_{i=1}^N \sum_{A=1}^M \frac{Z_A}{r_{iA}} + \sum_{i=1}^N \sum_{j>i}^N \frac{1}{r_{ij}} + \sum_{A=1}^M \sum_{B>A}^M \frac{Z_A Z_B}{R_{AB}} \quad (3.2)$$

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \quad (3.3)$$

sendo M_A , a massa do núcleo A em unidades atômicas e Z_A o número atômico do núcleo A . O laplaciano corresponde a energia cinética do sistema. O terceiro termo representa a atração de Coulomb entre os elétrons e o núcleo. O quarto e quinto termos representam a repulsão entre os elétrons e entre os núcleos, respectivamente.

Na aproximação de Born-Oppenheimer os elétrons se movem em um campo médio produzido pelos núcleos fixos. O próton tem massa da ordem de 2000 vezes maior que o elétron. Em uma explicação qualitativa dessa aproximação, pode-se imaginar que o núcleo, em grande parte, composto por prótons, se move muito lentamente comparado aos elétrons. Na

aproximação de Born Oppenheimer, os elétrons atingem a configuração mais estável em um tempo tão curto que durante esse processo considera-se que os núcleos estejam parados. Por outro lado, os núcleos se movem em uma superfície eletrônica, obtida a partir da solução do hamiltoniano eletrônico em cada ponto do espaço.

$$\hat{H}_{elec} = - \sum_{i=1}^N \frac{1}{2} \nabla_i^2 - \sum_{i=1}^N \sum_{A=1}^M \frac{Z_A}{r_{iA}} + \sum_{i=1}^N \sum_{j>i}^N \frac{1}{r_{ij}} \quad (3.4)$$

A solução da equação de Schrödinger envolvendo o hamiltoniano eletrônico é dada por:

$$\hat{H}_{elec} \psi_{elec}(r_i; R_A) = E_{elec} \psi_{elec}(r_i; R_A) \quad (3.5)$$

a função de onda depende das coordenadas dos elétrons r_i enquanto a posição do núcleo R_A é apenas um parâmetro. Os elétrons são férmions, então a função de onda que os descreve precisa ser anti-simétrica, isso pode ser obtido através do determinante de Slater:

$$\Phi(x_1, x_2, \dots, x_N) = \frac{1}{\sqrt{N!}} \begin{vmatrix} \psi_i(x_1) & \psi_j(x_1) & \dots & \psi_k(x_1) \\ \psi_i(x_2) & \psi_j(x_2) & \dots & \psi_k(x_2) \\ \dots & \dots & \dots & \dots \\ \psi_i(x_N) & \psi_j(x_N) & \dots & \psi_k(x_N) \end{vmatrix} \quad (3.6)$$

$$\psi_i(x_1) = \chi_i(x_1) \alpha_i(x_1) \quad (3.7)$$

onde χ_i é a função de onda espacial e α_i uma função de onda que descreve o spin, característica intrínseca dos elétrons. Nos métodos descritos neste trabalho, cada orbital (função de onda espacial) será ocupado por dois elétrons de spins opostos. Sistemas conhecidos como camada fechada. Então, por simplicidade, a função de onda de spin será omitida no restante do texto.

3.2 - Princípio variacional

A equação de Schrodinger possui solução analítica apenas para poucos sistemas. Métodos aproximados são necessários para sistemas químicos, entre eles está o método

variacional. Seja E_0 a energia do estado fundamental e f uma função qualquer normalizada, sabe-se que:

$$E_f = \langle f | \hat{H} | f \rangle \quad (3.8)$$

$$E_f \geq E_0 \quad (3.9)$$

o valor médio da energia no estado fundamental (E_0) será sempre menor que o obtido caso qualquer outra função seja utilizada (E_f).

Prova. O conjunto de autofunções de um operador hermitiano formam uma base completa para o espaço. Resultado proveniente da álgebra linear. Sabe-se que o operador hamiltoniano que descreve um sistema físico é sempre hermitiano. Seja ψ_N as autofunções do operador hamiltoniano, a seguinte expressão é válida para qualquer função do espaço:

$$f = \sum_{n=0} c_n \psi_n \quad ; \quad \hat{H} \psi_n = E_n \psi_n \quad (3.10)$$

é sempre possível normalizar a função f assim como construir uma base ortonormal com as autofunções do operador hamiltoniano. Usando (3.10) têm-se:

$$\langle f | \hat{H} | f \rangle = \langle \sum_{m=0} c_m \psi_m | \hat{H} | \sum_{n=0} c_n \psi_n \rangle \quad (3.11)$$

como a base é ortonormal vale a relação:

$$\langle \psi_m | \psi_n \rangle = 0 \quad ; \quad m \neq n \quad (3.12)$$

usando (3.10) têm-se:

$$\langle f | \hat{H} | f \rangle = \sum_{n=0} E_n |c_n|^2 \quad (3.13)$$

onde E_0 ($n=0$) é por definição o menor valor entre as energias. Por isso, caso todos os E_n da expressão (3.13) sejam substituídos por E_0 , o resultado será sempre menor ou igual à expressão (3.13):

$$\sum_{n=0} E_n |c_n|^2 \geq E_0 \sum_{n=0} |c_n|^2 \quad (3.14)$$

mas a função f está normalizada:

$$1 = \langle f | f \rangle = \sum_{n=0} |c_n|^2 \quad (3.15)$$

usando (3.13), (3.14) e (3.15) têm-se:

$$\langle f | \hat{H} | f \rangle \geq E_0 \quad (3.16)$$

que é o que queríamos demonstrar.

O método variacional consiste em realizar uma busca por funções que não sejam necessariamente autofunções do operador hamiltoniano, mas que minimizam a energia. Quanto menor a energia, mais próxima a função estará do estado fundamental.

Dentre as funções tentativa usadas em química teórica, a mais comum para problemas moleculares são do tipo gaussiana:

$$f = (x - x_0)^{l_1} (y - y_0)^{l_2} (z - z_0)^{l_3} e^{-\alpha(r-r_0)^2} \quad (3.17)$$

onde r_0 é onde a função está centrada, l_i o momento angular e α um coeficiente que define a gaussiana. Existem conjuntos de coeficientes ajustados para determinados sistemas, esses são chamados de bases. Por exemplo, a base 3-21g do átomo de hidrogênio centrado na origem [1] é composta por duas funções (as constantes foram arredondadas por motivo de concisão):

$$f_1 = 0.16e^{-5.4r^2} + 0.90e^{-0.82r^2} \quad (3.18)$$

$$f_2 = e^{-0.18r^2} \quad (3.19)$$

a anti-simetria da função de onda é uma propriedade aplicável quando mais de uma partícula indistinguível está envolvida. Como o átomo de hidrogênio possui apenas um elétron, não é aplicável nesse caso. A função de onda será escrita como uma combinação linear das funções de base:

$$\psi = k_1 f_1 + k_2 f_2 \quad (3.20)$$

o método Hartree Fock, de forma muito resumida, consiste em encontrar os coeficientes k_1 e k_2 para que o sistema possua a menor energia possível. Justificado pelo princípio variacional, essa energia é a que mais se aproxima do estado fundamental. Usando o software GAMESS [2], obtêm-se os seguintes coeficientes para o hidrogênio atômico: $k_1=0,373$, $k_2=0,717$. Essa função corresponde à energia -0.496 Hartree. O átomo de hidrogênio possui solução analítica, a energia do estado fundamental é -0.500 Hartree, muito próxima da obtida pelo princípio variacional.

3.3 - Método Hartree Fock

De forma geral a função de onda eletrônica precisaria ser escrita como uma combinação linear de vários determinantes de Slater, assim seria possível levar em consideração a interação entre vários estados eletrônicos. O método de Hartree Fock consiste em aplicar o princípio variacional em apenas um determinante. A partir de (3.6) têm-se:

$$E = \langle \Phi | H_{ele} | \Phi \rangle \quad (3.21)$$

depois de algumas transformações matemáticas [3], (3.21) adquire o seguinte formato:

$$E = \langle \psi_i | \sum_{i=1}^N -\frac{1}{2} \nabla_i^2 - \sum_{i=1}^N \sum_{A=1}^M \frac{Z_A}{r_{iA}} | \psi_i \rangle + \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \langle \psi_i | \langle \psi_j | \frac{1}{r_{ij}} | \psi_j \rangle | \psi_i \rangle - \langle \psi_i | \langle \psi_j | \frac{1}{r_{ij}} | \psi_i \rangle | \psi_j \rangle \quad (3.22)$$

algumas definições fazem-se necessárias:

$$\hat{h}(i) = -\frac{1}{2} \nabla_i^2 - \sum_{A=1}^M \frac{Z_A}{r_{iA}} \quad (3.23)$$

$$\hat{J}_j |\psi_i\rangle = \langle \psi_j | \frac{1}{r_{ij}} | \psi_j \rangle |\psi_i\rangle \quad (3.24)$$

$$\hat{K}_j |\psi_i\rangle = \langle \psi_j | \frac{1}{r_{ij}} | \psi_i \rangle |\psi_j\rangle \quad (3.25)$$

o operador \hat{h} é chamado operador de um elétron, \hat{J} é o operador de coulomb e \hat{K} é o operador de troca. A partir dessas relações se define o operador de Fock:

$$\hat{F}_i = \hat{h}(i) + \sum_j \hat{J}_j - \hat{K}_j \quad (3.26)$$

o somatório no lado esquerdo de (3.26) é conhecido como potencial de Hartree Fock, pois é o termo que descreve a interação do elétron com o campo médio gerado por todos os outros. A função de onda pode ser encontrada, após algumas aproximações, a partir da seguinte equação de autovalor:

$$\hat{F}_i |\psi_i\rangle = \varepsilon_i |\psi_i\rangle \quad (3.27)$$

embora a última equação possa ser escrita como um problema de autovalor linear, o operador de Fock tem uma dependência dos orbitais ψ_i , por isso é necessário um método iterativo para resolver esse problema.

No operador de Fock (3.26), cada elétron experimenta um campo formado por todos os demais elétrons. Isso faz com que a interação entre os elétrons seja contada duas vezes. Por isso, após a obtenção das autofunções ψ_i , essa energia é recalculada com o hamiltoniano correto:

$$E_{HF} = \langle \Phi_{HF} | \hat{H}_{ele} | \Phi_{HF} \rangle \quad (3.28)$$

a partir do princípio variacional pode-se garantir que a energia obtida em (3.28) é maior ou igual à energia do estado fundamental.

3.4 - Moller Plesset de segunda ordem (MP2)

Seja \hat{H} um hamiltoniano que possa ser decomposto em $H^o + V$. Se $\langle V \rangle$ for muito pequeno, pode-se considerar que esse operador é uma perturbação do sistema. A teoria de perturbação em química quântica é um procedimento sistemático que permite obter uma solução aproximada para um problema que possui uma perturbação.

A teoria de perturbação aplicada a moléculas e que é comumente utilizada foi desenvolvida por Moller e Plesset. Consiste em considerar o hamiltoniano não perturbado H^o como o operador de Fock. E a perturbação, uma correção que remove a repetição na contagem de elétrons:

$$H^o = \sum_i \hat{F}_i \quad (3.29)$$

$$V = \sum_i^{oc} \sum_{j>i}^{oc} \frac{1}{r_{ij}} - \sum_i^{oc} \sum_j^{oc} (J_{ij} - \frac{1}{2}K_{ij}) \quad (3.30)$$

é fácil verificar que a soma de (3.29) com (3.30) resulta no hamiltoniano exato \hat{H} . A precisão desse método depende em grande parte do quanto a perturbação é pequena. O método Moller Plesset é uma forma aproximada de resolver o hamiltoniano exato.

Na teoria de perturbação existem correções em várias ordens. A energia de ordem zero é a solução para o hamiltoniano não perturbado. A correção de ordem um é o valor esperado da perturbação usando a função de onda de ordem zero:

$$E^1 = \langle \Phi_{HF} | H^o | \Phi_{HF} \rangle + \langle \Phi_{HF} | V | \Phi_{HF} \rangle \quad (3.31)$$

mas como a soma do hamiltoniano não perturbado e sua perturbação resulta no hamiltoniano original temos:

$$E^1 = \langle \Phi_{HF} | \hat{H} | \Phi_{HF} \rangle \quad (3.32)$$

mas como demonstrado anteriormente (3.28), essa é a energia do método Hartree Fock. Ou seja, a correção em primeira ordem no método Moller Plesset é equivalente ao método Hartree Fock. A correção em segunda ordem é mais complexa, abaixo está apenas a forma final [4]:

$$E^2 = \sum_i^{oc} \sum_{j>i}^{oc} \sum_a^{vir} \sum_{b>a}^{vir} \frac{[(ij|ab) - (ia|jb)]^2}{\varepsilon_i + \varepsilon_j - \varepsilon_a - \varepsilon_b} \quad (3.33)$$

$$(ij|ab) = \langle \psi_i | \langle \psi_j | \frac{1}{r} | \psi_a \rangle | \psi_b \rangle \quad (3.34)$$

o método Moller Plesset consiste em resolver o hamiltoniano exato de forma aproximada, a saber, usando teoria de perturbação. Neste trabalho foi utilizada até segunda ordem, como apresentado em (3.33).

Neste capítulo foi apresentado um breve resumo sobre os métodos de estrutura eletrônica relevantes para este trabalho. Mais informações podem ser obtidas, por exemplo, nos seguintes livros: [3-6].

No próximo capítulo será apresentado o algoritmo DRKAI, seu funcionamento e características principais.

3.5 – Referências Bibliográficas

- [1] J.S. Binkley, J.A. Pople, W.J. Hehre, J. Am. Chem. Soc. **1980**, 102, 939.
- [2] M. W. Schmidt, K. K. Baldrige, J. A. Boatz, S. T. Elbert, M. S. Gordon, J. H. Jensen, S. Koseki, N. Matsunaga, K. A. Nguyen, S. J. Su, T. L. Windus, M. Dupuis, J. A. Montgomery. J. Comp. Chem. **1993**, 14, 1347.
- [3] D. J. Griffiths. Introduction to quantum mechanics. Pearson, Upper Saddle River, **2005**.
- [4] C. J. Cramer. Essentials of Computational Chemistry: theories and models. Wiley, Sussex, **2004**.
- [5] F. Jensen, J. Introduction to Computational Chemistry. Wiley, Sussex, **1999**.

[6] A. Szabo, N. S. Ostlund. Modern Quantum Chemistry: An Introduction to Advanced Electronic Structure Theory, Macmillan, New York, **1982**.

Capítulo 4: Dinâmica Runge Kutta *ab initio*

4.1 – O programa

O DRKAI (Dinâmica Runge Kutta *ab initio*) é um software de dinâmica molecular direta. O programa realiza uma integração numérica das equações de movimento clássicas, e pode ou não utilizar forças a partir de métodos *ab initio*. Nesse momento, está linkado ao pacote GAMESS-US [1], e usa esse pacote para os cálculos quânticos. A figura 4.1 ilustra o comportamento do DRKAI.

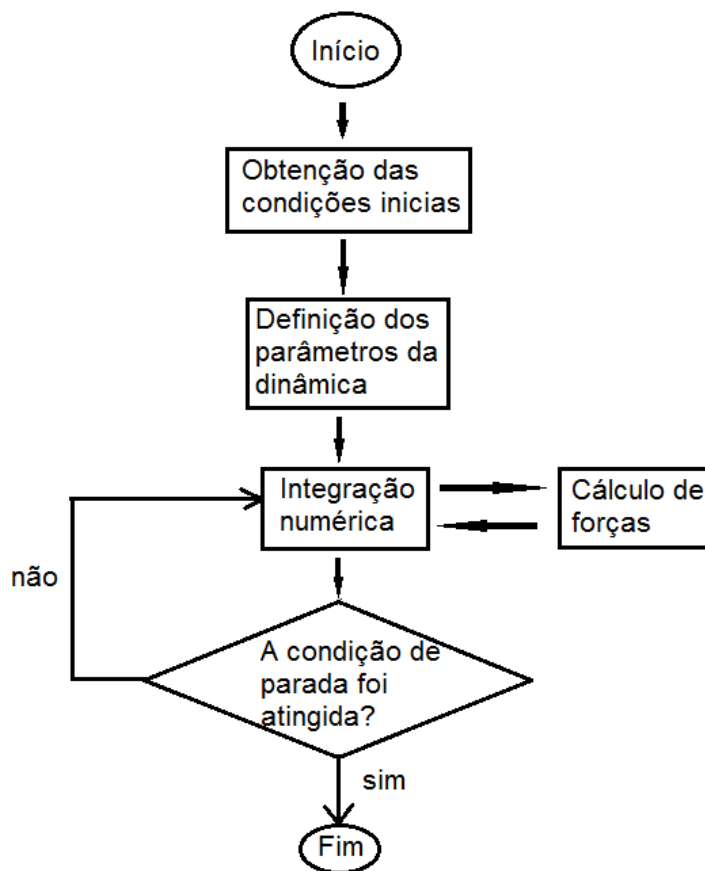


Figura 4.1 - Fluxograma do software DRKAI. Mostra os principais passos da dinâmica molecular *ab initio*.

As condições iniciais podem ser inseridas de duas formas. A primeira é feita pelo usuário, embora condições iniciais manuais, em geral, possuam a desvantagem de não atender as regras da mecânica estatística, esse método permite que o usuário tenha uma ideia geral sobre o comportamento do sistema, antes de se dedicar a uma dinâmica longa. A outra maneira de se obter condições iniciais está disponível apenas para moléculas poliatômicas, e será descrita no capítulo 5.

Além das condições iniciais, o usuário precisa definir o integrador a ser utilizado. Isso é feito através de um índice que corresponde a seguinte tabela:

Tabela 4.1 - Índice dos integradores no software DRKAI (Dinâmica Runge Kutta ab initio)

Índice	Integrador
1	Runge Kutta Nystron
2	Beeman
3	Gauss Radau
4	Runge Kutta simétrico (4ª ordem)
5	Runge Kutta simétrico (6ª ordem)
6	Runge Kutta padrão (4ª ordem)
7	Runge Kutta Gill
8	Adams Bashforth Moulton

Pela tabela, observa-se que existem oito integradores implementados. Todos obtiveram resultados consistentes com suas capacidades em testes preliminares, mas apenas os descritos neste trabalho foram exaustivamente testados.

Também é necessário definir o tempo total da trajetória, única condição de parada disponível. Em geral, nos softwares populares, a condição de parada é definida a partir das distâncias dos átomos, quando atingem um valor pré-estabelecido, significa que houve uma reação ou espalhamento. No DRKAI essa opção ainda não está disponível. A trajetória termina após atingir um tempo determinado pelo usuário, 100 fs por exemplo. Esse tempo é facilmente obtido após alguns testes. O usuário também precisa definir o passo da integração, que é particular para cada integrador. A discussão do melhor passo a ser escolhido é feita no capítulo 5.

É necessário definir o número atômico de cada átomo, mas a massa, até o presente momento, sempre será a do isótopo mais estável. Caso o usuário deseje realizar uma dinâmica *ab initio*, é necessário construir o cabeçalho de um input do GAMESS. As únicas restrições na

construção do input são os métodos, apenas MP2 e Hartree Fock estão disponíveis. No grupo \$CONTRL é importante requisitar o cálculo do gradiente. Todas as demais funções que podem ser obtidas pelo cabeçalho estão disponíveis, por exemplo: convergência do SCF, uso de solventes do tipo PCM, cálculos em paralelo e todas as bases fornecidas pelo programa. Para mais informações sobre as funcionalidades do GAMESS, consultar o manual do software [1].

A integração numérica é feita através dos algoritmos discutidos no capítulo 2. Além das forças obtidas por métodos *ab initio*, o software possui um potencial de força central, discutido na seção 5.2 e um potencial empírico do tipo Morse. Entretanto, antes de utilizar essas funções, é necessário entrar em contato com o desenvolvedor.

A simulação gera três arquivos de saída com as seguintes extensões: “.csv”, “.xyz” e “.log”. Na extensão “.csv” se encontra a energia total, a energia potencial obtida pelo GAMESS e o espaço de fase a cada passo da simulação. O arquivo com a extensão “.xyz” possui apenas as coordenadas, o formato é facilmente lido por visualizadores de moléculas tradicionais. O arquivo “.log” contém informações gerais da dinâmica, como o integrador utilizado e o tempo de CPU. Não confundir tempo de trajetória com tempo de CPU. O primeiro se refere a um tempo em femtossegundos definido pelo usuário, e está relacionado à escala atômica. O tempo de CPU é o quanto o computador demora para realizar todas as operações matemáticas da trajetória. O tempo de CPU costuma estar em torno de 30 min. O tempo da trajetória costuma variar entre 100-400 fs.

4.2 – Teste do algoritmo: Colisão entre duas partículas

Seja um sistema de duas partículas sujeitas a um potencial Morse (4.1), a colisão frontal entre elas possui um ponto característico chamado ponto de retorno clássico.

$$V(r) = D_e(1 - e^{-\alpha(r-R_e)})^2 - D_e \quad (4.1)$$

O ponto de retorno ocorre quando toda a energia cinética se converte em energia potencial. Essa é a região de máxima aproximação entre as partículas. A figura 4.2 ilustra três momentos desse tipo de trajetória. Na figura o índice 'a' representa o início da trajetória, o índice 'b' representa o ponto de retorno e o índice 'c' representa o sistema alguns instantes após o ponto de retorno.

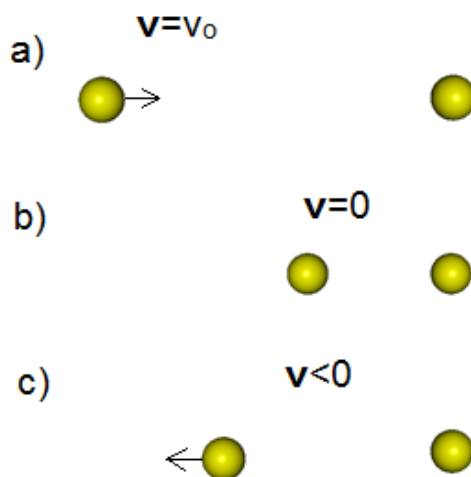


Figura 4.2 – Representação do ponto de retorno clássico. Em 'a' têm-se o início da trajetória, em 'b' apresenta-se o ponto de retorno e em 'c' está representado alguns instantes após o ponto de retorno.

Uma trajetória foi integrada para verificar se o programa era capaz de verificar essa grandeza intrínseca do sistema. A tabela 4.2 mostra as condições iniciais e os parâmetros do potencial Morse utilizados nessa tarefa.

Tabela 4.2 – Parâmetros referentes ao potencial Morse (4.1) e condições iniciais para uma colisão frontal entre duas partículas que interagem por esse potencial

Parâmetro	Valor (ua)
A	1,0290
D_e	0,1745
R_e	1,4011
$\mathbf{v}_{inicial}$	1,0000
$\mathbf{r}_{inicial}$	20,0000

Para alguns sistemas, o ponto de retorno pode ser obtido de forma analítica, definido pela equação (4.2).

$$V(r_{pr}) - E = 0 \quad (4.2)$$

Substituindo (4.1) em (4.2) têm-se:

$$r_{pr} = -\frac{1}{\alpha} \ln \left(1 + \sqrt{\frac{E + D_e}{D_e}} \right) + R_e \quad (4.3)$$

onde r_{pr} é o ponto de retorno clássico. Para as condições da tabela 4.2 o valor da expressão (4.3) é $r_{pr} = 0,157670 \text{ Bohr}$.

O gráfico abaixo mostra o resultado da integração numérica usando o integrador Runge Kutta simétrico. As condições iniciais são as expressas na tabela 4.2 com um passo de 0,0001 ua.

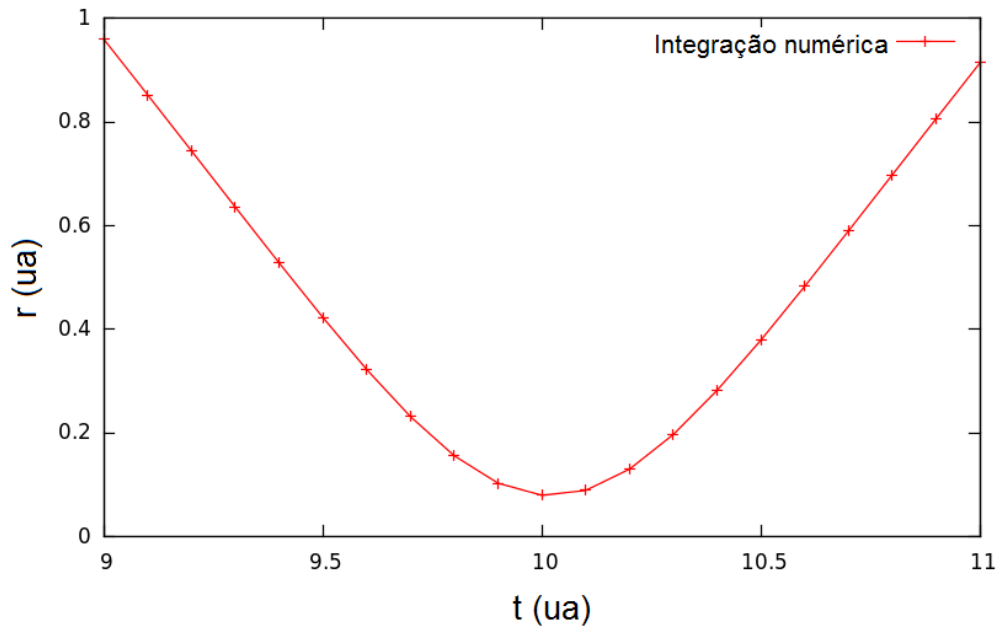


Figura 4.3 – Resultado da integração numérica de duas partículas descritas por um potencial Morse realizando uma colisão frontal. Os parâmetros da simulação estão apresentados na tabela 4.2. O foco do gráfico está na região próxima ao ponto de retorno clássico.

Observa-se que a integração numérica descreveu corretamente os aspectos qualitativos dessa trajetória. O ponto de máxima aproximação obtido a partir da integração numérica foi $r_{pr} = 0,157671 \text{ Bohr}$. Com um erro de $1 \times 10^{-6} \text{ Bohr}$ do resultado analítico, completamente aceitável na precisão desejada.

Esse teste é um forte indicativo de que o programa está corretamente implementado e que consegue descrever com precisão os fenômenos físicos desse sistema.

4.3 - Referências Bibliográficas

[1] M. W. Schmidt, K. K. Baldrige, J. A. Boatz, S. T. Elbert, M. S. Gordon, J. H. Jensen, S. Koseki, N. Matsunaga, K. A. Nguyen, S. J. Su, T. L. Windus, M. Dupuis, J. A. Montgomery. *J. Comp. Chem.* **1993**, 14, 1347.

Capítulo 5: Sistemas modelo e resultados

5.1 Introdução

Devido ao alto custo computacional dos cálculos *ab initio*, desejam-se integradores que preservem os elementos da dinâmica com o maior passo possível [1]. O principal fator de uma boa integração é a conservação de energia. É uma condição necessária, mas não garante que a trajetória esteja correta [2]. Neste trabalho, foram realizados testes de conservação de energia assim como o quanto o sistema desviou da trajetória correta. Apesar das dificuldades de se definir uma trajetória como correta quando o sistema não possui solução analítica, é possível através do uso de um passo muito menor [3].

O erro na energia será definido em todo o texto de acordo com a seguinte equação:

$$\overline{Erro} = \frac{1}{N} \sum_i^N |E_t - E_i| \quad (5.1)$$

onde E_t é a energia total do sistema, E_i é a energia total calculada pelo integrador no passo i , e N é o número de passos.

Neste trabalho os cálculos quânticos foram feitos usando um conjunto de funções gaussianas com coeficientes pré-determinados, chamados bases. Foi utilizada a base TZV (*triple zeta valence*) para a molécula de hidrogênio, que é um conjunto de expoentes e coeficientes de contração encontrados na referência [4]. Para as reações químicas foi utilizada a base 3-21g, que é o conjunto encontrado na referência [5].

Abaixo serão apresentados os diversos testes realizados neste trabalho. Os resultados não mostram apenas a eficiência dos diversos integradores, mas também a robustez do software DRKAI.

5.2 - Planetário

O sistema, aqui chamado planetário, consiste em uma partícula de massa um, sob efeito de um potencial de força central (5.2).

$$u(\mathbf{r}_i) = \frac{1}{|\mathbf{r}_i|} \quad (5.2)$$

A solução do sistema planetário é analítica e bem conhecida. Para uma condição inicial onde a partícula está a uma unidade de distância do centro de massa. Momento angular inicial igual a uma unidade, o sistema descreve um movimento circular uniforme. A velocidade angular desse movimento é 1 rad/ut (unidade de tempo). Foram utilizadas essas condições iniciais em todos os testes deste trabalho. O tempo total de simulação foi sempre igual a 63 ut. Esse tempo é o equivalente a 10 rotações em torno da origem.

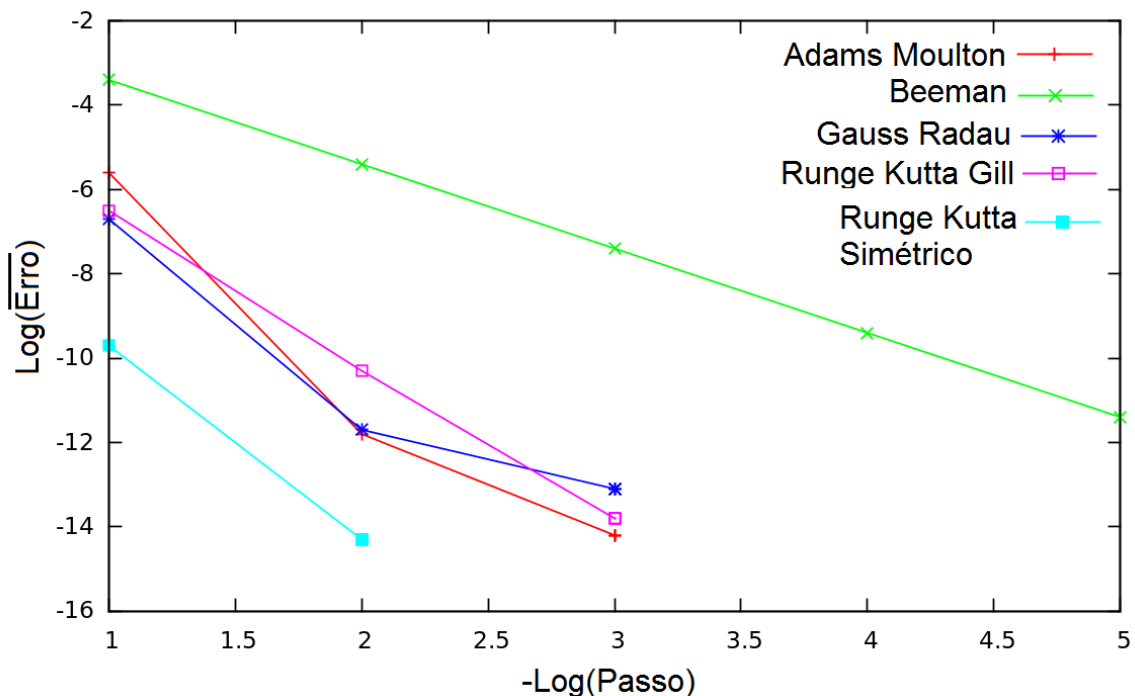


Figura 5.1 - Variação da energia em função do passo para cinco integradores diferentes.

Estudo de uma partícula sob o efeito de uma força central.

A partir desses resultados foi possível observar que o algoritmo Runge Kutta simétrico teve o melhor desempenho, obtendo um erro inferior a 10^{-14} com um passo de 0.01 ut (figura 5.1). Outros integradores chegaram a resultados semelhantes apenas com passos 10 vezes menores (0.001 ut).

O segundo algoritmo mais eficiente foi o Adams Bashforth Moulton, pois teve resultados comparáveis ao Runge Kutta Gill e ao Gauss Radau, necessitando de menos cálculos de força por passo. O algoritmo menos eficiente para esse sistema foi o integrador de Beeman, estimando a partir da figura 5.1, o algoritmo de Beeman necessitaria de um tempo mil vezes maior para se obter o mesmo resultado do que obtido pelo Runge Kutta simétrico. O gráfico abaixo mostra o quanto a posição da partícula se desviou da solução analítica.

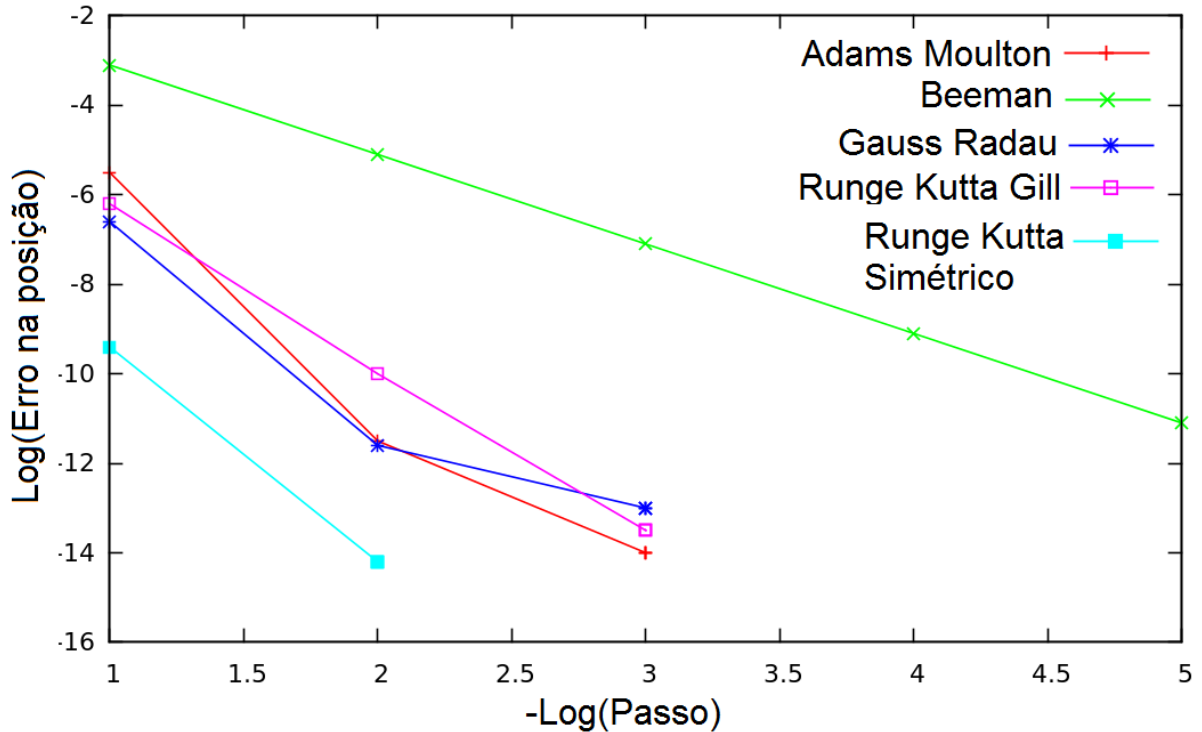


Figura 5.2. Variação da posição da partícula obtida pelos integradores em relação à posição obtida pela solução analítica da equação diferencial. Estudo de uma partícula sob o efeito de uma força central.

Para esse sistema o erro na posição teve a mesma ordem de grandeza do que o erro na energia. Pode-se observar que o integrador mais preciso foi o Runge Kutta simétrico, e o que mais se afastou da solução analítica foi o Beeman.

Para a equação diferencial planetária, a integração numérica demonstra-se eficiente não apenas na conservação dos elementos da dinâmica. A figura 5.2 mostra que há um grande acordo entre a posição exata e a obtida pelo integrador. Mostrando que o erro na energia é um bom parâmetro para avaliar a eficiência do método nesse caso.

Esse resultado é um forte indicativo de que o software DRKAI foi corretamente implementado, pois obteve um resultado condizente com o que se espera dos integradores numéricos, e correto no ponto de vista físico.

5.3 - Vibração da molécula de H₂

A molécula de hidrogênio foi estudada com dinâmica molecular *ab initio* direta. As condições iniciais foram obtidas pelo método da amostragem normal, assim como no pacote VENUS96 [6]. Nessa abordagem, após a definição da energia, nesse caso, o estado fundamental. As coordenadas são geradas considerando que o sistema se comporte como um oscilador harmônico simples. Definida a energia total e a frequência angular, a amplitude clássica provém da expressão (5.3).

$$A = \frac{1}{\omega} \sqrt{\frac{2E}{m}} \quad (5.3)$$

Neste trabalho, a frequência foi obtida com o método MP2 e base TZV. Todos os cálculos quânticos deste trabalho foram realizados com o software GAMESS. A trajetória foi iniciada com a molécula em repouso na amplitude máxima. Todas as informações necessárias para a obtenção das condições iniciais estão presentes na tabela 5.1.

Tabela 5.1: Informações necessárias para a obtenção das condições iniciais. O sistema é uma molécula de hidrogênio no nível MP2 e base TZV

Frequência (cm ⁻¹)	Massa (amu)	Amplitude (Å)	Distância de equilíbrio (Å)
4460,33	1,00782	0,122	0,737

Simulações foram realizadas no nível MP2 com base TZV. As trajetórias se iniciaram no mesmo ponto, diferindo apenas no integrador e no passo. Esse foi variado de 3.5 fs a 0.01 fs, de acordo com as necessidades do integrador. O tempo total da trajetória foi 50 fs. Para cada integrador foram feitas em torno de quatro trajetórias.

O passo de integração foi variado de forma a obter erros entre 10⁻³ e 10⁻⁷ Hartree. Os resultados desses testes estão apresentados no gráfico abaixo.

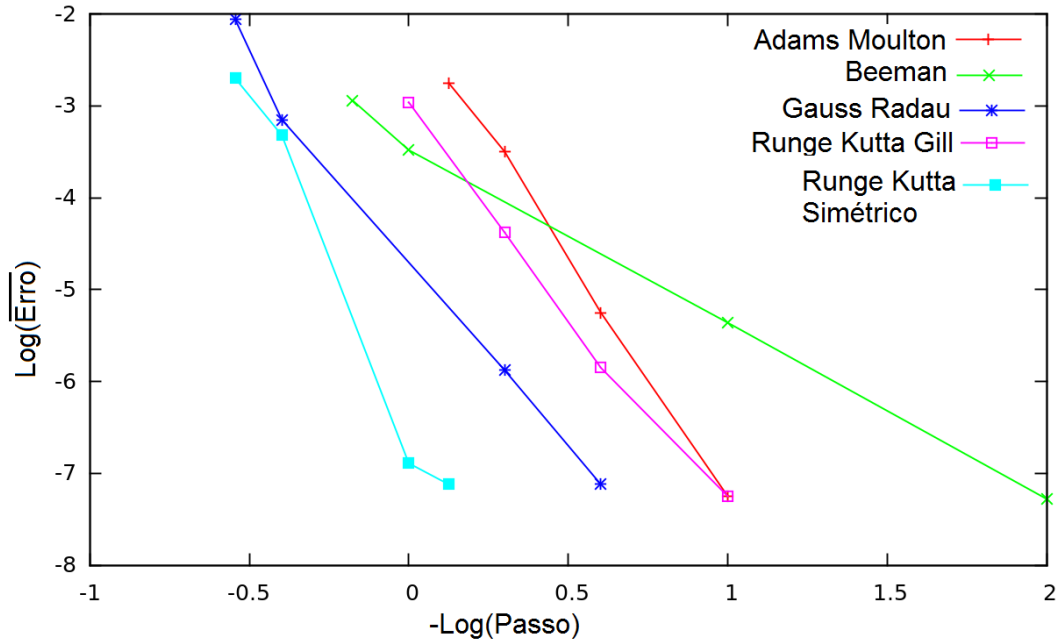


Figura 5.3 - Variação da energia em função do passo para cinco integradores diferentes. Estudo da molécula de hidrogênio no nível MP2 e base TZV.

A performance do algoritmo de Beeman foi surpreendente para esse sistema. Apesar de necessitar de um passo muito menor para atingir o limite inferior, 0.01 fs. Obteve ótima performance com um passo de 1.0 fs. Na seção seguinte, um sistema mais complexo foi estudado, permitindo uma avaliação mais precisa dos integradores.

5.4 - Moléculas poliatômicas

A obtenção das condições iniciais em um sistema contendo moléculas poliatômicas é ligeiramente mais complexo. A seção seguinte apresenta de forma geral como isso é feito.

5.4.1 - Amostragem de moléculas poliatômicas

O que define as características de uma partícula em mecânica quântica é a função de onda. Nessa abordagem, não se pode definir a posição e a velocidade exatas do corpo. Entretanto, em mecânica clássica, é necessário atribuir valores para posições e velocidades. Para o caso do oscilador harmônico, mesmo definindo a energia, é necessário apontar em qual ponto do potencial a partícula se encontra e qual a sua velocidade.

Neste trabalho, foi realizado um estudo clássico das trajetórias. Apesar de atribuir o estado vibracional fundamental a todas as moléculas, foi necessário gerar fases diferentes para as vibrações. O processo descrito a seguir foi implementado com as mesmas equações de uma das opções do software VENUS96.

Para gerar as coordenadas dos modos normais é necessário um cálculo da hessiana, isso foi realizado com o software GAMESS. Seja ω a frequência do modo normal i , e considerando que o sistema está no estado fundamental, a energia é obtida a partir da seguinte expressão:

$$E_o = \frac{1}{2} \hbar \omega \quad (5.4)$$

e a amplitude do deslocamento vem do oscilador harmônico clássico (5.5).

$$A = \frac{\sqrt{2E_o}}{\omega} \quad (5.5)$$

A massa foi omitida na equação (5.5). Deveria ser um valor que corresponde a inércia do modo normal em questão, no caso de duas partículas, seria a massa reduzida. Esse valor está incluso na matriz \mathbf{L}_x , que converte as coordenadas dos modos normais em coordenadas cartesianas. Essa matriz foi obtida pelo pacote GAMESS ao se realizar um cálculo da hessiana.

Um número aleatório R_i irá definir a fase do sistema, a partir dele as coordenadas dos modos normais são geradas a partir das equações de um oscilador harmônico simples (5.6) e (5.7).

$$Q_i = A_i \cos(2\pi R_i) \quad (5.6)$$

$$\dot{Q}_i = -\omega_i A_i \sin(2\pi R_i) \quad (5.7)$$

Para converter em coordenadas cartesianas, a matriz L_x foi utilizada. As seguintes equações formalizam a conversão:

$$X = L_x Q + X_o \quad (5.8)$$

$$P_x = M L_x \dot{Q} \quad (5.9)$$

onde M é uma matriz diagonal com as massas dos átomos. Na atual implementação, os momentos são substituídos por velocidades, então a expressão (5.9) toma a forma simples (5.10).

$$V_x = L_x \dot{Q} \quad (5.10)$$

Esse método gera um momento angular espúrio J_s [6]. Em todas as simulações desse trabalho o momento angular foi definido como zero, por isso, o movimento de rotação é corrigido. Seja w um vetor que define a velocidade angular, o corpo gira em torno de w como descrito pela figura 5.4.

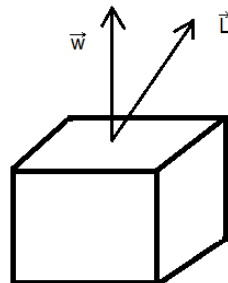


Figura 5.4 - Corpo girando em torno de w . L é o momento angular, sabe-se que nem sempre esse aponta na direção de w .

A expressão que relaciona o momento angular e a velocidade angular é:

$$w = I^{-1}J \quad (5.11)$$

onde I^{-1} é o inverso do tensor de inércia. Deseja-se remover o J_s que foi gerado no processo de obtenção das condições iniciais, ou seja:

$$J_s + J_x = 0 \quad ; \quad J_s = -J_x \quad (5.12)$$

para isso, basta encontrar o valor do momento angular atual e subtraí-lo do resultado. Mas o sistema não está definido em relação a momentos angulares e sim velocidades, por isso, é necessário passar pela equação (5.11). Na atual implementação, esse processo é feito da seguinte forma:

1. *Cálculo do momento angular usando a fórmula:*

$$L = \sum L_i = \sum r_i \times v_i \quad (5.13)$$

2. *Cálculo do tensor de inércia. É uma matriz quadrada que obedece*

as seguintes fórmulas:

$$I_{xx} = \sum m_i(y_i^2 + z_i^2) \quad (5.14)$$

$$I_{yy} = \sum m_i(x_i^2 + z_i^2) \quad (5.15)$$

$$I_{zz} = \sum m_i(x_i^2 + y_i^2) \quad (5.16)$$

$$I_{xy} = \sum m_i x_i y_i \quad (5.17)$$

$$I_{xz} = \sum m_i x_i z_i \quad (5.18)$$

$$I_{yz} = \sum m_i y_i z_i \quad (5.19)$$

3. *Obtenção do w usando o momento angular calculado em 1, o tensor de inércia calculado em 2 e a expressão (5.11).*

4. Pode-se modificar a velocidade angular de um corpo no centro de massa a partir da seguinte expressão:

$$v_{nova} = v_{antiga} + \omega \times r \quad (5.20)$$

usando (5.20) para aplicar uma velocidade angular oposta à obtida por 3 obtêm-se o momento angular igual a zero, que é o que se desejava.

Em resumo, obter as condições iniciais do sistema implica em usar as expressões (5.6) e (5.7), converter o resultado para coordenadas cartesianas e corrigir o momento angular.

5.5 - Reações estudadas

Neste trabalho foram estudadas moléculas com o potencial para realizar uma substituição química bimolecular. Utilizou-se a técnica de dinâmica molecular *ab initio*. Todos os cálculos quânticos foram feitos com o pacote GAMESS [7], método MP2 e base 321g. Os seguintes sistemas foram abordados:

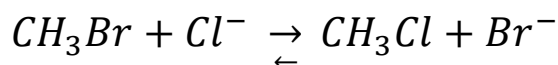
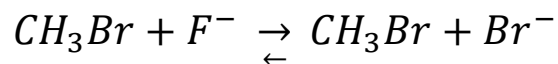
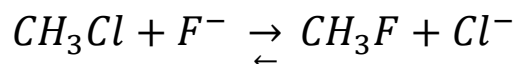


Figura 5.5 – Equações químicas dos sistemas estudados neste trabalho.

Em primeira aproximação, os sistemas foram estudados em fase gasosa. A todos os modos normais foi atribuído o estado fundamental, mas com fases obtidas de forma aleatória. Em seguida o parâmetro de impacto foi definido como $0,5 A$. Seja \mathbf{R} um vetor que une o haleto ao centro da molécula e seja \mathbf{v} o vetor velocidade inicial do haleto. O parâmetro de impacto é definido como a componente de \mathbf{R} perpendicular a \mathbf{v} antes da colisão [8]. A figura 5.6 ilustra essa grandeza.

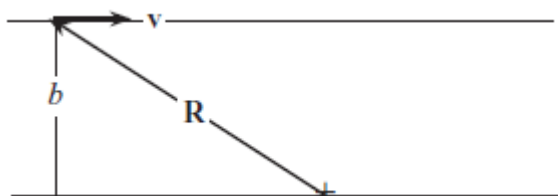


Figura 5.6 – Representação de um parâmetro de impacto (b) genérico. É definido como a componente de \mathbf{R} perpendicular a \mathbf{v} .

A energia cinética translacional inicial foi definida como $3kT/2$, com $T=300K$ ($1,4 \times 10^{-3}$ Hartree). A figura abaixo ilustra um exemplo de condição inicial.

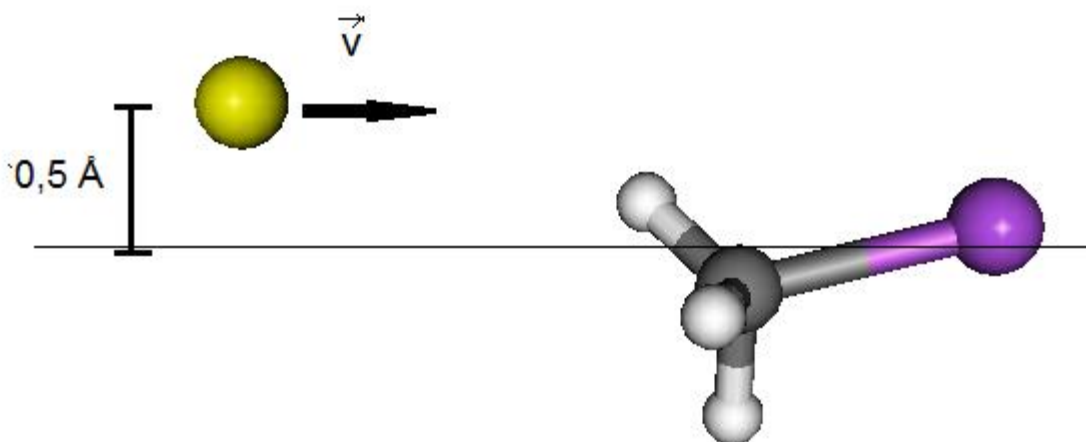


Figura 5.7 – Exemplo de condição inicial. Ataque nucleófilo de um átomo de cloro à molécula CH_3Br . O parâmetro de impacto foi definido como 0,5 Å.

Ao todo, foram geradas cinco trajetórias para cada sistema, resultando em um total de 15 trajetórias diferentes. Cada uma foi conduzida com um dos integradores de forma a se obter um erro médio na energia próximo de 10^{-5} Hartree. Resultando em um total de 75 trajetórias. A tabela abaixo mostra a conservação do centro de massa e a conservação do momento angular.

Tabela 5.2: Conservação do centro de massa e do momento angular para vários integradores. A energia cinética translacional foi definida como $3kT/2$, para uma temperatura de 300K ($1,4 \times 10^{-3}$ Hartree)

Integradores	Erro médio do centro de massa (10^{-9} Å)	Erro do momento angular (10^{-5} Hartree.fs)
Adams Moulton	1,20	2,63
Beeman	5,30	4,11
Gauss Radau	2,60	0,45
Runge Kutta Gill	8,60	0,061
Runge Kutta simétrico	1,90	0,11

Na tabela 5.2 se observa uma conservação do centro de massa na ordem de 10^{-9} Å, que é o limite de casas decimais usadas neste trabalho. A conservação do momento angular foi muito próxima da conservação de energia (10^{-5} Hartree) como era de se esperar.

Cinco trajetórias é um valor muito pequeno para a compreensão do comportamento de um sistema, o objetivo desses testes é avaliar os resultados numéricos e a conservação dos aspectos qualitativos da dinâmica. A escolha do sistema foi feita apenas para ilustrar uma reação química comum. Um estudo mais profundo é necessário caso se deseje resultados comparáveis a experimentos.

5.5.1 - $\text{CH}_3\text{Cl} + \text{F}^-$

É muito difícil determinar as posições corretas quando o sistema não tem solução analítica¹⁸. Por isso, foi gerada uma trajetória adicional para o sistema $\text{CH}_3\text{Cl} + \text{F}^-$. Nessa, o parâmetro de impacto foi definido como zero. O flúor foi iniciado na linha da ligação C-Cl, favorecendo a substituição. Além de realizar a integração dessa trajetória, o sistema foi recalculado com um passo muito menor, 0.25 Å, tarefa realizada usando o algoritmo Runge Kutta simétrico. Essa trajetória com passo menor foi definida como correta, dado que para esse sistema não há solução analítica. A trajetória correta foi utilizada para avaliar o desvio das posições dos átomos. O erro desse desvio é obtido a partir de um cálculo RMSD (*root mean square deviation*) [9] (5.21).

$$\text{RMSD}(v, w) = \sqrt{\frac{1}{n} \sum_{i=1}^n \|v_i - w_i\|^2} \quad (5.21)$$

Um procedimento semelhante foi feito na ref. [3]. É um teste importante, pois se sabe que a conservação na energia não garante que a trajetória esteja correta.

Tabela 5.3: Resultados de uma dinâmica molecular direta do sistema $CH_3Cl + F$. A base utilizada foi a 3-21g, com o método MP2. A energia cinética translacional foi definida como $3kT/2$ para uma temperatura de 300K ($1,4 \times 10^{-3}$ Hartree)

Integradores	Passo (fs)	Erro médio da energia (10^{-5} Hartree)	RMSD*	Número de pontos da superfície calculados
Adams moulton	0,25	1,04	3,99E-06	1208
Beeman	0,25	4,91	2,07E-03	608
Gauss Radau	1,0	9,87	7,22E-05	1366
Runge Kutta Gill	0,25	0,39	9,03E-03	2400
Runge Kutta simétrico	2,5	5,62	1,14E-04	600

Na tabela 5.3 estão apresentados resultados de uma trajetória da reação $CH_3Cl + F$. O passo dos diferentes integradores foi ajustado de forma a se obter a mesma ordem de grandeza no erro da energia. Em um conjunto onde todos apresentam o mesmo erro, o integrador mais eficiente é aquele que realiza esse procedimento em menos tempo. Nesse caso, o tempo é diretamente proporcional ao número de pontos da superfície eletrônica calculados. A partir da tabela 5.3, pode-se concluir que o Runge Kutta simétrico e o Beeman são mais eficientes, pois apresentam o mesmo erro dos demais, no entanto menos pontos da superfície eletrônica foram necessários.

O RMSD mostrou que apesar do algoritmo de Beeman apresentar boa conservação de energia com um grande passo, teve um desvio substancial da trajetória correta. Esse teste sugere que o algoritmo Runge Kutta simétrico seja superior ao Beeman, mesmo ambos possuindo erros na energia e tempos de CPU muito próximos.

A tabela abaixo mostra os resultados das cinco trajetórias geradas com fator de impacto igual a 0,5 Å.

Tabela 5.4: Erro na energia de um grupo de trajetórias para cinco integradores. Descrevem uma dinâmica direta da reação $\text{CH}_3\text{Cl} + \text{F}$

Integrador	Passo (fs)	Erro médio das 5 trajetórias (10^{-5} Hartree)	Número de pontos da superfície calculados
Adams Moulton	0,25	0.65	1208
Beeman	0,25	2.98	608
Gauss Radau	1,00	6.75	1366
Runge Kutta Gill	0,25	0.26	2400
Runge Kutta simétrico	2,50	4.01	600

As cinco trajetórias foram reativas, finalizando completamente a substituição em 150 fs. Novamente os passos foram ajustados de forma a todos os integradores possuírem erros na mesma ordem de grandeza. Os resultados da tabela 5.4 demonstram que os integradores Beeman e Runge Kutta simétrico são seguramente superiores aos demais, em termos de velocidade e conservação de energia. A figura abaixo mostra imagens de uma das cinco trajetórias desse sistema.

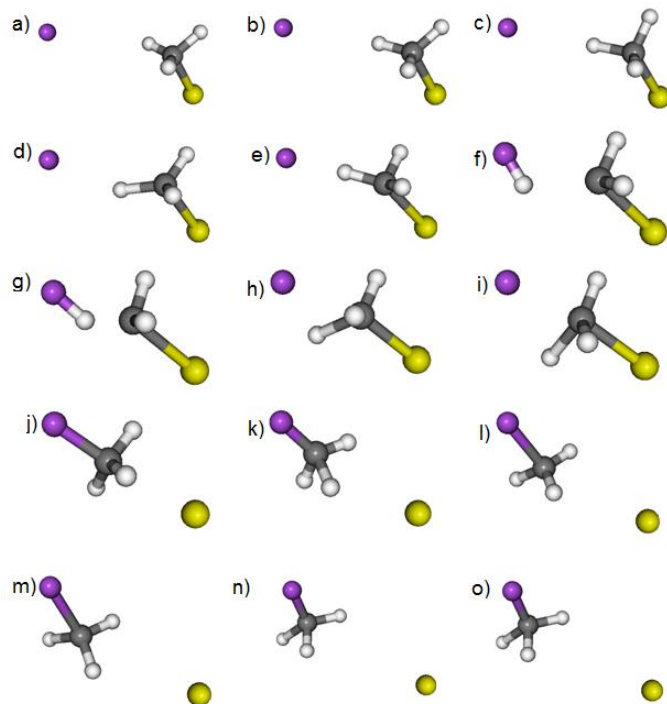


Figura 5.8 – Dinâmica direta da reação S_N2 : $CH_3Cl + F^-$. O átomo no centro do tetraedro é o carbono, a esfera maior ligada ao carbono é o cloro e o nucleófilo, inicialmente distante é o fluoreto. As figuras estão em sequência com uma diferença de 10 fs entre cada trajetória.

A abstração do próton nos itens 'f' e 'g' da figura 5.8 foram geradas pelo visualizador de moléculas (Molekel), esse tem como critério para ligações químicas apenas as distâncias entre os átomos. Na trajetória, essa distância curta surgiu devido à colisão dos dois compostos.

5.5.2 - $CH_3Br + F^-$

A tabela abaixo mostra os resultados das cinco trajetórias geradas desse sistema.

Tabela 5.5: Erro na energia de um grupo de trajetórias para cinco integradores. Descrevem uma dinâmica direta da reação **CH₃Br + F**

Integrador	Passo (fs)	Erro médio das 5 trajetórias (10 ⁻⁵ Hartree)	Número de pontos da superfície calculados
Adams Moulton	0,25	0.50	1208
Beeman	0,25	3.13	608
Gauss Radau	1,00	5.34	1366
Runge Kutta Gill	0,25	0.18	2400
Runge Kutta simétrico	2,50	4.29	600

A trajetória 1 foi reativa obtendo-se o produto CH₃F. As trajetórias 2-4 resultaram em uma abstração de um próton de acordo com a seguinte reação: **CHBr + F⁻ -> CHBr⁻ + HF**. A abstração de próton nessas condições é muito comum se o nucleófilo for o hidróxido (HO⁻), como as duas espécies apresentam algumas semelhanças, esse resultado foi considerado aceitável. Na trajetória de número 5 houve apenas um espalhamento sem ocorrer nenhum tipo de reação química.

A tabela 5.5 foi construída com o mesmo princípio das anteriores, o passo foi ajustado de forma a se obter o mesmo erro em todos os integradores. Novamente os integradores Beeman e Runge Kutta simétrico demonstraram-se superiores aos demais, com diferenças muito pequenas entre si. A figura abaixo mostra imagens da trajetória reativa.

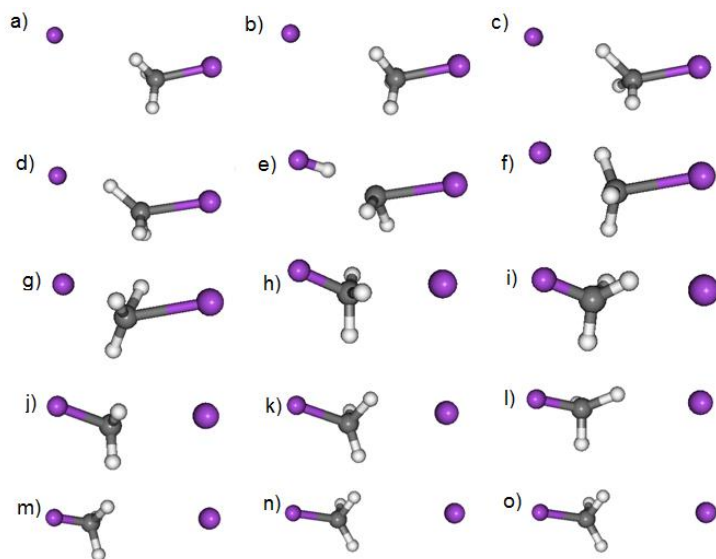


Figura 5.9 – Dinâmica direta da reação S_N2 : $CH_3Br + F$. O átomo no centro do tetraedro é o carbono, a esfera maior ligada ao carbono é o bromo e o nucleófilo, inicialmente distante é o fluoreto. As figuras estão em sequência com uma diferença de 10 fs entre cada trajetória.

A abstração do próton novamente foi gerada pelo visualizador de moléculas, a distância curta é um resultado natural da colisão.

5.5.3 – $CH_3Br + Cl^-$

Nenhuma das cinco trajetórias foi reativa, resultando em apenas um espalhamento. O tempo da trajetória foi aumentado para 400 fs nesse sistema, pois houveram dúvidas se a reação havia acabado no tempo anterior. Mesmo com o aumento do tempo o resultado foi o mesmo, não reativo para todas as trajetórias. Na tabela abaixo estão expressos os erros na energia para esse sistema.

Tabela 5.6: Erro na energia de um grupo de trajetórias para cinco integradores. Descrevem uma dinâmica direta da reação $CH_3Br + Cl$

Integrador	Passo (fs)	Erro médio das 5 trajetórias (10^{-5} Hartree)	Cálculos de força
Adams Moulton	0,25	01.47	3208
Beeman	0,25	02.84	1608
Gauss Radau	1,00	16.20	3616
Runge Kutta Gill	0,25	00.60	6400
Runge Kutta simétrico	2,50	02.61	1600

Observa-se que o erro do integrador Gauss Radau foi muito maior nesse caso. Isso foi devido ao longo tempo de simulação (400 fs). O Gauss Radau não é um algoritmo simplético, isso significa que enquanto os demais provocam oscilações na energia em torno da correta, o Gauss Radau gera um aumento ou decréscimo constante na energia. O resultado da tabela 5.6 sugere que esse algoritmo seja fortemente evitado em simulações longas.

A figura abaixo mostra imagens de uma das trajetórias.

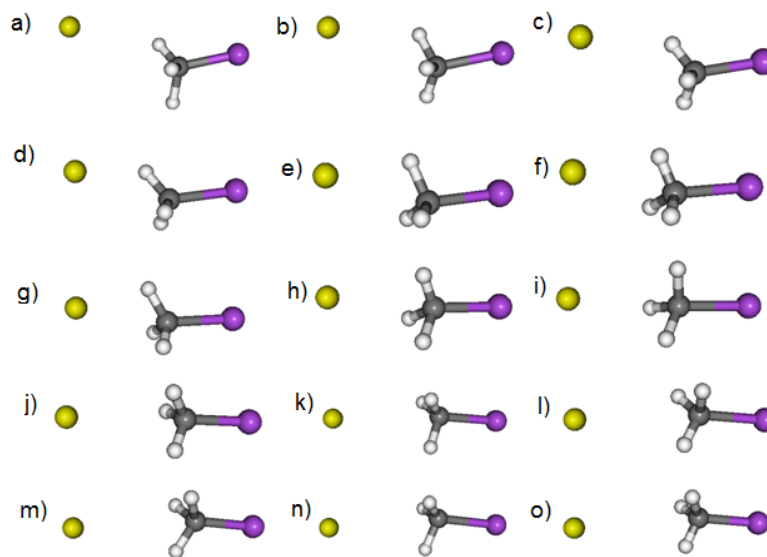


Figura 5.10 – Dinâmica direta da reação S_N2 : $CH_3Br + Cl$. O átomo no centro do tetraedro é o carbono, a esfera maior ligada ao carbono é o bromo e o nucleófilo, inicialmente distante é o cloro. As figuras estão em sequência com uma diferença de 25 fs entre cada trajetória.

Na figura 5.10 ocorre apenas um espalhamento onde 5.10.g é o ponto de máxima aproximação.

5.6 – Considerações gerais sobre os resultados

Os testes realizados levam-nos a acreditar que o programa está corretamente implementado. Os algoritmos foram consistentes com suas respectivas capacidades e o sistema apresentou boa conservação de energia. O teste envolvendo RMSD não é definitivo, mas apresentou acordo com o restante do trabalho.

Foi observado que o algoritmo de Beeman-Verlet, dentre os integradores testados, foi um dos mais eficientes para esse problema. Foi observado que é um algoritmo rápido, mas que não é adequado para sistemas de grande precisão. Os resultados mostraram que o Runge Kutta simétrico demonstrou eficiência comparável ao Beeman-Verlet e um desvio na posição

(RMSD) menor. Por isso, recomenda-se o Runge-Kutta simétrico para uma dinâmica direta baseada em gradiente. Os outros algoritmos foram inferiores na maioria dos quesitos avaliados, por isso não se recomenda para esse problema em particular.

A escolha do integrador é de grande importância em dinâmica direta. Este trabalho mostrou, por exemplo, que caso um conjunto de trajetórias integradas pelo Runge Kutta Gill demorem quatro meses para se completar. O mesmo resultado pode ser obtido pelo Runge Kutta simétrico em apenas um mês.

5.7 – Referências Bibliográficas

- [1] S. Tsai, M. Krech, D. P. Landau. Braz. J. Phys, **2004**, 34, 2A.
- [2] R. I. McLachlan, G. R. W. Quispel. J. Phys A: Math Gen. **2006**, 39, 5251-5285.
- [3] C. Schlier, A. Seiter. J. Phys. Chem. A. **1998**, 102, 9399-9404.
- [4] T.H. Dunning, Jr. J. Chem. Phys. **1989**, 90, 1007.
- [5] J.S. Binkley, J.A. Pople, W.J. Hehre. J. Am. Chem. Soc. **1980**, 102, 939; M.S. Gordon, J.S. Binkley, J.A. Pople, W.J. Pietro, W.J. Hehre. J. Am. Chem. Soc. **1983**, 104, 2797; K.D. Dobbs, W.J. Hehre. J. Comput. Chem. **1986**, 7, 359.
- [6] <https://cdssim.chem.ttu.edu/nav/htmlpages/licensemenu.jsp>. Acessado em 29/06/2014.
- [7] M. W. Schmidt, K. K. Baldrige, J. A. Boatz, S. T. Elbert, M. S. Gordon, J. H. Jensen, S. Koseki, N. Matsunaga, K. A. Nguyen, S. J. Su, T. L. Windus, M. Dupuis, J. A. Montgomery. J. Comp. Chem. **1993**, 14, 1347.
- [8] R. D. Levine. Molecular Reaction Dynamics. **2005**, Cambridge University Press, Cambridge.
- [9] A. M. Lesk. Introdução à Bioinformática. **2005**, Artmed, São Paulo.

Capítulo 6: Considerações finais

6.1 – Conclusões e perspectivas

O trabalho apresentado nessa dissertação foi dividido em duas etapas. A primeira foi o desenvolvimento de um software de dinâmica molecular *ab initio*, o DRKAI. A segunda foram testes de alguns algoritmos, com o intuito de manter o mais eficiente no software DRKAI.

Dentre os integradores testados, os que mais se destacaram foram o Runge Kutta simétrico e o Beeman (variação do Verlet). A referência [1] menciona que em uma comparação para o sistema 1,5 Dinitroviuret, Verlet foi superior ao Runge Kutta obtendo melhor conservação de energia com custo computacional menor. Neste trabalho, foi observado que as diferenças são muito pequenas para moléculas CH_3X . Também foi observado, para uma das trajetórias estudadas, que o RMSD foi melhor no Runge Kutta simétrico, significando que a trajetória com esse integrador se aproximou mais da correta, mesmo com uma conservação de energia inferior. Entretanto, há a necessidade de realizar mais testes numéricos, no geral, os dois algoritmos são muito eficientes no estudo desses sistemas, fato confirmado por trabalhos recentes com ambos [1-3].

O software DRKAI completo está presente no anexo B, foi desenvolvido em linguagem Prolog e acoplado ao software GAMESS. Os testes mostraram que ele aparenta estar corretamente implementado, mas não houve tempo para testar em uma dinâmica reativa completa, a fim de se comparar com resultados experimentais.

Esse trabalho gerou duas perspectivas distintas. A primeira é o desenvolvimento de um integrador baseado em Hessiana, existem algoritmos muito eficientes disponíveis na literatura [4-5]. A segunda perspectiva é aplicar o software a um sistema químico de interesse, inclusive, deseja-se incluir um solvente do tipo PCM (*Polarizable continuum model*), opção disponível na atual implementação.

6.2 – Referências Bibliográficas

- [1] W. L. Hase et. al. J. Phys. Chem. A. **2014**, 118, 2228-2236.
- [2] M. Pu, T. Privalov. J. Chem Phys. **2013**, 138, 154305.
- [3] J. Zhang, U. Lourderaj, R. Sun, J. Mikosch, R. Wester, W. L. Hase. J. Chem. Phys. **2013**, 138, 114309.
- [4] H. B. Schlegel. J. Chem. Theory Comput. **2013**, 9, 3293.
- [5] Hase et. al. J. Chem. Phys. **2010**, 133, 074101.

Anexo A: Outros trabalhos desenvolvidos durante o mestrado

Quando o mestrado foi iniciado, o título do projeto era: “Desenvolvimento de metodologias de otimização eficientes aplicadas ao estudo de cluster atômicos e/ou moleculares”. E tinha como objetivo desenvolver e/ou estudar novas técnicas de otimização para serem aplicadas no estudo de sistemas atômicos e moleculares. Não houve tempo de construir uma nova técnica, um novo algoritmo de integração ou um novo operador genético. Mas durante o mestrado, além do DRKAI, outro programa foi implementado. Um software que se aproxima do estado da arte na otimização de clusters, a saber, um algoritmo genético que tem como fitness a energia *ab initio*. A programação foi feita por mim e pelo Doutor Geison Voga Pereira.

Neste momento, está sendo aplicado a um cluster de sódio e potássio, e têm apresentado ótimos resultados, com perspectiva de publicação. Na figura 7.1 está apresentado um fluxograma do programa.

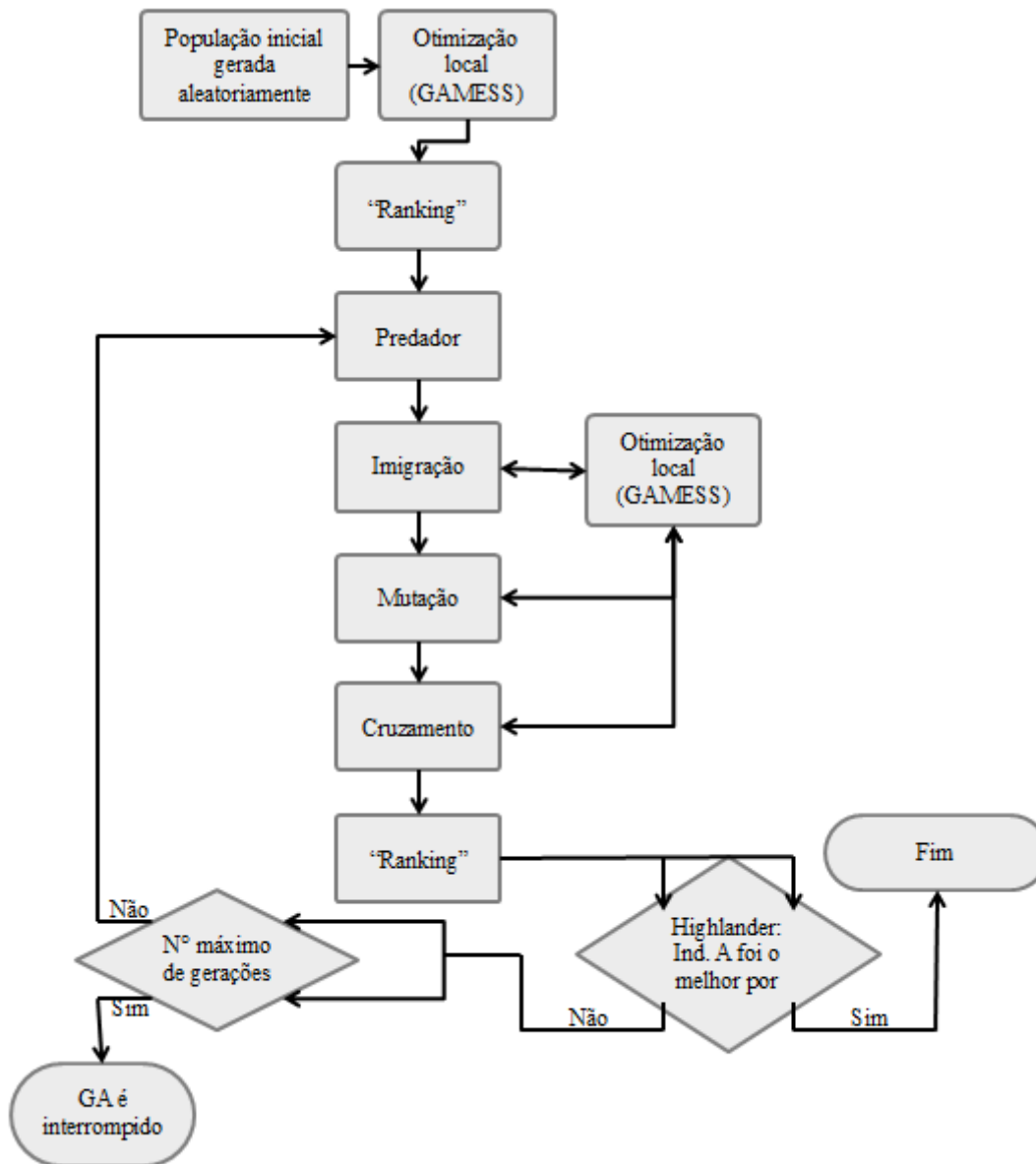


Figura A.1 – Fluxograma do algoritmo genético ab initio desenvolvido no período de mestrado.

Durante o período de mestrado também foi desenvolvido um trabalho em conjunto com o aluno de doutorado Rodrigo B. Kato. Nesse, foi feito um refinamento dos parâmetros do campo de força AMBER para moléculas que compõe o RNA. Tive uma participação ativa nos cálculos de estrutura eletrônica e na metodologia de solvatação. Foi escrito um

artigo desse trabalho e submetido na revista “*Journal of Chemical Theory and Computation*”. O artigo submetido está no fim dessa dissertação (Anexo C).

Anexo B: Código completo do DRKAI

O software foi dividido em quatro partes:

1. drakai.pl: Módulo principal, contém os integradores e gerencia os demais módulos.
2. forcagam.pl: Módulo que executa o GAMESS e coleta os resultados para realimentar o drakai.
3. cond_ini.pl: Módulo que gera as condições iniciais de uma molécula como descrito no capítulo 5.
4. alglin.pl: Módulo de álgebra linear. Contém diversos algoritmos para a manipulação de matrizes. Entre eles, um muito eficiente para a inversão de matrizes, chamado decomposição LU.

O algoritmo será mantido na versão mais atualizada no seguinte link:

<https://www.dropbox.com/sh/sh4bl30rjsl2pxo/AAAaTVEdRjGbkOXAhj3XSZlva>

Abaixo a versão atual será apresentada (03/09/2014)

-> drakai.pl

```
% Autor: Fred Vultor
% Data: 13/11/2013

% DRKAI - Um programa de dinâmica molecular ab initio

% como dados de entrada:
% -intervalo de tempo, e tempo do passo.
% -coordenadas e velocidades iniciais
% -potencial

:-use_module(forcagam),use_module(mag).

:-consult('siminput.drk').

:-discontiguous comeca_din/5.

:-dynamic(forca_ij/3).
:-dynamic(int_i/3).
:-dynamic(part/3).
:-dynamic(arqsai/1).
:-dynamic(forca_j/3).
:-dynamic(enlog/1).
:-dynamic(tipos/1).
```

```
:-dynamic(enerpot/1).
:-dynamic(for_rkg/2).
```

```
%%%%%%%%%% ROTINA QUE COMECA A DINAMICA %%%%%%%%%%%
```

```
%dinamica(+integrador 1-verlet 2-rkn, tmax,X,V,Tipos)
dinamica(Integrador,Tmax,X,V,Tipo):-
```

```
  inicia_info(X,Tipo,Len,Dlog,T1),
  comeca_din(Integrador,X,V,Len,Tmax),
  termina_info(Dlog,T1,Tmax).
```

```
inicia_info(X,Tipo,Len,Dlog,T1):-
  retractall(forca_ij(_,_)),
  retractall(forca_j(_,_)),
  retractall(int_i(_,_)),
  retractall(tipos(_)),
```

```
  forcagam:deleta_anterior,
```

```
  assert(tipos(Tipo)),
```

```
  get_time(T1),
  % criar_nome(Data),
  % string_concat(Data,'molekel.xyz',Nomemolekel),
  % string_concat(Data,'dinamica.log',Nomesaida),
  Nomeemolekel = 'nome.xyz',
  Nomesaida = 'nome.log',
  Nomeeener = 'nome.csv',
```

```
  open(Nomemolekel,write,Molout),
  open(Nomesaida,write,Dlog),
  assert(arqsai([Molout])),
```

```
% abertura espaco de fase
% string_concat(Data,'fase.csv',Nomeeener),
  open(Nomeeener,write,Enlog),
  assert(enlog(Enlog)),
% fim
```

```
  length(X,Len),
  grava_info_tipo(Tipo,1,Len).
```

```
termina_info(Dlog,T1,Tmax):-
  retractall(part(_,_)),
  retract(arqsai([Molout])),
  close(Molout),
  get_time(T2),
  T3 is T2-T1,
  write(Dlog,'o tempo de execucao foi: '),
  write(Dlog,T3),nl(Dlog),
  write(Dlog,'com input '),nl(Dlog),
  write(Dlog,'tempo total de simulacao e passo'),nl(Dlog),
  drk:param(TPass),
  write(Dlog,Tmax),write(Dlog,' '),write(Dlog,TPass),
  close(Dlog),
  retract(enlog(Enlog)),
  close(Enlog).
```

```
%%%%%%%%%% INTEGRADOR 1 - RUNGE-KUTTA-NYSTROM %%%%%%%%%%%
%%%%%%%%%%
```

```

%%%%%%%%%%
%%%%%%%%%%
%%%%%%%%%%
% Dinamica runge-kutta de quarta ordem
% S. Blanes, P.C. Moan; Practical Symplectic partitioned
% Runge-Kutta and Runge-Kutta Nystrom methods.
% Journal of Computational and Applied Mathematics 142 (2002) 313-330
comeca_din(1,X,V,Len,Tmax):-
  write_out(X,1,Len),

% variacao no tempo
drk:param(T1),
loop_tempo_rkn(T1,Tmax,[X,V],Len).

loop_tempo_rkn(Tcorr,Tmax,[Xo,Vo],Len):-
(
  Tmax<Tcorr,
  write('pronto'),nl,
  !
;
  drk:param(Tpass),

% Isso aqui e o k1, mas tem q multiplicar pela unidade
forbt(Xo,1),

  fase_esc(Xo,Vo,[Tcorr]),

  loop_atomos_k2(1,Len,Vo,Tpass,Xo,[],Xk2),

  forbt(Xk2,2),

  loop_atomos_rknx(1,Len,Vo,Tpass,Xo,[],Xsai),

  loop_atomos_rknv(1,Len,Tpass,Vo,[],Vsai),

  write_out(Xsai,1,Len),
  Tcorr2 is Tcorr+Tpass,
  loop_tempo_rkn(Tcorr2,Tmax,[Xsai,Vsai],Len)
).

loop_atomos_rknx(l,Len,V,Tpass,Xo,Xtemp,Xsai):-
(
  l>Len,
  reverse(Xtemp,Xsai),
  !
;
  nth1(l,Xo,[X1i,X2i,X3i]),
  nth1(l,V,[V1i,V2i,V3i]),
  forca_j(1,l,[F1,F2,F3]),
  forca_j(2,l,[F1m,F2m,F3m]),
  !,
  part(l,_,Mass),

  Alfa1 is 0.25e0,
  Alfa2 is 0.25e0,

  X1ic is X1i+Tpass*V1i + (Alfa1*(F1/Mass)+Alfa2*(F1m/Mass))*Tpass*Tpass,
  X2ic is X2i+Tpass*V2i + (Alfa1*(F2/Mass)+Alfa2*(F2m/Mass))*Tpass*Tpass,
  X3ic is X3i+Tpass*V3i + (Alfa1*(F3/Mass)+Alfa2*(F3m/Mass))*Tpass*Tpass,

  l2 is l+1,
  loop_atomos_rknx(l2,Len,V,Tpass,Xo,[[X1ic,X2ic,X3ic]]Xtemp,Xsai)
).

loop_atomos_rknv(l,Len,Tpass,Vo,Vtemp,Vsai):-
(
  l>Len,

```



```

loop_din(X,V,X,V,_,Tmax,Tcorr):-
  Tmax<Tcorr,
  write('pronto'),nl,
  assert(xfim(X)),
  assert(vfim(V)),!.
loop_din(Xcor,Vcor,Xfim,Vfim,Len,Tmax,Tcorr):-
  drk:param(Tpass),
  Tesc is truncate(Tcorr/Tpass),
  Tesc2 is Tesc mod 1,
  Tesc2 == 0,
  Tcorr2 is Tcorr+Tpass,

  fase_esc(Xcor,Vcor,[Tcorr]),

  loop_dinj(Xcor,Vcor,Xcor2,Vcor2,Len),!,
  loop_din(Xcor2,Vcor2,Xfim,Vfim,Len,Tmax,Tcorr2).

loop_din(Xcor,Vcor,Xfim,Vfim,Len,Tmax,Tcorr):-
  loop_dinj(Xcor,Vcor,Xcor2,Vcor2,Len),
  drk:param(Tpass),
  Tcorr2 is Tcorr+Tpass,
  loop_din(Xcor2,Vcor2,Xfim,Vfim,Len,Tmax,Tcorr2).

% LOOP QUE VARIA NO J, NAS PARTÍCULAS PRESENTES. PRA CADA
% PARTÍCULA, ELE CALCULA AS FORÇA QUE ATUAM SOBRE ELA, E RESOLVE
% AS EQUAÇÕES DE MOVIMENTO. TENHO UM ESQUEMA INTELIGENTE NAS FORÇAS
% QUE FAZ USO DA TERCEIRA LEI DE NEWTON.
%loop_dinj(Xtemp,Vtemp,Xtemp2,Vtemp2,1,Len,+integracao usada)
%o xtemp vai sendo modificado, mas é pra calcular as forças
%usando o x de entrada.
% os primeiros2 devem ser mantidos até o final pra ficar calculando
% os próximos é o espaço que eu tenho para acumular ao longo do loop
% os último são a saída, como o loop é tail recursivo, ele unifica
% no final.
% o indice final 1 significa o primeiro calculo, e o indice 2
% significa o uso da dinamica de Beeman.
loop_dinj(_,_,X,V,X,V,J,Len):-
  J>Len,
  !.
loop_dinj(Xtodos,Vtodos,Xtemp,Vtemp,Xsai,Vsai,J,Len):-
% calcula todas as forcas
% assert(forca_j(1,J,[F1,F2,F3])
% ai pode rodar o de baixo
  din_xv(Xtodos,Vtodos,Xj,Vj,J),
  adiciona(Xtemp,1,J,Xj,Xtemp1),
  adiciona(Vtemp,1,J,Vj,Vtemp1),
  J2 is J+1,
  loop_dinj(Xtodos,Vtodos,Xtemp1,Vtemp1,Xsai,Vsai,J2,Len),!.

% O loop_dinj/8 e o inicializador
% O loop_dinj/5 e o de beeman
% ESSA ROTINA CHAMA 2 LOPS: O QUE VARIA O X EM TODOS OS J's
% DEPOIS O QUE ATUALIZA A VELOCIDADE JÁ USANDO O NOVO X
% PRA CALCULAR A FORÇA.
% loop com rotina beeman
loop_dinj(Xtodos,Vtodos,Xsai,Vsai,Len):-
  loop_x(1,Len,Xtodos,Vtodos,Xtodos,Xsai),
  forbt(Xsai,3),

  loop_v(1,Len,Xsai,Vtodos,Vtodos,Vsai).

% loop_x(1,Len,Xtodos,Vtodos,Xtodos,Xj)
loop_x(J,Len,_,_,X,X):-
  J>Len,
  !.
loop_x(J,Len,Xtodos,Vtodos,Xtemp,Xfim):-
  dinb_x(Xtodos,Vtodos,Xj,J),
  adiciona(Xtemp,1,J,Xj,Xtemp1),
  J2 is J+1,

```

```

loop_x(J2,Len,Xtodos,Vtodos,Xtemp1,Xfim).

% loop_v(1,Len,Xtodos,Vtodos,Xtodos,Xj)
loop_v(J,Len,_,_,V,V):-
  J>Len,
  !.
loop_v(J,Len,Xtodos,Vtodos,Vtemp,Vfim):-
  dinb_v(Vtodos,Vj,J),
  adiciona(Vtemp,1,J,Vj,Vtemp1),
  J2 is J+1,
  loop_v(J2,Len,Xtodos,Vtodos,Vtemp1,Vfim).

% ROTINA QUE CALCULA A NOVA POSIÇÃO E VELOCIDADE DA PARTÍCULA
% dinb_xv(Xtodos,Vtodos listas de entrada,Xj,Vj elementos mod
%   ,J corrente,Len),
dinb_x(Xtodos,Vtodos,Xj,J):-
  nth1(J,Xtodos,Xcor),
  nth1(J,Vtodos,Vcor),
  nth1(1,Xcor,X1),
  nth1(2,Xcor,X2),
  nth1(3,Xcor,X3),
  nth1(1,Vcor,V1),
  nth1(2,Vcor,V2),
  nth1(3,Vcor,V3),

  part(J,_,Mass),
% Cm is 1.6605402e-27,
% Mass2 is Mass*Cm,

  drk:param(Tpass),
  forca_j(1,J,[F1ant,F2ant,F3ant]),
  forca_j(2,J,[F1,F2,F3]),

  Xa is X1+V1*Tpass+((4.0e0*F1-F1ant)*Tpass*Tpass)/(6.0e0*Mass),
  Xb is X2+V2*Tpass+((4.0e0*F2-F2ant)*Tpass*Tpass)/(6.0e0*Mass),
  Xc is X3+V3*Tpass+((4.0e0*F3-F3ant)*Tpass*Tpass)/(6.0e0*Mass),

  Xj=[Xa,Xb,Xc],!
.

% ROTINA QUE CALCULA A NOVA POSIÇÃO E VELOCIDADE DA PARTÍCULA
% dinb_xv(Xtodos,Vtodos listas de entrada,Xj,Vj elementos mod
%   ,J corrente,Len),
dinb_v(Vtodos,Vj,J):-
  nth1(J,Vtodos,Vcor),
  nth1(1,Vcor,V1),
  nth1(2,Vcor,V2),
  nth1(3,Vcor,V3),

% forb(Xtodos,_[0.0e0,0.0e0,0.0e0],[F1pos,F2pos,F3pos],1,J,Len),
% todas as forcas ja foram calculadas
forca_j(3,J,[F1pos,F2pos,F3pos]),
part(J,_,Mass),

  drk:param(Tpass),
  forca_j(1,J,[F1ant,F2ant,F3ant]),
  forca_j(2,J,[F1m,F2m,F3m]),

%% VELOCIDADE CORRIGIDA DE BEEMAN

Va is V1+(2.0e0*F1pos +5.0e0*F1m-F1ant)*(Tpass/(6.0e0*Mass)),
Vb is V2+(2.0e0*F2pos +5.0e0*F2m-F2ant)*(Tpass/(6.0e0*Mass)),
Vc is V3+(2.0e0*F3pos +5.0e0*F3m-F3ant)*(Tpass/(6.0e0*Mass)),

Vj=[Va,Vb,Vc],

retractall(forca_j(1,J,_)),
retractall(forca_j(2,J,_)),

```



```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
comeca_din(3,X,V,Len,Tmax):-
    write_out(X,1,Len),
    % variacao no tempo
    drk:param(T1),
    % forca inicial
    forbt(X,1),
    % Iniciando as constantes A's
    inicia_a(1,Len,[],Aini,[],Alfas),
    % inicio da simulacao
    loop_tempo_grd(T1,Tmax,[Aini,Alfas],[X,V],Len).

% No inicio da simulacao todos os A's sao zero,
% Depois que eles vao prevendo e corrigindo.'
inicia_a(I,Len,Atemp,Aini,Alfatemp,Alfaini):-
    (
        I>Len,
        Atemp=Aini,
        Alfatemp=Alfaini,
        !
    ;
        I2 is I+1,
        Z = 0,
        inicia_a(I2,Len,[ [0,0,0],[0,0,0],[0,0,0]]|Atemp ],Aini,[ [[Z,Z,Z],[Z,Z,Z],[Z,Z,Z]]|Alfatemp ],Alfaini)
    ).

% LOOP NO TEMPO
loop_tempo_grd(Tcorr,Tmax,[Aini,Alfas],[Xo,Vo],Len):-
    (
        Tmax<Tcorr,
        write('pronto'),nl,
        !
    ;
        drk:param(Tpass),
        (
            Tcorr=Tpass,
            grd_seq([1,6],[Aini,Alfas],[Xo,Vo],[Afim,Alfafim],_,_[Xf,Vf]),
            !
        ;
            grd_seq([1,2],[Aini,Alfas],[Xo,Vo],[Afim,Alfafim],_,_[Xf,Vf])
        ),
        fase_esc(Xf,Vf,[Tcorr]),

        Tcorr2 is Tcorr+Tpass,
        loop_tempo_grd(Tcorr2,Tmax,[Afim,Alfafim],[Xf,Vf],Len)
    ).

grd_seq([I,Nmax],A_,A,[X,V],[X,V]):-
    I>Nmax,
    forbt(X,1),
    !.
grd_seq([I,Nmax],[Aini,Alfas],[Xo,Vo],[Afim,Alfafim],_,XVf):-
    drk:param(Tpass),
    T2 is Tpass*0.2123e0,
    T3 is Tpass*0.5905e0,
    T4 is Tpass*0.9114e0,
    length(Xo,Len),

% Velocidades sao uteis apenas no final
% acho que vou precisar de 4 operacoes, tem que calcular de novo
% no final. Nesse momento t4=T

% Iteracao 1 - calculo do previsor x2
loop_atomos_grdx(1,Len,1,[Xo,Vo],Aini,T2,[],X2),
forbt(X2,2),
loop_Ais_grd(1,Len,alfa1,[Aini,Alfas],[T2,T3],[[],[]],[A2,Alfa2]),

```



```

% Iteracao 2 - calculo do previsor x3
loop_atomos_grdx(1,Len,1,[Xo,Vo],A2,T3,[],X3),
forbt(X3,3),
loop_Ais_grd(1,Len,alfa2,[A2,Alfa2],[T2,T3],[[],[]],[A3,_]),

% Iteracao 3 - calculo do previsor x4
loop_atomos_grdx(1,Len,1,[Xo,Vo],A3,T4,[],X4),
forbt(X4,4),
loop_Ais_grd(1,Len,alfa3,[T2,T3,T4],[[],[]],[A4,Alfa4]),

% Por ultimo calculo das novas posicoes e velocidades.
loop_atomos_grdv(1,Len,1,Vo,A4,Tpass,[],V4),
loop_atomos_grdx(1,Len,1,[Xo,Vo],A4,Tpass,[],X5),

I2 is I+1,
grd_seq([I2,Nmax],[A4,Alfa4],[Xo,Vo],[Afim,Alfafim],[X5,V4],XVf).

loop_Ais_grd(I,Len,alfa1,[Aini,Alfas],[T2,T3],[Atemp,Alfatemp],[A2,Alfa2]):-
(
I>Len,
reverse(Atemp,A2),
reverse(Alfatemp,Alfa2),
!
;
nth1(I,Aini,[_ ,Ax2,Ax3],[_ ,Ay2,Ay3],[_ ,Az2,Az3] ),
nth1(I,Aini,[_ ,Alfax2,Alfax3],[_ ,Alfay2,Alfay3],[_ ,Alfaz2,Alfaz3] ),
forca_j(1,I,[Fx,Fy,Fz]),
forca_j(2,I,[Fx2,Fy2,Fz2]),!

part(I,_ ,Mass),
Alfax1 is (Fx2-Fx)/(T2*Mass),
Alfay1 is (Fy2-Fy)/(T2*Mass),
Alfaz1 is (Fz2-Fz)/(T2*Mass),

Ax1n is Alfax1 - Alfax2*T2 + T2*T3*Alfax3,
Ay1n is Alfay1 - Alfay2*T2 + T2*T3*Alfay3,
Az1n is Alfaz1 - Alfaz2*T2 + T2*T3*Alfaz3,

Acont = [[Ax1n,Ax2,Ax3],[Ay1n,Ay2,Ay3],[Az1n,Az2,Az3] | Atemp],
Alfacont = [[Alfax1,Alfax2,Alfax3],[Alfay1,Alfay2,Alfay3],[Alfaz1,Alfaz2,Alfaz3] | Alfatemp],

I2 is I+1,
loop_Ais_grd(I2,Len,alfa1,[Aini,Alfas],[T2,T3],[Acont,Alfacont],[A2,Alfa2])
).

loop_Ais_grd(I,Len,alfa2,[Aini,Alfas],[T2,T3],[Atemp,Alfatemp],[Af,Alfaf]):-
(
I>Len,
reverse(Atemp,Af),
reverse(Alfatemp,Alfaf),
!
;
nth1(I,Aini,[_ ,_ ,Ax3],[_ ,_ ,Ay3],[_ ,_ ,Az3] ),
nth1(I,Aini,[_ ,_ ,Alfax3],[_ ,_ ,Alfay3],[_ ,_ ,Alfaz3] ),
forca_j(1,I,[Fx,Fy,Fz]),
forca_j(2,I,[Fx2,Fy2,Fz2]),
forca_j(3,I,[Fx3,Fy3,Fz3]),!

part(I,_ ,Mass),
Alfax1 is (Fx2-Fx)/(T2*Mass),
Alfay1 is (Fy2-Fy)/(T2*Mass),
Alfaz1 is (Fz2-Fz)/(T2*Mass),

Alfax2 is (((Fx3-Fx)/(T3*Mass)) - Alfax1)/(T3-T2),
Alfay2 is (((Fy3-Fy)/(T3*Mass)) - Alfay1)/(T3-T2),
Alfaz2 is (((Fz3-Fz)/(T3*Mass)) - Alfaz1)/(T3-T2),

Ax2n is Alfax2 + (-T2-T3)*Alfax3,

```

```

Ay2n is Alfay2 + (-T2-T3)*Alfay3,
Az2n is Alfaz2 + (-T2-T3)*Alfaz3,

Ax1n is Alfax1 - Alfax2*T2 + T2*T3*Alfax3,
Ay1n is Alfay1 - Alfay2*T2 + T2*T3*Alfay3,
Az1n is Alfaz1 - Alfaz2*T2 + T2*T3*Alfaz3,

Acont = [[[Ax1n,Ax2n,Ax3],[Ay1n,Ay2n,Ay3],[Az1n,Az2n,Az3]] | Atemp],
Alfacont = [ [[Alfax1,Alfax2,Alfax3],[Alfay1,Alfay2,Alfay3],[Alfaz1,Alfaz2,Alfaz3]] | Alfatemp],

I2 is I+1,
loop_Ais_grd(I2,Len,alfa2,[Aini,Alfas],[T2,T3],[Acont,Alfacont],[Af,Alfaf])
).

```

```

loop_Ais_grd(I,Len,alfa3,[T2,T3,T4],[Atemp,Alfatemp],[Af,Alfaf):-

```

```

(
  I>Len,
  reverse(Atemp,Af),
  reverse(Alfatemp,Alfaf),
  !
;
forca_j(1,I,[Fx,Fy,Fz]),
forca_j(2,I,[Fx2,Fy2,Fz2]),
forca_j(3,I,[Fx3,Fy3,Fz3]),
forca_j(4,I,[Fx4,Fy4,Fz4]),!,

part(I,_,Mass),
Alfax1 is (Fx2-Fx)/(T2*Mass),
Alfay1 is (Fy2-Fy)/(T2*Mass),
Alfaz1 is (Fz2-Fz)/(T2*Mass),

Alfax2 is (((Fx3-Fx)/(T3*Mass)) - Alfax1)/(T3-T2),
Alfay2 is (((Fy3-Fy)/(T3*Mass)) - Alfay1)/(T3-T2),
Alfaz2 is (((Fz3-Fz)/(T3*Mass)) - Alfaz1)/(T3-T2),

Alfax3 is (((Fx4-Fx)/(T4*Mass)) - Alfax1)/(T4-T2) - Alfax2/(T4-T3),
Alfay3 is (((Fy4-Fy)/(T4*Mass)) - Alfay1)/(T4-T2) - Alfay2/(T4-T3),
Alfaz3 is (((Fz4-Fz)/(T4*Mass)) - Alfaz1)/(T4-T2) - Alfaz2/(T4-T3),

```

```

%%%%%%%%%%
%%%%%%%%%%
% append('2014.txt'),
% write('alfax1: '),write(Alfax1),tab(2),
% write('alfay1: '),write(Alfay1),tab(2),nl,
% write('alfax2: '),write(Alfax2),tab(2),
% write('alfay2: '),write(Alfay2),tab(2),nl,
% write('alfax3: '),write(Alfax3),tab(2),
% write('alfay3: '),write(Alfay3),tab(2),
% nl,
% nl,
% told,
%%%%%%%%%%
%%%%%%%%%%

```

```

Ax3n is Alfax3,
Ay3n is Alfay3,
Az3n is Alfaz3,

Ax2n is Alfax2 + (-T2-T3)*Alfax3,
Ay2n is Alfay2 + (-T2-T3)*Alfay3,
Az2n is Alfaz2 + (-T2-T3)*Alfaz3,

Ax1n is Alfax1 - Alfax2*T2 + T2*T3*Alfax3,
Ay1n is Alfay1 - Alfay2*T2 + T2*T3*Alfay3,
Az1n is Alfaz1 - Alfaz2*T2 + T2*T3*Alfaz3,

Acont = [[Ax1n,Ax2n,Ax3n],[Ay1n,Ay2n,Ay3n],[Az1n,Az2n,Az3n]] | Atemp],
Alfacont = [ [[Alfax1,Alfax2,Alfax3],[Alfay1,Alfay2,Alfay3],[Alfaz1,Alfaz2,Alfaz3]] | Alfatemp],

```



```

loop_tempo_srk(T1,Tmax,[X,V],Len).

loop_tempo_srk(Tcorr,Tmax,[Xo,Vo],Len):-
(
  Tmax<Tcorr,
  write('pronto'),nl,
  !
;
  drk:param(Tpass),
% Iteracao - 0
  A0 is 0.09517625454177405268e0,
  loop_atomos_srkx(1,Len,Vo,Tpass,A0,Xo,[],X1),

  forbt(X1,1),
  A1 is 0.66629689399770780134e0,
  loop_atomos_srkx(1,Len,Vo,Tpass,A1,[],V1),

% Iteracao - 2
  A2 is -0.12795028552368677941e0,
  loop_atomos_srkx(1,Len,V1,Tpass,A2,X1,[],X2),

  forbt(X2,1),
  A3 is 0.02461890095210508713e0,
  loop_atomos_srkx(1,Len,V1,Tpass,A3,[],V2),

% Iteracao - 4
  A4 is 0.10597295345325113144e0,
  loop_atomos_srkx(1,Len,V2,Tpass,A4,X2,[],X3),

  forbt(X3,1),
  A5 is -0.41072553361795113232e0,
  loop_atomos_srkx(1,Len,V2,Tpass,A5,[],V3),

% Iteracao - 6
  A6 is 0.44822227660082748417e0,
  loop_atomos_srkx(1,Len,V3,Tpass,A6,X3,[],X4),

  forbt(X4,1),
  A7 is 0.65772926205091317769e0,
  loop_atomos_srkx(1,Len,V3,Tpass,A7,[],V4),

% Iteracao - 8
  A8 is -0.02142119907216588887e0,
  loop_atomos_srkx(1,Len,V4,Tpass,A8,X4,[],X5),

  forbt(X5,1),
  A9 is 2*(-0.43791952338277493384e0),
  loop_atomos_srkx(1,Len,V4,Tpass,A9,[],V5),

% Iteracao - 10
  A10 is A8,
  loop_atomos_srkx(1,Len,V5,Tpass,A10,X5,[],X6),

  forbt(X6,1),
  A11 is A7,
  loop_atomos_srkx(1,Len,V5,Tpass,A11,[],V6),

% Iteracao - 12
  A12 is A6,
  loop_atomos_srkx(1,Len,V6,Tpass,A12,X6,[],X7),

  forbt(X7,1),
  A13 is A5,
  loop_atomos_srkx(1,Len,V6,Tpass,A13,[],V7),

% Iteracao - 14
  A14 is A4,
  loop_atomos_srkx(1,Len,V7,Tpass,A14,X7,[],X8),

  forbt(X8,1),

```

```

A15 is A3,
loop_atomos_srkv(1,Len,V7,Tpass,A15,[],V8),

% Iteracao - 16
A16 is A2,
loop_atomos_srkv(1,Len,V8,Tpass,A16,X8,[],X9),

forbt(X9,1),
A17 is A1,
loop_atomos_srkv(1,Len,V8,Tpass,A17,[],V9),

% x final
loop_atomos_srkv(1,Len,V9,Tpass,A0,X9,[],X10),
forbt(X10,1),
fase_esc(X10,V9,[Tcorr]),

Tcorr2 is Tcorr+Tpass,
loop_tempo_srk(Tcorr2,Tmax,[X10,V9],Len)
).

loop_tempo_srk4(Tcorr,Tmax,[Xo,Vo],Len):-
(
Tmax<Tcorr,
write('pronto'),nl,
!
;
drk:param(Tpass),
% Iteracao - 0
forbt(Xo,1),
A0 is 0.5e0,
A1 is -(1/48),

loop_atomos_srkv(1,Len,Vo,Tpass,A0,[],V1),
loop_atomos_srkv(1,Len,V1,Tpass,A1,Xo,[],X1),

% Iteracao - 2
A2 is 1/3,
A3 is 3/8,
forbt(X1,1),

loop_atomos_srkv(1,Len,V1,Tpass,A2,[],V2),
loop_atomos_srkv(1,Len,V2,Tpass,A3,X1,[],X2),

% Iteracao - 4
A4 is -(1/3),
A5 is 7/24,
forbt(X2,1),

loop_atomos_srkv(1,Len,V2,Tpass,A4,[],V3),
loop_atomos_srkv(1,Len,V3,Tpass,A5,X2,[],X3),

% Iteracao - 6
A6 is -(1/3),
A7 is 3/8,
forbt(X3,1),

loop_atomos_srkv(1,Len,V3,Tpass,A6,[],V4),
loop_atomos_srkv(1,Len,V4,Tpass,A7,X3,[],X4),

% Iteracao - 8
A8 is 1/3,
A9 is -(1/48),

forbt(X4,1),
loop_atomos_srkv(1,Len,V4,Tpass,A8,[],V5),
loop_atomos_srkv(1,Len,V5,Tpass,A9,X4,[],X5),

% v final
forbt(X5,1),
loop_atomos_srkv(1,Len,V5,Tpass,A0,[],V6),

```



```

% yn = [ xn vn ]
% fyn = [ vn f(xn) ]
loop_tempo_rk4(Tcorr,Tmax,[Xo,Vo],Len):-
(
Tmax<Tcorr,
write('pronto'),nl,
!
;
drk:param(Tpass),

% k1 = h*f(to,[xo vo]) = h*[vo f(xo)]
forbt(Xo,1),

fase_esc(Xo,Vo,[Tcorr]),

% k2 = h*f(to+h/2,[xo+hvo/2 vo+hf(xo)/2]) = f([xo+hvo/2 vo+h...])
% repara no anterior tem uma troca
% f([xo+hvo/2 vo+hf(xo)/2]) = [vo+hf(xo/2) f(xo+hvo/2)]

loop_atomos_kk(1,Len,0.5e0,[Vo,Xo,Vo],Tpass,[],[],[K1x,K1v],[],[],[Xk2,Vk2]),
forbt(Xk2,1),

loop_atomos_kk(1,Len,0.5e0,[Vk2,Xo,Vo],Tpass,[],[],[K2x,K2v],[],[],[Xk3,Vk3]),
forbt(Xk3,1),

loop_atomos_kk(1,Len,1.0e0,[Vk3,Xo,Vo],Tpass,[],[],[K3x,K3v],[],[],[Xk4,Vk4]),
forbt(Xk4,1),

loop_atomos_kk(1,Len,f,[Vk4],Tpass,[],[],[K4x,K4v]),

append(Xo,Xoc),
append(Vo,Voc),

Cons1 is 1/6,
Cons2 is 1/3,
append(K1x,K1xc),
append(K1v,K1vc),
alglin:soma_vec(K1xc,Cons1,K1xf),
alglin:soma_vec(K1vc,Cons1,K1vf),

append(K2x,K2xc),
append(K2v,K2vc),
alglin:soma_vec(K2xc,Cons2,K2xf),
alglin:soma_vec(K2vc,Cons2,K2vf),

append(K3x,K3xc),
append(K3v,K3vc),
alglin:soma_vec(K3xc,Cons2,K3xf),
alglin:soma_vec(K3vc,Cons2,K3vf),

append(K4x,K4xc),
append(K4v,K4vc),
alglin:soma_vec(K4xc,Cons1,K4xf),
alglin:soma_vec(K4vc,Cons1,K4vf),

alglin:soma_vec(Xoc,K1xf,Xf1),
alglin:soma_vec(Xf1,K2xf,Xf2),
alglin:soma_vec(Xf2,K3xf,Xf3),
alglin:soma_vec(Xf3,K4xf,Xff),

alglin:soma_vec(Voc,K1vf,Vf1),
alglin:soma_vec(Vf1,K2vf,Vf2),
alglin:soma_vec(Vf2,K3vf,Vf3),
alglin:soma_vec(Vf3,K4vf,Vff),

mag:appendinv(Xff,Xsai),
mag:appendinv(Vff,Vsai),

Tcorr2 is Tcorr+Tpass,
loop_tempo_rk4(Tcorr2,Tmax,[Xsai,Vsai],Len)

```

```

).
loop_atomos_kk(l,Len,f,[V],Tpass,[Xt,Vt],[Xs,Vs):-
(
  l>Len,
  reverse(Xt,Xs),
  reverse(Vt,Vs),
  !
;
  nth1(l,V,[V1i,V2i,V3i]),
  forca_j(1,l,[F1,F2,F3]),
  !,
  part(l,_,Mass),

  V1ik is Tpass*V1i,
  V2ik is Tpass*V2i,
  V3ik is Tpass*V3i,

  X1ik is Tpass*(F1/Mass),
  X2ik is Tpass*(F2/Mass),
  X3ik is Tpass*(F3/Mass),

  l2 is l+1,
% atencao, linhas adicionais por causa de cabeçalho grande
  loop_atomos_kk(l2,Len,f,[V],Tpass,
    [[[V1ik,V2ik,V3ik]|Xt],[[X1ik,X2ik,X3ik]|Vt]], [Xs,Vs])
).

loop_atomos_kk(l,Len,Ky,[V,Xo,Vo],Tpass,[Xt,Vt],[Xs,Vs],[Yxt,Yvt],[Yx,Yv):-
(
  l>Len,
  reverse(Xt,Xs),
  reverse(Vt,Vs),
  reverse(Yxt,Yx),
  reverse(Yvt,Yv),
  !
;
  nth1(l,V,[V1i,V2i,V3i]),
  nth1(l,Xo,[X1o,X2o,X3o]),
  nth1(l,Vo,[V1o,V2o,V3o]),
  forca_j(1,l,[F1,F2,F3]),
  !,
  part(l,_,Mass),

  V1ik is Tpass*V1i,
  V2ik is Tpass*V2i,
  V3ik is Tpass*V3i,

  X1ik is Tpass*(F1/Mass),
  X2ik is Tpass*(F2/Mass),
  X3ik is Tpass*(F3/Mass),

% Calculando o y de k2
  Y1k2x is X1o+Ky*V1ik,
  Y2k2x is X2o+Ky*V2ik,
  Y3k2x is X3o+Ky*V3ik,

  Y1k2v is V1o+Ky*X1ik,
  Y2k2v is V2o+Ky*X2ik,
  Y3k2v is V3o+Ky*X3ik,

  l2 is l+1,
% atencao, linhas adicionais por causa de cabeçalho grande
  loop_atomos_kk(l2,Len,Ky,[V,Xo,Vo],Tpass,
    [[[V1ik,V2ik,V3ik]|Xt],[[X1ik,X2ik,X3ik]|Vt]], [Xs,Vs],
    [[[Y1k2x,Y2k2x,Y3k2x]|Yxt],[[Y1k2v,Y2k2v,Y3k2v]|Yvt]], [Yx,Yv])
).

```



```

alglin:cons_vec(Kx2,Cons2,Qx2t),
alglin:cons_vec(Kv2,Cons2,Qv2t),
alglin:cons_vec(Qx1,Cons3,Qx2tt),
alglin:cons_vec(Qv1,Cons3,Qv2tt),
alglin:soma_vec(Qx2t,Qx2tt,Qx2),
alglin:soma_vec(Qv2t,Qv2tt,Qv2),

mag:appendinv(X2,X2f),
forbt(X2f,1),
loop_atomos_foper(Len,[V2],[Fx3,Fv3]),
alglin:cons_vec(Fx3,Tpass,Kx3),
alglin:cons_vec(Fv3,Tpass,Kv3),

Cons4 is (1.0e0+0.5e0*sqrt(2.0e0)),
alglin:subtrai_vec(Kx3,Qx2,X3t),
alglin:subtrai_vec(Kv3,Qv2,V3t),
alglin:cons_vec(X3t,Cons4,X3tt),
alglin:cons_vec(V3t,Cons4,V3tt),
alglin:soma_vec(X2,X3tt,X3),
alglin:soma_vec(V2,V3tt,V3),

Cons5 is (2.0e0+sqrt(2.0e0)),
Cons6 is -(2.0e0+1.5e0*sqrt(2.0e0)),
alglin:cons_vec(Kx3,Cons5,Qx3t),
alglin:cons_vec(Kv3,Cons5,Qv3t),
alglin:cons_vec(Qx2,Cons6,Qx3tt),
alglin:cons_vec(Qv2,Cons6,Qv3tt),
alglin:soma_vec(Qx3t,Qx3tt,Qx3),
alglin:soma_vec(Qv3t,Qv3tt,Qv3),

mag:appendinv(X3,X3f),
forbt(X3f,1),
loop_atomos_foper(Len,[V3],[Fx4,Fv4]),
alglin:cons_vec(Fx4,Tpass,Kx4),
alglin:cons_vec(Fv4,Tpass,Kv4),

Cons7 is 1/6,
Cons8 is -1/3,
alglin:cons_vec(Kx4,Cons7,X4t),
alglin:cons_vec(Kv4,Cons7,V4t),
alglin:cons_vec(Qx3,Cons8,X4tt),
alglin:cons_vec(Qv3,Cons8,V4tt),
alglin:soma_vec(X4t,X4tt,X4ttt),
alglin:soma_vec(V4t,V4tt,V4ttt),
alglin:soma_vec(X4ttt,X3,X4),
alglin:soma_vec(V4ttt,V3,V4),

Tcorr2 is Tcorr+Tpass,
loop_tempo_rkg(Tcorr2,Tmax,[X4,V4],Len)
).

loop_atomos_foper(Len,[V],[Xs,Vs):-
mag:appendinv(V,Vini),
loop_atomos_foper(1,Len,[Vini],[[]],[Xst,Vst]),
append(Xst,Xs),
append(Vst,Vs).

loop_atomos_foper(l,Len,[V],[Xt,Vt],[Xs,Vs):-
(
l>Len,
reverse(Xt,Xs),
reverse(Vt,Vs),
!
;
nth1(l,V,[V1i,V2i,V3i]),
forca_j(1,l,[F1,F2,F3]),
!,
part(l,_,Mass),

```



```
Tcorr2 is Tcorr+Tpass,
loop_tempo_admou(Tcorr2,Tmax,[Xf,Vf],Len,Prim2)
).
```

```
adams_corrector([Xo,Vo],Prim,Tpass,[Xf,Vf]):-
```

```
for_rkg(Prim,[Fx1,Fv1]),
P2 is Prim+1,
for_rkg(P2,[Fx2,Fv2]),
P3 is P2+1,
for_rkg(P3,[Fx3,Fv3]),
P4 is P3+1,
for_rkg(P4,[Fx4,Fv4]),
P5 is P4+1,
for_rkg(P5,[Fx5,Fv5]),
P6 is P5+1,
for_rkg(P6,[Fx6,Fv6]),!,
```

```
% Lembrar Fx6 e o mais recente
alglin:cons_vec(Fx1,27.0e0,Fx1t),
alglin:cons_vec(Fv1,27.0e0,Fv1t),
Cons1 is -173.0e0,
alglin:cons_vec(Fx2,Cons1,Fx2t),
alglin:cons_vec(Fv2,Cons1,Fv2t),
alglin:cons_vec(Fx3,482.0e0,Fx3t),
alglin:cons_vec(Fv3,482.0e0,Fv3t),
Cons2 is -798.0e0,
alglin:cons_vec(Fx4,Cons2,Fx4t),
alglin:cons_vec(Fv4,Cons2,Fv4t),
alglin:cons_vec(Fx5,1427.0e0,Fx5t),
alglin:cons_vec(Fv5,1427.0e0,Fv5t),
alglin:cons_vec(Fx6,475.0e0,Fx6t),
alglin:cons_vec(Fv6,475.0e0,Fv6t),
```

```
alglin:soma_vec(Fx1t,Fx2t,Fxtt),
alglin:soma_vec(Fxtt,Fx3t,Fxttt),
alglin:soma_vec(Fxttt,Fx4t,Fxtttt),
alglin:soma_vec(Fxtttt,Fx5t,Fxttttt),
alglin:soma_vec(Fxttttt,Fx6t,Fxtttttt),
alglin:soma_vec(Fv1t,Fv2t,Fvtt),
alglin:soma_vec(Fvtt,Fv3t,Fvttt),
alglin:soma_vec(Fvttt,Fv4t,Fvtttt),
alglin:soma_vec(Fvtttt,Fv5t,Fvttttt),
alglin:soma_vec(Fvttttt,Fv6t,Fvtttttt),
```

```
Const is Tpass/1440.0e0,
```

```
alglin:cons_vec(Fxttttt,Const,Xcorr),
alglin:cons_vec(Fvttttt,Const,Vcorr),
```

```
alglin:soma_vec(Xo,Xcorr,Xf),
alglin:soma_vec(Vo,Vcorr,Vf).
```

```
adams_predictor([Xo,Vo],Prim,Tpass,[Xpred,Vpred]):-
```

```
for_rkg(Prim,[Fx1,Fv1]),
P2 is Prim+1,
for_rkg(P2,[Fx2,Fv2]),
P3 is P2+1,
for_rkg(P3,[Fx3,Fv3]),
P4 is P3+1,
for_rkg(P4,[Fx4,Fv4]),
P5 is P4+1,
for_rkg(P5,[Fx5,Fv5]),!,
```

```
% Lembrar Fx5 e o mais recente
alglin:cons_vec(Fx1,251.0e0,Fx1t),
alglin:cons_vec(Fv1,251.0e0,Fv1t),
Cons1 is -1274.0e0,
alglin:cons_vec(Fx2,Cons1,Fx2t),
```

```

alglin:cons_vec(Fv2,Cons1,Fv2t),
alglin:cons_vec(Fx3,2616.0e0,Fx3t),
alglin:cons_vec(Fv3,2616.0e0,Fv3t),
Cons2 is -2774.0e0,
alglin:cons_vec(Fx4,Cons2,Fx4t),
alglin:cons_vec(Fv4,Cons2,Fv4t),
alglin:cons_vec(Fx5,1901.0e0,Fx5t),
alglin:cons_vec(Fv5,1901.0e0,Fv5t),

alglin:soma_vec(Fx1t,Fx2t,Fxtt),
alglin:soma_vec(Fxtt,Fx3t,Fxttt),
alglin:soma_vec(Fxttt,Fx4t,Fxtttt),
alglin:soma_vec(Fxtttt,Fx5t,Fxttttt),
alglin:soma_vec(Fv1t,Fv2t,Fvtt),
alglin:soma_vec(Fvtt,Fv3t,Fvttt),
alglin:soma_vec(Fvttt,Fv4t,Fvtttt),
alglin:soma_vec(Fvtttt,Fv5t,Fvttttt),

Const is Tpass/720.0e0,

alglin:cons_vec(Fxttttt,Const,Xpredt),
alglin:cons_vec(Fvttttt,Const,Vpredt),

alglin:soma_vec(Xo,Xpredt,Xpred),
alglin:soma_vec(Vo,Vpredt,Vpred).

% yn = [ xn vn ]
% fyn = [ vn f(xn) ]
loop_tempo_rkgin(l,lmax,[Xo,Vo],[Xf,Vf],Len):-
(
  l>lmax,
  Xo=Xf,
  Vo=Vf,
  !
;
  drk:param(Tpass),

  mag:appendinv(Xo,Xof),
  forbt(Xof,1),
  loop_atomos_foper(Len,[Vo],[Fx1,Fv1]),
  alglin:cons_vec(Fx1,Tpass,Kx1),
  alglin:cons_vec(Fv1,Tpass,Kv1),

  alglin:cons_vec(Kx1,0.5e0,X1t),
  alglin:cons_vec(Kv1,0.5e0,V1t),
  alglin:soma_vec(Xo,X1t,X1),
  alglin:soma_vec(Vo,V1t,V1),

  Qx1 = Kx1,
  Qv1 = Kv1,

  mag:appendinv(X1,X1f),
  forbt(X1f,1),
  loop_atomos_foper(Len,[V1],[Fx2,Fv2]),
  alglin:cons_vec(Fx2,Tpass,Kx2),
  alglin:cons_vec(Fv2,Tpass,Kv2),

  Cons1 is (1.0e0-0.5e0*sqrt(2.0e0)),
  alglin:subtrai_vec(Kx2,Qx1,X2t),
  alglin:subtrai_vec(Kv2,Qv1,V2t),
  alglin:cons_vec(X2t,Cons1,X2tt),
  alglin:cons_vec(V2t,Cons1,V2tt),
  alglin:soma_vec(X1,X2tt,X2),
  alglin:soma_vec(V1,V2tt,V2),

  Cons2 is (2.0e0-sqrt(2.0e0)),
  Cons3 is -(2.0e0-1.5e0*sqrt(2.0e0)),
  alglin:cons_vec(Kx2,Cons2,Qx2t),
  alglin:cons_vec(Kv2,Cons2,Qv2t),

```



```

comeca_din(9,X,V,Len,Tmax):-
write_out(X,1,Len),

% variacao no tempo
drk:param(T1),
append(X,Xo),
append(V,Vo),
Tini is T1,

loop_tempo_epred(Tini,Tmax,[Xo,Vo],Len,1),
true.

% lembra que tem que inicializar
loop_tempo_epred(Tcorr,Tmax,[Xo,Vo],Len,Prim):-
(
Tmax<Tcorr,
write('pronto'),nl,
!
;
drk:param(Tpass),

mag:appendinv(Xo,Xof),
forbt(Xof,1),
loop_atomos_foper(Len,[Vo],[Fxo,Fvo]),
assert(for_rkg(Prim,[Fxo,Fvo])),

euler_predictor([Xo,Vo],Prim,Tpass,[Xpred,Vpred]),

mag:appendinv(Xpred,Xpredf),
forbt(Xpredf,1),
loop_atomos_foper(Len,[Vpred],[Fx2,Fv2]),
Prim is Prim+1,
assert(for_rkg(Prim,[Fx2,Fv2])),

euler_corrector([Xo,Vo],Prim,Tpass,[Xf,Vf]),

mag:appendinv(Xf,Xff),
mag:appendinv(Vf,Vff),
forbt(Xff,1),
loop_atomos_foper(Len,[Vf],[Fx2f,Fv2f]),
fase_esc(Xff,Vff,[Tcorr]),

retract(for_rkg(Prim,_)),!,
assert(for_rkg(Prim,[Fx2f,Fv2f])),
retract(for_rkg(Prim,_)),!,

Prim2 is Prim+1,
Tcorr2 is Tcorr+Tpass,
loop_tempo_epred(Tcorr2,Tmax,[Xf,Vf],Len,Prim2)
).

euler_predictor([Xo,Vo],Prim,Tpass,[Xf,Vf]):-
for_rkg(Prim,[Fx1,Fv1]),!,

Const is Tpass,

alglin:cons_vec(Fx1,Const,Xc),
alglin:cons_vec(Fv1,Const,Vc),

alglin:soma_vec(Xo,Xc,Xf),
alglin:soma_vec(Vo,Vc,Vf).

euler_corrector([Xo,Vo],Prim,Tpass,[Xf,Vf]):-
for_rkg(Prim,[Fx1,Fv1]),
P2 is Prim+1,
for_rkg(P2,[Fx2,Fv2]),!,

alglin:soma_vec(Fx1,Fx2,Fxt),

```



```
Tcorr2 is Tcorr+Tpass,
loop_tempo_admou4(Tcorr2,Tmax,[Xf,Vf],Len,Prim2)
).
```

```
adams_corrector4([Xo,Vo],Prim,Tpass,[Xf,Vf]):-
```

```
P2 is Prim+1,
for_rkg(P2,[Fx2,Fv2]),
P3 is P2+1,
for_rkg(P3,[Fx3,Fv3]),
P4 is P3+1,
for_rkg(P4,[Fx4,Fv4]),
P5 is P4+1,
for_rkg(P5,[Fx5,Fv5]),!,
```

```
Cons1 is -5.0e0,
alglin:cons_vec(Fx3,Cons1,Fx3t),
alglin:cons_vec(Fv3,Cons1,Fv3t),
Cons2 is 19.0e0,
alglin:cons_vec(Fx4,Cons2,Fx4t),
alglin:cons_vec(Fv4,Cons2,Fv4t),
alglin:cons_vec(Fx5,9.0e0,Fx5t),
alglin:cons_vec(Fv5,9.0e0,Fv5t),
```

```
alglin:soma_vec(Fx2,Fx3t,Fxttt),
alglin:soma_vec(Fxttt,Fx4t,Fxtttt),
alglin:soma_vec(Fxtttt,Fx5t,Fxttttt),
alglin:soma_vec(Fv2,Fv3t,Fvttt),
alglin:soma_vec(Fvttt,Fv4t,Fvtttt),
alglin:soma_vec(Fvtttt,Fv5t,Fvttttt),
```

```
ConsT is Tpass/24.0e0,
```

```
alglin:cons_vec(Fxttttt,ConsT,Xcorrt),
alglin:cons_vec(Fvttttt,ConsT,Vcorrt),
```

```
alglin:soma_vec(Xo,Xcorrt,Xf),
alglin:soma_vec(Vo,Vcorrt,Vf).
```

```
adams_predictor4([Xo,Vo],Prim,Tpass,[Xpred,Vpred]):-
```

```
for_rkg(Prim,[Fx1,Fv1]),
P2 is Prim+1,
for_rkg(P2,[Fx2,Fv2]),
P3 is P2+1,
for_rkg(P3,[Fx3,Fv3]),
P4 is P3+1,
for_rkg(P4,[Fx4,Fv4]),!,
```

```
% Lembrar Fx5 e o mais recente
```

```
Cons1 is -9.0e0,
alglin:cons_vec(Fx1,Cons1,Fx1t),
alglin:cons_vec(Fv1,Cons1,Fv1t),
alglin:cons_vec(Fx2,37.0e0,Fx2t),
alglin:cons_vec(Fv2,37.0e0,Fv2t),
Cons2 is -59.0e0,
alglin:cons_vec(Fx3,Cons2,Fx3t),
alglin:cons_vec(Fv3,Cons2,Fv3t),
alglin:cons_vec(Fx4,55.0e0,Fx4t),
alglin:cons_vec(Fv4,55.0e0,Fv4t),
```

```
alglin:soma_vec(Fx1t,Fx2t,Fxtt),
alglin:soma_vec(Fxtt,Fx3t,Fxttt),
alglin:soma_vec(Fxttt,Fx4t,Fxtttt),
alglin:soma_vec(Fv1t,Fv2t,Fvtt),
alglin:soma_vec(Fvtt,Fv3t,Fvttt),
alglin:soma_vec(Fvttt,Fv4t,Fvtttt),
```

```
ConsT is Tpass/24.0e0,
```



```

RR is sqrt(Rij1*Rij1+Rij2*Rij2+Rij3*Rij3),

Harjou is 4.35974394e-18,
Unitcon is 1e-10*Harjou,
Cm is 1.660538921e-27,
U is Unitcon/Cm,

Hder is 1.0e-6,

RR1 is RR-Hder,
RR2 is RR+Hder,

v_breno(RR1,Vx1),
v_breno(RR2,Vx2),

Dedr is (Vx2-Vx1)/(2*Hder),

Drxi1 is -U*(Dedr*Rij1)/RR,
Drxi2 is -U*(Dedr*Rij2)/RR,
Drxi3 is -U*(Dedr*Rij3)/RR,
Drxj1 is U*(Dedr*Rij1)/RR,
Drxj2 is U*(Dedr*Rij2)/RR,
Drxj3 is U*(Dedr*Rij3)/RR,

ass_forca_j(Ind,1,2,[[Drxi1,Drxi2,Drxi3],[Drxj1,Drxj2,Drxj3]]),

v_breno(RR,Ener),

retractall(enerpot(_)),
assert(enerpot(Ener)).

v_breno(X,Vx):-
    Re is 0.74,
    A1 is 2.56163,
    A2 is 0.208116,
    A3 is 3.01648,
    D is 0.166062,
    G0 is 1.23894,
    G1 is 12.7519,
    G2 is 0.0624156,

    Vx is (-D/X)*(1+A1*(X-Re) + A2*(X-Re)**2 + A3*(X-Re)**3)*exp(-(G0*(1+G1*tanh(G2*(X-Re))))*(X-Re)).

```

```

% ROTINA QUE CALCULA A FORCA RESULTANTE EM TODOS OS ATOMOS

```

```

forb2(Xcord,Indice):-
    tipos(Tipos),
    roda_gam(Xcord,Tipos,Fvec,Ener),
    retractall(enerpot(_)),
    assert(enerpot(Ener)),
    length(Xcord,Len),
    Unitcon is -4.961474e-1,
    mult_forca(Unitcon,Fvec,[],Fvec2),
    ass_forca_j(Indice,1,Len,Fvec2).

% MULTIPLICA PELA UNIDADE
mult_forca(_,[],Fvec2,Fvecf):-
    reverse(Fvec2,Fvecf),
    !.
mult_forca(U,[F|Fcont],Ftemp,Ffim):-
    agliin:cons_vec(F,U,F2),
    mult_forca(U,Fcont,[F2|Ftemp],Ffim).

```

```

% COLOCA FORCAS

```

```

ass_forca_j(,J,Len,Fvec):-
  J>Len,
  !,
  Fvec=[],!.
ass_forca_j(Ind,J,Len,[F|Fcont]):-
  retractall(forca_j(Ind,J,_)),
  assert(forca_j(Ind,J,F)),
  J2 is J+1,
  ass_forca_j(Ind,J2,Len,Fcont).

% ROTINA QUE TEM COMO DADO DE ENTRADA TODAS AS COORDENADAS, E CALCULA
% A FORÇA RESULTANTE EM J.
% PARA USAR A TERCEIRA LEI DE NEWTON USAREI A SEGUINTE LÓGICA, SE
% JÁ FOI CALCULADO USA E RETIRA DA MEMORIA. SE NÃO FOI CALCULA E COLOCA
% NA MEMÓRIA.
% Vou usar a interação na memória por enquanto, inevitável
% for(Xall uma lista onde cada elemento são as coordenadas dos átomos
%   Vtodos todas as velocidades,Ftemp [0,0,0],Fsai força de saída,
%   K contador -> 1, J posição átomo o qual estou calculando as forças
%   , Len)
forb(,F,F,K,Len):-
  K>Len,
  !.
forb(Xall,_,Ftemp,Fsai,J,J,Len):-
  K2 is J+1,
  forb(Xall,_,Ftemp,Fsai,K2,J,Len),!.
forb(Xall,_,Ftemp,Fsai,K,J,Len):-
  forca_ij(K,J,Fold),
  subtrai_vec(Ftemp,Fold,3,Ftemp2),
  retract(forca_ij(K,J,Fold)),
  K2 is K+1,
  forb(Xall,_,Ftemp2,Fsai,K2,J,Len),!.
forb(Xall,_,Ftemp,Fsai,K,J,Len):-
  int(J,K,Tipo),
  for_tj(Xall,_,Ftemp,Ftemp2,K,J,Tipo),
  K2 is K+1,
  forb(Xall,_,Ftemp2,Fsai,K2,J,Len).

% CALCULA A FORÇA DE K EM J
% for_tj(+Xall vetor com todas as coordenadas,+Vall vetor com as
%   velocidades,+Ftemp as forças que as outras partículas
%   faziam,-Ftemp2 força anterior mais essa,+K particula que
%   atua,+J particula passiva, +Tipo um valor que define o
%   potencial).
for_tj(Xall,_,Ftemp,Ftemp2,K,J,1):-
  nth1(J,Xall,Jcoord),
  nth1(1,Jcoord,X1),
  nth1(2,Jcoord,Y1),
  nth1(3,Jcoord,Z1),
  nth1(K,Xall,Kcoord),
  nth1(1,Kcoord,X2),
  nth1(2,Kcoord,Y2),
  nth1(3,Kcoord,Z2),
  R3 is ((X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2)+(Z1-Z2)*(Z1-Z2))^(3/2),
  F1 is -(X1-X2)/R3,
  F2 is -(Y1-Y2)/R3,
  F3 is -(Z1-Z2)/R3,
  assert(forca_ij(J,K,[F1,F2,F3])),
  soma_vec(Ftemp,[F1,F2,F3],3,Ftemp2),!.
for_tj(Xall,_,Ftemp,Ftemp2,K,J,2):-
  nth1(J,Xall,Jcoord),
  nth1(1,Jcoord,X1),
  nth1(2,Jcoord,Y1),
  nth1(3,Jcoord,Z1),
  nth1(K,Xall,Kcoord),
  nth1(1,Kcoord,X2),
  nth1(2,Kcoord,Y2),
  nth1(3,Kcoord,Z2),
  R3 is ((X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2)+(Z1-Z2)*(Z1-Z2))^(3/2),
  F1 is (X1-X2)/R3,

```

```

F2 is (Y1-Y2)/R3,
F3 is (Z1-Z2)/R3,
assert(forca_ij(J,K,[F1,F2,F3]),
soma_vec(Ftemp,[F1,F2,F3],3,Ftemp2),!).
for_tj(Xall,_,Ftemp,Ftemp2,K,J,3):-
nth1(J,Xall,Jcoord),
nth1(1,Jcoord,X1),
nth1(2,Jcoord,Y1),
nth1(3,Jcoord,Z1),
nth1(K,Xall,Kcoord),
nth1(1,Kcoord,X2),
nth1(2,Kcoord,Y2),
nth1(3,Kcoord,Z2),
R3 is ((X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2)+(Z1-Z2)*(Z1-Z2))^(3/2),
F1 is (X1-X2)/R3,
F2 is (Y1-Y2)/R3,
F3 is (Z1-Z2)/R3,

```

```

%%%% SEM FORÇA MAGNÉTICA TERMINA AQUI:
assert(forca_ij(J,K,[F1,F2,F3]),
soma_vec(Ftemp,[F1,F2,F3],3,Ftemp2),!).

```

```

%%%% PARTE QUE INSERE UM TERMO DE FORÇA MAGNÉTICA

```

```

% força magnetica K é o que gera o campo magnético,
% J é o que atua, a velocidade fica do lado de fora.
% nth1(J,Vall,Jvel),
% nth1(1,Jvel,Vx1),
% nth1(2,Jvel,Vy1),
% nth1(3,Jvel,Vz1),
% nth1(K,Vall,Kvel),
% nth1(1,Kvel,Vx2),
% nth1(2,Kvel,Vy2),
% nth1(3,Kvel,Vz2),
% subtrai_vec([X2,Y2,Z2],[X1,Y1,Z1],3,R12),
% prod_int([Vx1,Vy1,Vz1],R12,Prodv1,3,0.0e0),
% prod_int([Vx1,Vy1,Vz1],[Vx2,Vy2,Vz2],Prodr12,3,0.0e0),
% cons_vec([Vx2,Vy2,Vz2],Prodv1,3,V1novo),
% cons_vec(R12,Prodr12,3,R12novo),
% subtrai_vec(V1novo,R12novo,3,Fmag),
% Magcons is (5.32793435e-5)/R3,
% cons_vec(Fmag,Magcons,3,[Fmag1,Fmag2,Fmag3]),
% write(K),tab(2),write(J),nl,
% write('força magnetica'),nl,
% write_list([Fmag1,Fmag2,Fmag3]),
% write('força eletrica'),nl,
% write_list([F1,F2,F3]),
% F1c is F1+Fmag1,
% F2c is F2+Fmag2,
% F3c is F3+Fmag3,
% assert(forca_ij(J,K,[F1c,F2c,F3c])),
% soma_vec(Ftemp,[F1c,F2c,F3c],3,Ftemp2).
%

```

```

% ROTINA QUE GRAVA NA MEMÓRIA O TIPO DE INTERAÇÃO E A MASSA.

```

```

% grava_info_tipo(+Tipo vetor com números que eu irei compreender,
% ? 1 iteração começa de 1, + Len comprimento)
% 1 eletron
% 2 proton
% a ordem desses números está relacionada a ordem das coordenadas
grava_info_tipo(_,J,Len):-
J>Len,!.
grava_info_tipo(Tipo,J,Len):-
nth1(J,Tipo,X),
grava_massa(X,J),
% grava_interacao(Tipo,1,J),
J2 is J+1,
grava_info_tipo(Tipo,J2,Len).

```

```

% SE FOR 1 E UM HIDROGENIO E ASSIM VAI
grava_massa(1,J):-

```

```

assert(part(J,'H',1.00782e0)),!.
grava_massa(5,J):-
assert(part(J,'B',11.00931e0)),!.
grava_massa(6,J):-
assert(part(J,'C',12.0000e0)).
grava_massa(7,J):-
assert(part(J,'N',14.00307e0)).
grava_massa(8,J):-
assert(part(J,'O',15.99491e0)).
grava_massa(9,J):-
assert(part(J,'F',18.99840322e0)).
grava_massa(17,J):-
assert(part(J,'Cl',34.96885e0)).
grava_massa(35,J):-
assert(part(J,'Br',78.91380e0)).
grava_massa(33,J):-
assert(part(J,'As',74.92160)).
grava_massa(100,J):-
assert(part(J,'Ti',1.0e0)).

% grava_interacao(+matriz com os tipos,? l começa de 1,+J em relacao
% a qual atomo serão gravadas,+Len tamanho).
grava_interacao(_,l,J):-
l>=J,!.
grava_interacao(Tipo,l,J):-
nth1(l,Tipo,Elem1),
nth1(J,Tipo,ElemJ),
grava_int_ij(Elem1,ElemJ,l,J),
l2 is l+1,
grava_interacao(Tipo,l2,J).

% 1 é atrativo 2 é repulsivo
% Se forem 2 elétrons a força inclui termo magnético
grava_int_ij(1,2,l,J):-
assert(int_i(l,J,1)),!.
grava_int_ij(2,1,l,J):-
assert(int_i(l,J,1)),!.
grava_int_ij(1,1,l,J):-
assert(int_i(l,J,3)),!.
grava_int_ij(2,2,l,J):-
assert(int_i(l,J,2)).

% Esse vai ficar
int(A,B,C):- int_i(A,B,C),!.
int(A,B,C):- int_i(B,A,C),!.

% ROTINA PARA ESCREVER NO OUTPUT NO FORMATO LIDO PELO MOLEKEL
% faz uso do predicado part(l,nome do átomo)
%write_out(Xcor,1,Len)
write_out(_,l,Len):-
l>Len,!.
write_out(X,1,Len):-
arqsai([Mo]),
write(Mo,' '),write(Mo,Len),nl(Mo),
write(Mo,'titulo'),nl(Mo),
part(1,Name,_),
nth1(1,X,X1),
nth1(1,X1,Xa),
nth1(2,X1,Xb),
nth1(3,X1,Xc),
write(Mo,Name),write(Mo,' '),
write(Mo,Xa),write(Mo,' '),write(Mo,Xb),write(Mo,' '),
write(Mo,Xc),nl(Mo),
write_out(X,2,Len),
!.
write_out(X,l,Len):-
arqsai([Mo]),
nth1(l,X,Xl),

```

```

nth1(1,Xl,Xa),
nth1(2,Xl,Xb),
nth1(3,Xl,Xc),
part(l,Name,_),
write(Mo,Name),write(Mo,' '),
write(Mo,Xa),write(Mo,' '),write(Mo,Xb),write(Mo,' '),
write(Mo,Xc),nl(Mo),
l2 is l+1,
write_out(X,l2,Len).

```

```

% write_out2 so pra teste
write_out2([[Xi1,Xi2,Xi3],[Xj1,Xj2,Xj3]]):-
arqsai([Mo]),
write(Mo,Xi1),write(Mo,'; '),
write(Mo,Xi2),write(Mo,'; '),
write(Mo,Xi3),write(Mo,'; '),
write(Mo,Xj1),write(Mo,'; '),
write(Mo,Xj2),write(Mo,'; '),
write(Mo,Xj3),nl(Mo).

```

```

% Escreve no formato do molekel, tudo e hidrogenio
% o arquivo e sempre teste.xyz
write_out3(X):-
length(X,Len),
open('teste.xyz',write,Arq),
write_out3(X,1,Len,Arq),
close(Arq).

```

```

write_out3(X,l,Len,Mo):-
(
l>Len,
!
;
l=1,
write(Mo,' '),write(Mo,Len),nl(Mo),
write(Mo,'titulo'),nl(Mo),
nth1(l,X,X1),
nth1(1,X1,Xa),
nth1(2,X1,Xb),
nth1(3,X1,Xc),
write(Mo,'H '),
write(Mo,' '),
write(Mo,Xa),
write(Mo,' '),
write(Mo,Xb),
write(Mo,' '),
write(Mo,Xc),nl(Mo),
write_out3(X,2,Len,Mo),
!
;
nth1(l,X,X1),
nth1(1,X1,Xa),
nth1(2,X1,Xb),
nth1(3,X1,Xc),
write(Mo,'H '),
write(Mo,' '),
write(Mo,Xa),
write(Mo,' '),
write(Mo,Xb),
write(Mo,' '),
write(Mo,Xc),nl(Mo),
l2 is l+1,
write_out3(X,l2,Len,Mo)
).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                UTILIDADES                %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

% ADICIONA O TERMO I A POSIÇÃO I DA LISTA.
% adiciona(Lista in, Imuleta=1, posição i a ser colocado,
%      Termo, Lista saida)
% se for maior coloca zero no caminho.
% !!! Se a linha for vazia vai dar um pau
%

```

```

adiciona([_|Lires],1,1,Term,[Term|Lires):-!.
adiciona([X|Lires],I,Iterm,Term,[X|Lout):-
    I2 is I+1,
    Iterm==I2,
    Lires=[_|B],
    Lout=[Term|B],
    !.
adiciona([X|Lires],I,Iterm,Term,[X|Lout):-
    I2 is I+1,
    adiciona(Lires,I2,Iterm,Term,Lout).

```

```

% ROTINA QUE REALIZA UMA SOMA BÁSICA DE VETORES
% Soma_vec(vec 1, vec 2, tamanho, resultado)
soma_vec(____,0,[]):- !.
soma_vec([X|Vec1],[Y|Vec2],I,[Z|Vecsai):-
    Z is X+Y,
    I2 is I-1,
    soma_vec(Vec1,Vec2,I2,Vecsai).

```

```

% ROTINA QUE REALIZA UMA SUBTRAÇÃO BÁSICA DE VETORES
% Soma_vec(+vec 1, +vec 2 vai ser trocado o sinal
%      ,+iteração,+tamanho, -resultado)
subtrai_vec(____,0,[]):- !.
subtrai_vec([X|Vec1],[Y|Vec2],I,[Z|Vecsai):-
    Z is X-Y,
    I2 is I-1,
    subtrai_vec(Vec1,Vec2,I2,Vecsai).

```

```

% ROTINA QUE ESCREVE TODOS OS TERMOS DE UMA LISTA
% write_list(Lista de entrada,tamanho)
write_list([]):-nl,!.
write_list([X|Lrest]):-
    write(X),tab(4),
    write_list(Lrest).

```

```

% PRODUTO INTERNO ENTRE 2 LISTAS
% prod_int(Lista 1,Listas 2, saida,iteracao-len,
%      muleta da saida-0).
% a muleta e pra ajdar a tail recursividade.
% A iteração deve começar do maximo
% A muleta tem que começar do 0
prod_int(____,Sai,0,Sai):-!.
prod_int([Xi|L1res],[Yi|L2res],Sai,Ite1,Saitem1):-
    Saitem2 is Saitem1+Xi*Yi,
    Ite2 is Ite1-1,
    prod_int(L1res,L2res,Sai,Ite2,Saitem2).

```

```

% ROTINA QUE MULTIPLICA UM VETOR POR UMA CONSTANTE
% cons_vec(Vetor, contante, tamanho, Saida)
cons_vec(____,0,[]):- !.
cons_vec([X|Vconin],Alfa,I,[Y|Vcont]):-
    Y is Alfa*X,
    I2 is I-1,
    cons_vec(Vconin,Alfa,I2,Vcont).

```

```

%Rotina que tem como saída um string no seguinte formato
%'Ano_dia_mês_hora_minutos_segundos'

```



```

%criar_nome(-Data)
criar_nome(Sai10):-
  get_time(T1),
  stamp_date_time(T1,date(Ano,Mes,Dia,Hgreen,Min,Segf,_,_,_),'UTC'),
  Hora is Hgreen-3,
  Seg is truncate(Segf),
  Under = '_',
  atom_number(A,Ano),
  atom_number(Ms,Mes),
  atom_number(D,Dia),
  atom_number(H,Hora),
  atom_number(Mn,Min),
  atom_number(S,Seg),
  string_concat(A,Under,Sai1),
  string_concat(Sai1,D,Sai2),
  string_concat(Sai2,Under,Sai3),
  string_concat(Sai3,Ms,Sai4),
  string_concat(Sai4,Under,Sai5),
  string_concat(Sai5,H,Sai6),
  string_concat(Sai6,Under,Sai7),
  string_concat(Sai7,Mn,Sai8),
  string_concat(Sai8,Under,Sai9),
  string_concat(Sai9,S,Sai10).

```

```

% ROTINA QUE IMPRIME A FASE, A ENERGIA TOTAL E QUALQUER
% OUTRA INFORMACAO UTIL.

```

```

fase_esc(Xo,Vo,[Tcorr]):-
  (
    drk:param(Tpass),
    drk:fase(Fas),
    Tesc is truncate(Tcorr/Tpass),
    Tesc2 is Tesc mod Fas,
    Tesc2 == 0,

    length(Xo,Len),
    write_out(Xo,1,Len),

    drk:sistema(Sistema),

    (
      Sistema='planetario',
      Vo=[[Vx,Vy,Vz]],
      Ecin is 0.5*sqrt(Vx*Vx+Vy*Vy+Vz*Vz),
      !
    ;
      ener_cin(1,Len,Vo,0.0e0,Ecin)
    ),

    enerpot(Epot),
    enlog(Enlog),
    Etotal is Ecin+Epot,

    write(Enlog,Etotal),write(Enlog,' ; '),
    write(Enlog,Epot),write(Enlog,' ; '),
    % write(Enlog,R),write(Enlog,' ; '),
    % write(Enlog,Req),write(Enlog,' ; '),
    % write(Enlog,O),write(Enlog,' ; '),
    % write(Enlog,Oeq),write(Enlog,' ; '),
    imp_tud(1,Len,Xo,Vo,Enlog),
    nl(Enlog),
    !
  ;
    true
  ).

```

```

% ROTINA QUE CALCULA A ENERGIA CINETICA
ener_cin(l,Len,_,Ec,Ec):-

```

```

I>Len,
!.
ener_cin(I,Len,V,Ectem,Ec):-
nth1(I,V,Vi),
part(I,_,Mass),

algiin:prod_int(Vi,Vi,Vi2),

Cm is 1.660538921e-27,
Jpeh is 2.29371257e17,
% A energia final vai estar em hartree mesmo
% Jpeh e o fator de conversao.

Ectem2 is Ectem+0.5*Mass*Cm*Vi2*1.0e10*Jpeh,
I2 is I+1,
ener_cin(I2,Len,V,Ectem2,Ec).

imp_tud(I,Len,X,V,Enlog):-
(
I>Len,
!
;
nth1(I,X,[Xi1,Xi2,Xi3]),
nth1(I,V,[Vi1,Vi2,Vi3]),

write(Enlog,Xi1),write(Enlog,' ; '),
write(Enlog,Xi2),write(Enlog,' ; '),
write(Enlog,Xi3),write(Enlog,' ; '),
write(Enlog,Vi1),write(Enlog,' ; '),
write(Enlog,Vi2),write(Enlog,' ; '),
write(Enlog,Vi3),write(Enlog,' ; '),

I2 is I+1,
imp_tud(I2,Len,X,V,Enlog)
).

```

-> forcagam.pl

```

% Rotina que executa o gamess e pega o gradiente.

:-module(forcagam,[roda_gam/4]).

c([[-5,0,0],[0,0,0]],[1,1]).

atomo(1,'H',1.0).
atomo(5,'B',5.0).
atomo(6,'C',6.0).
atomo(7,'N',7.0).
atomo(8,'O',8.0).
atomo(9,'F',9.0).
atomo(17,'Cl',17.0).
atomo(35,'Br',35.0).
atomo(33,'As',33.0).

% Rotina que entra com as coordenadas e os tipos de atomos
% roda o gamess e pega o gradiente. ( T e tipos)
roda_gam(X,T,Fvec,Ener):-
esc_input(X,T)->
deleta_anterior,
drk:camin_gam(Gampath),
drk:nome_arq(Narq),
drk:versao_gam(Vergam),
drk:camin_grad(Gradpath),
drk:camin_ener(Enerpath),

```

```

atom_concat(Narq, '.inp', Narqinp),
atom_concat(Narq, '.log', Narqlog),

atom_concat(Gampath, Narqinp, GP2),
atom_concat(GP2, ' ', GP3),
atom_concat(GP3, Vergam, GP4),
atom_concat(GP4, ' > ', GP5),
atom_concat(GP5, Narqlog, Gampathf),

shell(Gampathf)->
learq(Gradpath, L),
learq(Enerpath, [[Ener]]),
forca_l_vec(L, Fvec),
!.
roda_gam(____):-
write("WARNING!!!"),nl,
write("WARNING!!!"),nl,
write("WARNING!!!"),nl,
write('algum false na hora de rodar o gameess, problema serio'),nl.

% Adaptacao do formato do gradiente.
forca_l_vec([], []):- !.
forca_l_vec([[__, Fxi, Fyi, Fzi] | Flcont], [[Fxi, Fyi, Fzi]|Fvecont]]:-
forca_l_vec(Flcont, Fvecont).

deleta_anterior:-
drk:camin_dat(Datpath),
drk:camin_grad(Gradpath),
drk:camin_ener(Enerpath),

(
exists_file(Datpath),
delete_file(Datpath),
!
;
true
),
(
exists_file(Gradpath),
delete_file(Gradpath),
!
;
true
),
(
exists_file(Enerpath),
delete_file(Enerpath),
!
;
true
).

% Rotinas que trabalham com leitura e escrita de arquivos nao saia
% de casa sem elas.
% learq(+file arquivo de entrada, -L grande lista de saida)
learq(File, L):-
open(File, read, Xler),
learq(Xler, 1, [], L),
close(Xler).

learq(Stream, N, Ac, L):-
(
at_end_of_stream(Stream)->
reverse(Ac, L),
!
;
read_line_to_codes(Stream, Codes),
% write(Codes),nl,

```

```

% Deletando caracteres que atrapalham a execucao
% delete(Codes, 40, Codes2),
% delete(Codes2, 41, Codes3),
% caracteres deletados
atom_codes(String, Codes),
downcase_atom(String,String2),
% write(String2),nl,
atomic_list_concat(Lista, '', String2),
% write('estou aqui'),nl,
delete(Lista, "", Lista2),
% write('estou aqui'),nl,
maplist(term_to_atom, Lista3, Lista2),
% write(Lista2),nl,
N2 is N+1,
!,
learq(Stream, N2, [Lista3]Ac, L)
).
```

```

esc_vec(File):-
open(File, read, Xler),
esc_vec(Xler, 1,1),
close(Xler).
```

```

esc_vec(Stream, N,Flag):-
(
at_end_of_stream(Stream)->
!
;
read_line_to_codes(Stream, Codes),
% write(Codes),nl,
atom_codes(String, Codes),
% write(String),nl,
N2 is N+1,

(
Codes=[32,36,86,69,67|_],
%String=' $VEC',
write(String),nl,
esc_vec(Stream, N2,2),
!
;
Flag=2,
Codes=[32,36,69,78,68|_],
%String=' $END',
write(String),nl,
esc_vec(Stream,N2, 3),
!
;
Flag=3,
esc_vec(Stream,N2, 3),
!
;
Flag=2,
write(String),nl,
esc_vec(Stream,N2, 2),
!
;
esc_vec(Stream,N2, 1)
)
).
```

```

% Rotina que escreve o input no formato do gamess.
% precisa das coordenadas no formato tradicional: [ [x1,y1,z1],[x2,y2,z2] ... ]
% tambem precisa do predicado atomo com o label e a carga
esc_input(X,Tipo):-
drk:nome_arq(Narq),
atom_concat(Narq, '.inp',Narqip),
tell(Narqip),
```

```

drk:camin_dat(Datpath),
(
  exists_file(Datpath),
  write(' $GUESS GUESS=MOREAD $END'),nl,
  !
;
  true
),
drk:cabecalho_games,
esc_inp_loop(X,Tipo),
write(' $END'),nl,
(
  exists_file(Datpath),
  esc_vec(Datpath),
  !
;
  true
),
told.

esc_inp_loop([],[]):-!.
esc_inp_loop([ [X,Y,Z]|Xcont],[Ti|Tcont]):-
  atomo(Ti,Label,Carga),
  write(Label),tab(2),write(Carga),tab(2),
  write(X),tab(2),
  write(Y),tab(2),
  write(Z),tab(2),
  nl,
  esc_inp_loop(Xcont,Tcont).

```

-> cond_ini.pl

```

% SUBROTINA RESPONSÁVEL POR GERAR CONDIÇÕES INICIAIS
% REQUER UM CÁLCULO DE FREQUÊNCIA, AS UNIDADES E OPERAÇÕES
% ESTÃO EM CONFORMIDADE COM O PACOTE GAMES.
% NESSE MOMENTO USA ELEMENTOS DA MEMÓRIA COMO x0,mass,v

:-module(mag,[ini_sim/2]).

:-use_module(alglin).

:-dynamic(prmode/2),dynamic(vn/1).

% ROTINA QUE GERA UM VETOR COM A PROBABILIDADE DO SISTEMA TER AQUELE
% MODO NORMAL.
% EM RESUMO
% iteracao i:
% Define um modo normal, imod.
% -- iteracao j:
% -- define seta o numero quantico como nj
% -- calcula a probabilidade do sistema estar
% no modo normal nj. e usada a distribuicao
% de boltzmann e a frequencia do modo normal.
% -- nj=nj+1
% -- para quando a probabilidade do sistema apresentar
% esse modo for inferior a 1e-6.
% adiciona na memoria o seguinte fato:
% prmod(Imod,prfim)

```

```

% sendo Prfim uma lista, o primeiro elemento e a probabilidade do
% sistema apresentar o numero quantico 0, o segundo elemento
% se refere ao numero quantico 1, e a assim por diante.
prob_todos(T,Freq):-
    retractall(prmode(_,)),
    length(Freq,Imax),
    prob_todos_i(1,Imax,T,Freq).

prob_todos_i(I,Imax,_,):-
    I>Imax,
    !.
prob_todos_i(I,Imax,T,Freq):-
    prob(0,[],I,T,Freq),
    I2 is I+1,
    prob_todos_i(I2,Imax,T,Freq).

prob(I,Plis,Imod,Temp,Freq):-
    C=1.43877696,
    nth1(Imod,Freq,Ni),
    Prob is exp(-(I*Ni*C)/Temp)*(1-exp(-(Ni*C)/Temp)),
    % write(I),tab(2),write(Prob),nl,
    (
        Prob<1e-6,
        reverse([0|Plis],Prfim),
        assert(prmode(Imod,Prfim)),
        !
    );
    I2 is I+1,
    prob(I2,[Prob|Plis],Imod,Temp,Freq)
).

%set_random(seed(nseed))
% SUBROTINA pr_ener
% NECESSITA DO FATO NA MEMÓRIA prmode, GERADO PELA ROTINA
% PROB_TODOS.
% NESSA ROTINA UM NUMERO ALEATORIO É GERADO, A PARTIR DA
% LISTA DE PROBABILIDADE É POSSIVEL DESCOBRIR A QUAL NUMERO
% QUANTICO ELE SE REFERE.
% pr_ener(+Imod: modo normal desejado,-Ini2: numero quantico)
% Obs: a temperatura já foi contemplada no prob_todos.
pr_ener(Imod,Ini2):-
    prmode(Imod,Prlis),
    Rnd is random_float,
    (
        Prlis=[P1|_],
        Rnd<P1,
        % write(Rnd),tab(2),write(P1),tab(2),write('primeiro'),
        Ini2=0,
        !
    );
    pr_ener_i(2,Rnd,Prlis,Ini),
    Ini2 is Ini-1
).

pr_ener_i(Icorr,Rnd,Prlis,Ini):-
    nth1(Icorr,Prlis,Pi),
    Pic is 1-Pi,
    (
        Pic>Rnd,
        % write(Rnd),tab(2),write(Pi),tab(2),write('foi esse'),nl,
        Ini=Icorr,
        !
    );
    Icorr2 is Icorr+1,
    pr_ener_i(Icorr2,Rnd,Prlis,Ini)
).

ini_sim(X,V):-
    T=300,
    % Gerando condicoes iniciais para agua

```

```

% borx0(Borx0),
% borfreq(Borfreq),
% bormred(Bormred),
% borhess(Borhess),
% bormass(Bormass),
% Bortrot=[1,2,3,4,5,6],

ch3brx0(Cx0),
ch3brfreq(Cfreq),
ch3brmred(Cmred),
ch3brhess(Chess),
ch3brmass(Cmass),
Ctrot=[1,2,3,4,5,6],

gera_mol(Xn,Vtemp,[T,Cx0,Cfreq,Cmred,Chess,Cmass,Ctrot]),

% Corrige o momento angular
corr_m_ang(Xn,Vtemp,Cmass,Vn),

% ADICIONANDO O ATOMO DE FLUOR

% posicao

Xflu = [-3,0.5,0],

% velocidade

Velflu = [0.005901621,0,0],

%

append(Xn,[Xflu],Xini),
append(Vn,[Velflu],Vt),
Massflu is sqrt(34.96885),
append(Cmass,[Massflu],Masstudo),

trans_rig(Vt,V,Masstudo,_),

centraliza(Xini,Masstudo,X),

% ate agora esta saindo X e V quero adicionar o fluor

%
% agux0(Agux0),
% agufreq(Agufreq),
% agumred(Agumred),
% aguress(Aguress),
% agumass(Agumass),
% Agutrot=[1,2,3,4,5,6],
%
gera_mol(Xag,Vag,[T,Agux0,Agufreq,Agumred,Aguress,Agumass,Agutrot]),
%
%
Xag=[Xo,Yo,Zo],[Xh1,Yh1,Zh1],[Xh2,Yh2,Zh2]],
%
%
Xon is Xo+4,
Xh1n is Xh1+4,
Xh2n is Xh2+4,
%
Vag=[[Vxo,Vyo,Vzo],[Vxh1,Vyh1,Vzh1],[Vxh2,Vyh2,Vzh2]],
%
%
Vxon is Vxo+0.01,
Vxh1n is Vxh1+0.01,
Vxh2n is Vxh2+0.01,
%
%
append(Xbor,[Xon,Yo,Zo],[Xh1n,Yh1,Zh1],[Xh2n,Yh2,Zh2]],X),
%

```

```

% append(Vbor, [[Vxon,Vyo,Vzo],[Vxh1n,Vyh1,Vzh1],[Vxh2n,Vyh2,Vzh2]],V),
%

% c_inic(Xiniag,Viniag,[T,Borx0,Borfreq,Bormred,Borhess,Bormass,Bortrot]),
% centraliza(Xiniag,Agumass,Xi),
% trans_rig(Viniag,Vi),
% rot_mol(Xi,Vi,X,V),

true.

gera_mol(X,V,[T,X0,Freq,Mred,Hess,Mass,Trot]):-
% Obtem as condicoes com toda aquela maquinaria de probabilidades
c_inic(X1,V1,[T,X0,Freq,Mred,Hess,Mass,Trot]),
% Coloca a molecula no centro de massa
centraliza(X1,Mass,X2),
% Retira o vetor de translacao rigida
trans_rig(V1,V2,Mass,_),
trans_rig(V2,_,Mass,_),
% Da um tapa na molecula, rodando ela em um eixo aleatorio
rot_mol(X2,V2,X,V),
% Ver se ta tudo em ordem.
trans_rig(V,_,Mass,_).

% ROTINA QUE GERA A AMPLITUDE E A VELOCIDADE DOS MODOS NORMAIS
% CONSIDERANDO O SISTEMA COMO OSCILADORES HARMÔNICOS CLÁSSICOS
% ACOPLADOS, DADO O NÚMERO QUÂNTICO, CONSEQUENTEMENTE A ENERGIA,
% É POSSÍVEL DESCOBRIR.
% O PRIMEIRO PASSO DESSA ROTINA E GERAR A LISTA DE PROBABILIDADES prmode
% O VETOR POSICAO E VELOCIDADE Q E QPONTO SAO DEFINIDOS COMO OSCILADORES
% HARMONICOS COM UMA DADA FASE.
% Obs: e possivel colocar todas as condicoes iniciais aqui sera feito no
% futuro.
c_inic(Xinic,Vinic,[T,X0,Freq,Mred,Hess,_,Trot]):-

% norm_mass(Hess,Mass,Hnorm),
Hess=Hnorm,

prob_todos(T,Freq),
length(Hnorm,Imax),
c_inic_vec(1,Imax,[],[],X,V,[Freq,Mred]),
% write(X),nl,
% write(V),nl,

trot(X,Trot,Xz),
trot(V,Trot,Vz),

matr_vec(Hnorm,Xz,Xm),
soma_vec(Xm,X0,Xin),
matr_vec(Hnorm,Vz,Vin),
appendinv(Xin,Xinic),
appendinv(Vin,Vinic).
% write(Xinic),nl,
% write(Vm),nl.

% LOOP DA ROTINA ANTERIOR
c_inic_vec(I,Imax,Xt,Vt,X,V,_) :-
I>Imax,
reverse(Xt,X),
reverse(Vt,V),
!.
c_inic_vec(I,Imax,Xtemp,Vtemp,X,V,[FreqI,_]):-
nth1(I,FreqI,Freq),
% nth1(I,MassI,Mass),

% pr_ener(I,Ni),
% fredmudar
Ni = 0,

```



```

Cf is 2.997924e10,
Cm is 1.660538921e-27,

Ener is (Ni+0.5)*Freq*1.98644568e-23,
Ampl is sqrt((2*Ener)/(Cm))/(2*pi*Freq*Cf),

% Foi feita a conversao para A e A/fs.
% NUMERO ALEATORIO
Aleanumeros =
[0.865,0.550,0.336,0.208,0.063,0.436,0.135,0.446,0.610,0.708,0.533,0.151,0.906,0.827,0.590,0.752,
0.116],

nth1(I,Aleanumeros,Rnd),

Q is Ampl*cos(2*pi*Rnd)*1e10,
Qponto is -2*pi*Ampl*Freq*Cf*sin(2*pi*Rnd)*1e-5,

I2 is I+1,
c_inic_vec(I2,Imax,[Q|Xtemp],[Qponto|Vtemp],X,V,[FreqI,_]).

% ROTINA QUE RETIRA OS MODOS NORMAIS DEFINIDOS COMO
% TRANSLACOES RIGIDAS.
trot(Lent,Ele0,Lsai):-
length(Lent,Len),
trot_i(1,Len,Lent,Ele0,Lsai).

trot_i(I,I,X,_,X):-
!.
trot_i(I,Len,Lent,Ele0,Lsai):-
(
not(Ele0=0),
Ele0=[E1|Elcont],
(
E1=I,
Lent=[_|Lcont],
Lsai=[0|Lsaicont],
I2 is I+1,
trot_i(I2,Len,Lcont,Elcont,Lsaicont),
!
;
Lent=[X|Lcont],
Lsai=[X|Lsaicont],
I2 is I+1,
trot_i(I2,Len,Lcont,Elcont,Lsaicont)
),
!
;
Lent=[X|Lcont],
Lsai=[X|Lsaicont],
I2 is I+1,
trot_i(I2,Len,Lcont,Elcont,Lsaicont)
).

% ROTINA QUE RETIRA A TRANSLACAO RIGIDA, ISSO
% E FEITO RETIRANDO UMA QUANTIDADE IDENTICA
% DE VELOCIDADE EM TODOS OS EIXOS CARTESIANOS.
trans_rig(V,V2,Mass,Mtotal):-
length(V,Len),
% write('mass: '),write(Mass),nl,
% write('V: '),write(V),nl,
trans_rig(1,Len,V,[0,0,0],[Tx,Ty,Tz],[Mass,0,Mtotal]),
% write('mtotal: '),write(Mtotal),nl,
% write('Tx: '),write(Tx),nl,
% Retirando o vetor de translacao rigida
Tx2 is Tx/Mtotal,
Ty2 is Ty/Mtotal,
Tz2 is Tz/Mtotal,

```

```

% Escrevendo na saida
% append('vcm.txt'),
% write('vcm: '),write([Tx2,Ty2,Tz2]),nl,
% Mod is sqrt(Tx2*Tx2+Ty2*Ty2+Tz2*Tz2),
% write('mod: '),write(Mod),nl,
% told,
% loop_centraliza(V,[Tx2,Ty2,Tz2],V2).

trans_rig(I,Len,V,[Tx,Ty,Tz],Tfim,[Mass,Msum,Mtotal]):-
(
    I>Len,
    Tfim = [Tx,Ty,Tz],
    Msum=Mtotal,
    !
;
    nth1(I,V,[Vx,Vy,Vz]),
    nth1(I,Mass,Matm),
    Tx2 is Tx+Vx*Matm*Matm,
    Ty2 is Ty+Vy*Matm*Matm,
    Tz2 is Tz+Vz*Matm*Matm,
    Msum2 is (Matm*Matm)+Msum,
    I2 is I+1,
    trans_rig(I2,Len,V,[Tx2,Ty2,Tz2],Tfim,[Mass,Msum2,Mtotal])
).

vcm_centraliza([],_,[],_):-!.
vcm_centraliza([[X,Y,Z]|Xentr],[Xcg,Ycg,Zcg],[[Xf,Yf,Zf]|Xfr],MM):-
MM=[[Mi|Masscont],Mtotal],
Xf is X-(Xcg*Mt*Mt)/Mtotal,
Yf is Y-(Ycg*Mt*Mt)/Mtotal,
Zf is Z-(Zcg*Mt*Mt)/Mtotal,
vcm_centraliza(Xentr,[Xcg,Ycg,Zcg],Xfr,[Masscont,Mtotal]).

% Rotina que centraliza as coordenadas do sistema no centro de massa
%centraliza(+Xent,+Mass,-Xsai)
centraliza(Xent,Mass,Xsai):-
length(Xent,Len),
loop_cengeo(1,Len,Xent,[0,0,0],Xcmass,[Mass,0]),

loop_centraliza(Xent,Xcmass,Xsai)
% write('centraliza'),nl,
% write(Xent),nl,nl,
% write(Xcmass),nl,nl,
% write(Xsai),nl
.

loop_cengeo(I,Len,_,[Xt,Yt,Zt],[Xcg,Ycg,Zcg],[_,Mtot]):-
I>Len,
Xcg is Xt/Mtot,
Ycg is Yt/Mtot,
Zcg is Zt/Mtot,
!.

loop_cengeo(I,Len,Xent,[Xt,Yt,Zt],Xcgeo,[Mass,Mtot]):-
nth1(I,Mass,Massatm),
Matm is Massatm*Massatm,
Mtot2 is Mtot+Matm,
% write(I),nl,
% write(Mass),nl,
% write(Mtot2),nl,

nth1(I,Xent,[X,Y,Z]),
Xt2 is Xt+X*Matm,
Yt2 is Yt+Y*Matm,
Zt2 is Zt+Z*Matm,
I2 is I+1,
loop_cengeo(I2,Len,Xent,[Xt2,Yt2,Zt2],Xcgeo,[Mass,Mtot2]).

loop_centraliza([],_,[],_):-!.
loop_centraliza([[X,Y,Z]|Xentr],[Xcg,Ycg,Zcg],[[Xf,Yf,Zf]|Xfr]):-
Xf is X-Xcg,

```

```

Yf is Y-Ycg,
Zf is Z-Zcg,
loop_centraliza(Xentr, [Xcg, Ycg, Zcg], Xfr) .

% GERANO UM VETOR UNITARIO PARA RODAR
alea_n(A1):-
    Sign is random(2),
    (
        Sign=0,
        A1 is random_float,
        !
    ;
        A1 is -random_float
    ).

% ROTINA QUE GERA UMA MATRIZ DE ROTACAO ALEATORIA
% HUE HUE BR BR
alea_m_rot(Mrot):-
    alea_n(X),
    alea_n(Y),
    alea_n(Z),
    Nor1 is sqrt(X*X+Y*Y+Z*Z),
    X2 is X/Nor1,
    Y2 is Y/Nor1,
    Z2 is Z/Nor1,
    alea_n(Tet),
    Tet2 is Tet*pi,
    matriz_rot_vet([X2, Y2, Z2], Tet2, Mrot) .

% ROTINA QUE FAZ UMA ROTACAO COMPLETAMENTE ALEATORIA
% NA MOLECULA, TEM QUE ESTAR NO CENTRO DE MASSA.
rot_mol(Xi, Vi, Xf, Vf):-
    transpoe(Xi, Xit),
    transpoe(Vi, Vit),

    alea_m_rot(Mrot),
    matr_matr(Mrot, Xit, Xiit),
    transpoe(Xiit, Xf),
    matr_matr(Mrot, Vit, Viit),
    transpoe(Viit, Vf) .

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CALCULO DO MOMENTO ANGULAR TOTAL E SUA CORRECAO %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
corr_m_ang(X, V, Mass, Vnovo):-
    length(X, Len),

    m_ang_loop(1, Len, [X, V, Mass], [0, 0, 0], Mang),

    append(X, Xappen),
    inert_tensor(Xappen, Mass, Inert),

    lu_inv(Inert, Inmen1),

    matr_vec(Inmen1, Mang, Omega),

    cons_vec(Omega, -1, Omega2),

    corrige_vel(1, Len, X, V, Omega2, [], Vnovo) .

% teste pra ver se funciona.
% m_ang_loop(1, Len, [X, Vnovo, Mass], [0, 0, 0], Mang2),
%
% write(Mang), nl,
% write(Mang2) .

```

```

corrige_vel(I,Len,X,V,Omega,Vtemp,Vfim):-
(
  I>Len,
  reverse(Vtemp,Vfim),
  !
;
  nth1(I,X,Xi),
  nth1(I,V,[Vix,Viy,Viz]),

  prod_vec(Omega,Xi,[Vixn,Viyn,Vizn]),

  Vix2 is Vix+Vixn,
  Viy2 is Viy+Viyn,
  Viz2 is Viz+Vizn,

  I2 is I+1,
  corrige_vel(I2,Len,X,V,Omega,[[Vix2,Viy2,Viz2]|Vtemp],Vfim)
).

m_ang_loop(I,Len,[X,V,Mass],[Lxt,Lyt,Lzt],Lfim):-
(
  I>Len,
  [Lxt,Lyt,Lzt]=Lfim,
  !
;
  nth1(I,X,Xi),
  nth1(I,V,Vi),
  nth1(I,Mass,Massi),
  Massi2 is Massi*Massi,

  cons_vec(Vi,Massi2,Pi),
  prod_vec(Xi,Pi,[Lxi,Lyi,Lzi]),

  Lxt2 is Lxt+Lxi,
  Lyt2 is Lyt+Lyi,
  Lzt2 is Lzt+Lzi,

  I2 is I+1,
  m_ang_loop(I2,Len,[X,V,Mass],[Lxt2,Lyt2,Lzt2],Lfim)
).

inert_tensor(X,Mass,Itensor):-
  length(X,Lent),
  Len is Lent/3,
  i_tens_loop([1,Len,Mass],X,[[0,0,0],[0,0,0],[0,0,0]],Itensor)
  %write('tensor: '),write(Itensor),nl
.

i_tens_loop([I,Len,_],_,[[Ixx,Ixy,Ixz],[_,Iyy,Iyz],[_,_,Izz]],[[Ixx,Ixy,Ixz],[Ixy,Iyy,Iyz],[Ixz,Iyz,Izz]]):-
  I>Len,
  !.
i_tens_loop([I,Len,M],[X,Y,Z|Lcont],[[Ixx,Ixy,Ixz],[_,Iyy,Iyz],[_,_,Izz]],Itens):-
  nth1(I,M,Mass2),
  Mass is Mass2*Mass2,

  Ixx2 is Ixx + Mass*(Y*Y + Z*Z),
  Iyy2 is Iyy + Mass*(X*X + Z*Z),
  Izz2 is Izz + Mass*(X*X + Y*Y),
  Ixy2 is Ixy + Mass*(-X*Y),
  Ixz2 is Ixz + Mass*(-X*Z),
  Iyz2 is Iyz + Mass*(-Y*Z),

  I2 is I+1,

```

```

i_tens_loop([I2,Len,M],Lcont,[Ixx2,Ixy2,Ixz2],[_,Iyy2,Iyz2],[_,_,Izz2]],Itens).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Rotina que gera uma matriz de rotaçao em torno de um vetor
% (Nao nos tradicionais angulos de Euler)
%matriz_rot_vet(+V vetor UNITARIO de referencia,+Tet angulo
%              desejado,-Mrot matriz de rotacao)
matriz_rot_vet([X,Y,Z],Tet,Mrot):-
    A11 is cos(Tet)+(1-cos(Tet))*X*X,
    A12 is (1-cos(Tet))*X*Y -sin(Tet)*Z,
    A13 is (1-cos(Tet))*X*Z + sin(Tet)*Y,
    A21 is (1-cos(Tet))*Y*X+sin(Tet)*Z,
    A22 is cos(Tet)+(1-cos(Tet))*Y*Y,
    A23 is (1-cos(Tet))*Y*Z-sin(Tet)*X,
    A31 is (1-cos(Tet))*Z*X - sin(Tet)*Y,
    A32 is (1-cos(Tet))*Z*Y+sin(Tet)*X,
    A33 is cos(Tet)+(1-cos(Tet))*Z*Z,
    Mrot=[[A11,A12,A13],[A21,A22,A23],[A31,A32,A33]].

% Rotina que desfaz o que o appendlist faz, pega uma lista
% e coloca no nosso formato xyz padrao
% appendinv(+Lcorr lista toda corrida,-Lxyz lista no formato xyz)
appendinv(Lcorr,Lxyz):-
    ainv_loop(Lcorr,Lxyz).

ainv_loop([],[]):-!.
ainv_loop([H1| [H2| [H3|Lres] ] ],[[H1,H2,H3]|Lxyz]):-
    ainv_loop(Lres,Lxyz).

% ROTINA PARA ESCREVER NO OUTPUT NO FORMATO LIDO PELO MOLEKEL
% Escreve a energia da configuracao no titulo
% faz uso do predicado part(I,nome do atomo)
write_out(Xcor,Ener):-
    open('mol.xyz',write,Mo),
    write_out(Xcor,Ener,Mo).

write_out(Xcor,Ener,Mo):-
    length(Xcor,Len),
    write_out(Xcor,Ener,1,Len,Mo).

write_out(_,_,I,Len,_):-
    I>Len,!.
write_out(X,Ener,1,Len,Mo):-
    write(Mo,' '),write(Mo,Len),nl(Mo),
    write(Mo,'energia charmm: '),
    write(Mo,Ener),
    nl(Mo),
    part(1,Name,_),
    nth1(1,X,X1),
    nth1(1,X1,Xa),
    nth1(2,X1,Xb),
    nth1(3,X1,Xc),
    write(Mo,Name),write(Mo,' '),
    write(Mo,Xa),write(Mo,' '),write(Mo,Xb),write(Mo,' '),
    write(Mo,Xc),nl(Mo),
    write_out(X,_,2,Len,Mo),
    !.
write_out(X,_,I,Len,Mo):-
    nth1(I,X,XI),
    nth1(1,XI,Xa),
    nth1(2,XI,Xb),
    nth1(3,XI,Xc),
    part(I,Name,_),
    write(Mo,Name),write(Mo,' '),
    write(Mo,Xa),write(Mo,' '),write(Mo,Xb),write(Mo,' '),
    write(Mo,Xc),nl(Mo),

```

```

I2 is I+1,
write_out(X,_, I2, Len, Mo) .

part(1, 'O', _) .
part(2, 'H', _) .
part(3, 'H', _) .

```

-> Alglin.pl

```

% Autor: Fred Vultor
% Datum: 03/11/2012

```

```

:-module(alglin,[
    prod_int/3,
    prod_vec/3,
    soma_vec/3,
    subtrai_vec/3,
    cons_vec/3,
    soma_matr/3,
    lu_inv/2,
    matr_vec/3,
    cons_matr/3,
    transpoe/2,
    matr_matr/3,
    gera_ident/3,
    gera_vec_zero/2
]).

```

```

% ROTINAS QUE USAM O MÉTODO LU PARA TRATAR SISTEMAS LINEARES,
% HÁ 2 OPÇÕES ATÉ AGORA QUE É A SOLUÇÃO DO SISTEMA, E O
% CÁLCULO DA MATRIZ INVERSA.

```

```

%!!! tem que tirar o assert

```

```

% OPÇÃO EFICIENTE DE CALCULAR A INVERSA DE UMA MATRIZ

```

```

%lu_inv(+Matriz de entrada, -Matriz de saída)

```

```

lu_inv(Aent,Ainv):-
% alea_matr(Aent),
dynamic(troca_ij/2),
lu_fator(Aent,LU),
nth1(1,LU,Lin),
length(Lin,Len),
gera_ident(1,Len,Id),
unscramb(Id,Nid,Len),
loop_inv(1,Len,LU,Nid,Atinv),
transpoe(Atinv,Ainv),
retractall(troca_ij(_,_)),!.

```

```

% ROTINA QUE DADA UMA MATRIZ LU E UMA MATRIZ IDENTIDADE ORGANIZADA

```

```

% DA MANEIRA CERTA COM POSSÍVEIS TROCAS DE LINHA,

```

```

%loop_inv(+1 Iteração, +Len Quantidade de colunas, +LU Matriz LU,

```

```

% +Nid Matriz identidade organizada, -Ainv a inversa)

```

```

loop_inv(J,Len,_,_,[]):-

```

```

J>Len,!.

```

```

loop_inv(J,Len,LU,Nid,[Ncolj|Acon]):-

```

```

pega_col(Nid,ColJ,1,J,Len),
for_sub(LU,ColJ,2,Len,Vecips),

```

```

back_sub(LU,Vecips,Len,Len,Ncolj),
J2 is J+1,
loop_inv(J2,Len,LU,Nid,Acon).

% ROTINA QUE OBTÉM A SOLUÇÃO DE UM SISTEMA DE EQUAÇÕES LINEARES
% USANDO O MÉTODO DIRETO DE DECOMPOSIÇÃO LU

% lu_method(matriz de entrada com os coeficientes, vetor com
% os termos independentes, vetor de saída).

lu_method(Aent,Vecb,Vecxis):-
% get_time(T1),
% alea_matr(Aent),
dynamic(troca_ij/2),
lu_fator(Aent,LU),
nth1(1,LU,Len),
length(Len,Len),
unscramb(Vecb,Vecb2,Len),
for_sub(LU,Vecb2,2,Len,Vecips),
back_sub(LU,Vecips,Len,Len,Vecxis),
retractall(troca_ij(_,_)),!.
% get_time(T2),
% T3 is T2-T1,
% write('tempo de execução '),write(T3).
lu_method(_,_):-
write('A ROTINA DE RESOLVER SISTEMAS LINEARES (VIA LU) FALHOU!'),
nl,!,fail.

% PROGRAMA QUE REALIZA UMA FATORAÇÃO LU
% SEPARANDO UMA MATRIZ EM DUAS: TRIANGULAR
% SUPERIOR E TRIANGULAR INFERIOR
lu_fator(Aent,LUout):-
nth1(1,Aent,Vetam),
length(Vetam,Len),
loop_j(Aent,LUout,Len,1).

% ESSE LOOP VARIA EM J REALIZANDO OS DOIS PROCEDIMENTOS DESCRITOS
% NO ARTIGO, A MATRIZ DE SAÍDA É UM ARGUMENTO QUE VAI FICANDO PARADO
% ATÉ O FINAL, ONDE ELA ATUALIZA COM A OUTRA QUE ESTÁ CORRENDO.
loop_j(LU,LU,Len,J):-
Lfim is Len+1,
J==Lfim,!.
loop_j(Aent,LUout,Len,J):-
loop_bet(Aent,LU1,Len,1,J),
J2 is J+1,
loop_alf(LU1,LU2,Len,J2,J),
loop_j(LU2,LUout,Len,J2).

%loop_bet
loop_bet(Aent,LU,_,J,J):-
pega_ij(J,J,Aent,Aij),
somat_kbet(Aent,Kbet,J,J,1,0.0e0),
% write('i e j '),write(J),tab(2),write(J),nl,
% write('beta -aij e kbet '),write(Aij),
% tab(2),write(Kbet),nl,
Betaj is Aij-Kbet,
Betaj=\=0,
adiciona_ij(J,J,Betaj,Aent,LU),!.
loop_bet(Aent1,LU,Len,J,J):-
write('precisou trocar'),nl,
pivoting(Aent1,J,Len,Aent),
pega_ij(J,J,Aent,Aij),
somat_kbet(Aent,Kbet,J,J,1,0.0e0),
% write('i e j '),write(I),tab(2),write(J),nl,
% write('beta -aij e kbet '),write(Aij),
% tab(2),write(Kbet),nl,
Betaj is Aij-Kbet,
adiciona_ij(J,J,Betaj,Aent,LU),!.
loop_bet(Aent,LUout,Len,I,J):-

```

```

pega_ij(I,J,Aent,Aij),
somat_kbet(Aent,Kbet,I,J,1,0.0e0),
% write('i e j '),write(I),tab(2),write(J),nl,
% write('beta -aij e kbet '),write(Aij),
% tab(2),write(Kbet),nl,
Betaij is Aij-Kbet,
adiciona_ij(I,J,Betaij,Aent,LU1),
I2 is I+1,
loop_bet(LU1,LUout,Len,I2,J).

```

```

loop_alf(LU,LU,Len,lfim,_) :-
    L2 is Len+1,
    lfim==L2,
    !.
loop_alf(Aent,LUout,Len,I,J) :-
    pega_ij(J,J,Aent,Bjj),
% write(Bjj),nl,
    Bjj\=0,
    pega_ij(I,J,Aent,Aij),
    somat_kalf(Aent,Kalf,I,J,1,0.0e0),
    Alfaij is (Aij-Kalf)/Bjj,
% write('i e j '),write(I),tab(2),write(J),nl,
% write('alfa -aij e kalf '),write(Aij),
% tab(2),write(Kalf),nl,
    adiciona_ij(I,J,Alfaij,Aent,LU1),
    I2 is I+1,
    loop_alf(LU1,LUout,Len,I2,J),!.
loop_alf(____):-
    write('erro no pivoting'),!,fail.

```

```

% somat_ip(g)(LU a matriz, Ksum saida,
%     I o termo corrente, J corrente, K começa de 1
%     ou 0, muleta começa de 0.0e0) :-
somat_kbet(_ ,Val,I,_,I,Val) :- !.
somat_kbet(LU,Valsum,I,J,K,Valtemp) :-
    nth1(I,LU,Vecal),
    nth1(K,Vecal,Alfa),
    nth1(K,LU,Vecbe),
    nth1(J,Vecbe,Beta),
    Valstemp2 is Valtemp+Beta*Alfa,
    K2 is K+1,
    somat_kbet(LU,Valsum,I,J,K2,Valstemp2).
somat_kalf(_ ,Val,_,J,J,Val) :- !.
somat_kalf(LU,Valsum,I,J,K,Valtemp) :-
    nth1(I,LU,Vecal),
    nth1(K,Vecal,Alfa),
    nth1(K,LU,Vecbe),
    nth1(J,Vecbe,Beta),
    Valstemp2 is Valtemp+Beta*Alfa,
    K2 is K+1,
    somat_kalf(LU,Valsum,I,J,K2,Valstemp2).

```

```

% PIVOTING PARCIAL (TROCA APENAS LINHAS)
% SUBROTINA QUE TROCA UMA LINHA POR OUTRA DE FORMA
% QUE O PIVÔ SEJA DIFERENTE DE ZERO
% pivoting(Matriz,I elemento com problema, J-contador começa com Len,
%     Saída).
% IMPORTANTE, SÓ REALIZAR O PIVOTING SE O NÚMERO FOR ZERO, ACHO QUE É
% INEFICIENTE EU COLOCAR ESSA CONDIÇÃO AQUI.
pivoting(_ ,I,I,_) :-
    write('não consegui realizar o pivoting.'),nl,
    write('provavelmente o determinante dos coeficientes é zero.'),nl,
    !,fail.
pivoting(_ ,0,_) :-
    write('não consegui realizar o pivoting.'),nl,
    write('provavelmente o determinante dos coeficientes é zero.'),nl,
    !,fail.

```



```

pivoting(Ment,I,J,Mentnovo):-
  J>1,
  nth1(I,Ment,Mveci),
  %O elemento que ele vai me dar deve ser útil
  %pra mim.
  nth1(J,Ment,Mvecj),
  nth1(I,Mvecj,Ilnovo),
  Ilnovo\=0,
  %agora vou trocar os vetores de lugar.
  adiciona(Ment,1,J,Mveci,Meni),
  adiciona(Meni,1,I,Mvecj,Mentnovo),
  assert(troca_ij(I,J)),
  !.
pivoting(Ment,I,J,Mentnovo):-
  J2 is J-1,
  pivoting(Ment,I,J2,Mentnovo).

```

```

% ROTINA QUE REORGANIZA O VETOR B SEGUINDO AS REGRAS DO PIVOTING

```

```

% A reorganização acontece se baseando no fato de que sempre o I,
% o primeiro termo teve sua operação realizada primeiro.
% unscramb(Vetor b de entrada, vetor b de saída, tamanho do vetor).
% depende de um fato na memória que é o troca_ij(I,J).

```

```

unscramb(Vec,Vec,0):-!.
unscramb(Vecb,Vecb3,I):-
  troca_ij(I,B),
  I2 is I-1,
  nth1(I,Vecb,Vba),
  nth1(B,Vecb,Vbb),
  adiciona(Vecb,1,B,Vba,Vecb0),
  adiciona(Vecb0,1,I,Vbb,Vecb2),
  unscramb(Vecb2,Vecb3,I2),!.
unscramb(Vecb,Vecb3,I):-
  I2 is I-1,
  unscramb(Vecb,Vecb3,I2).

```

```

% ROTINA QUE REALIZA UMA SUBSTITUIÇÃO PARA FRENTE
% SERVE PARA RESOLVER SISTEMAS LINEARES QUANDO A
% MATRIZ FOR TRIANGULAR INFERIOR.
% ESTAMOS ASSUMINDO QUE OS TERMOS DA DIAGONAL PRINCIPAL
% SÃO TODOS 1.

```

```

% for_sub(Matriz triangular, vetor b, I resolve por linha começa
% do 2 (o 1 já é o vecb), Len tamanho do vetor,
% vetor solução).

```

```

for_sub(_,Vecips,Ifim,Len,Vecips):-
  Lb is Len+1,
  Ifim==Lb,!.

```

```

for_sub(LU,Vecb,I,Len,Vecips):-
  nth1(I,Vecb,Bei),
  somat_ips(LU,Vecb,Sumips,I,1,0.0e0),
  Nips is Bei-Sumips,
  adiciona(Vecb,1,I,Nips,Vecb1),
  I2 is I+1,
  for_sub(LU,Vecb1,I2,Len,Vecips).

```

```

% somat_ips(Matriz LU junta, Vetor b mas que carrega os ipsilon
% antigos, valor de saída, I índice do ipsilon,
% J começa de 1, muleta começa de 0)

```

```

somat_ips(_,_,Val,I,I,Val):-!.
somat_ips(LU,Vecb,Valsum,I,J,Valtemp):-
  pega_ij(I,J,LU,Alfaij),
  nth1(J,Vecb,Ipsj),
  Valtemp2 is Valtemp+Ipsj*Alfaij,
  J2 is J+1,
  somat_ips(LU,Vecb,Valsum,I,J2,Valtemp2).

```

```

% ROTINA QUE REALIZA UMA SUBSTITUIÇÃO PARA TRÁS
% SERVE PARA RESOLVER SISTEMAS LINEARES QUANDO A

```

```

% MATRIZ FOR TRIANGULAR SUPERIOR.
% back_sub(Matriz triangular, vetor b, l resolve por linha começa
%   do Len, Len tamanho do vetor,
%   vetor solução).
back_sub(_, Vecxis, 0, _, Vecxis):-!.
back_sub(LU, Vecips, Len, Len, Vecxis):-
    pega_ij(Len, Len, LU, Betann),
    nth1(Len, Vecips, Ipsi),
    Xisn is Ipsi/Betann,
    adiciona(Vecips, 1, Len, Xisn, Vecips1),
    I is Len-1,
    back_sub(LU, Vecips1, I, Len, Vecxis),!.
back_sub(LU, Vecips, I, Len, Vecxis):-
    nth1(I, Vecips, Ipsi),
    J is I+1,
    somat_xis(LU, Vecips, Sumxis, Len, I, J, 0.0e0),
    pega_ij(I, I, LU, Betaii),
    Xis is (Ipsi-Sumxis)/Betaii,
    adiciona(Vecips, 1, I, Xis, Vecips1),
    I2 is I-1,
    back_sub(LU, Vecips1, I2, Len, Vecxis).

% somat_xis(LU matriz, vetor ipylon, soma saida, tamanho do vetor Len,
%   I corre do lado de fora, J começa de I+1, 0.0e0 muleta).
somat_xis(_, Sumxis, Len, _, J, Sumxis):-
    Lb is Len+1,
    J==Lb,!.
somat_xis(LU, Vecips, Sumxis, Len, I, J, Sumtemp):-
    pega_ij(I, J, LU, Betaij),
    nth1(J, Vecips, Xj),
    Sumtemp2 is Sumtemp+Xj*Betaij,
    J2 is J+1,
    somat_xis(LU, Vecips, Sumxis, Len, I, J2, Sumtemp2).

% PRODUTO INTERNO ENTRE 2 LISTAS
% prod_int(Lista 1, Lista 2, 0 - soma temporaria, saida).
% a muleta e pra ajdar a tail recursividade.
% A muleta tem que começar do 0
% as listas não forem colocadas direito dá erro.
prod_int(V1, V2, Prod):-
    prod_int(V1, V2, 0, Prod).

prod_int([], [], Prod, Prod):-!.
prod_int([X1|L1res], [Y1|L2res], Prod, Prod):-
    Prodt2 is Prodt+X1*Y1,!,
    prod_int(L1res, L2res, Prodt2, Prod).
prod_int(_, _, _, _):-
    write('Erro na rotina prod_int'),nl,
    write('Possivelmente os vetores não foram definidos corretamente'),
    nl,!,fail.

% ROTINA QUE CALCULA O PRODUTO VETORIAL
% Sempre 3x3 very easy
prod_vec(V1, V2, Vf):-
    V1=[X1, Y1, Z1],
    V2=[X2, Y2, Z2],
    Vf=[I, J, K],
    I is Y1*Z2-Y2*Z1,
    J is Z1*X2-X1*Z2,
    K is X1*Y2-Y1*X2.

% ROTINA QUE REALIZA UMA SOMA BÁSICA DE VETORES
% Soma_vec(vec 1, vec 2, resultado)
soma_vec([], [], []):-!.

```

```

soma_vec([X|Vec1],[Y|Vec2],[Z|Vecsai):-
    Z is X+Y,!,
    soma_vec(Vec1,Vec2,Vecsai).
soma_vec(_,_):-
    write('Erro na rotina soma_vec'),nl,
    write('Possivelmente os vetores não estão bem definidos'),nl,
    !,fail.

% ROTINA QUE REALIZA UMA SUBTRAÇÃO BÁSICA DE VETORES
% subtrai_vec(vec 1, vec 2 vai ser trocado o sinal
%      ,iteração--tamanho, resultado)
subtrai_vec([],[],[]):-!.
subtrai_vec([X|Vec1],[Y|Vec2],[Z|Vecsai):-
    Z is X-Y,!,
    subtrai_vec(Vec1,Vec2,Vecsai).
subtrai_vec(_,_):-
    write('Erro na rotina subtrai_vec'),nl,
    write('Possivelmente os vetores não estão bem definidos'),nl,
    !,fail.

% ROTINA QUE MULTIPLICA UM VETOR POR UMA CONSTANTE
% cons_vec(Vetor, contante, tamanho, Saida)
cons_vec([],_):-!.
cons_vec([X|Vconin],Alfa,[Y|Vcont):-
    Y is Alfa*X,
    cons_vec(Vconin,Alfa,Vcont).

% ROTINA QUE SOMA DUAS MATRIZES
% soma_matr(+Matriz 1,+Matriz 2, +LenI quantidade de linhas,
%      +LenJ quantidade de colunas, -Matriz de saída)
soma_matr([],[],[]):-!.
soma_matr([X|Vec1],[Y|Vec2],[Z|Vecsai):-
    soma_vec(X,Y,Z),!,
    soma_matr(Vec1,Vec2,Vecsai).
soma_matr(_,_):-
    write('Erro na rotina soma_matr'),nl,
    write('Possivelmente as definições de matrizes estão incorretas'),nl,
    !,fail.

% ROTINA QUE SUBTRAI A MATRIZ 1 DA 2
% subtrai_matr(+Matriz 1,+Matriz 2, +LenI quantidade de linhas,
%      +LenJ quantidade de colunas, -Matriz de saída)
subtrai_matr([],[],[]):-!.
subtrai_matr([X|Vec1],[Y|Vec2],[Z|Vecsai):-
    subtrai_vec(X,Y,Z),!,
    subtrai_matr(Vec1,Vec2,Vecsai).
subtrai_matr(_,_):-
    write('Erro na rotina subtrai_matr'),nl,
    write('Possivelmente as definições de matrizes estão incorretas'),nl,
    !,fail.

% ROTINA QUE MULTIPLICA UMA MATRIZ POR UMA CONSTANTE
% cons_matr(+Matr,+Cons,+1 I+LenI,+LenJ,-MatrSai)
cons_matr([],_):-!.
cons_matr([Mentli|Mentcont],Cons,[LinInova|Matrsai):-
    cons_vec(Mentli,Cons,LinInova),
    cons_matr(Mentcont,Cons,Matrsai).

% ROTINA QUE REALIZA UMA MULTIPLICAÇÃO UMA MATRIZ POR UM VETOR
% matr_vec(matriz,vetor de entrada,vetor de saída)
matr_vec([],_):-!.
matr_vec([Mli|Mred],Vec,[Mvi|Mvcont):-
    prod_int(Mli,Vec,0,Mvi),!,
    matr_vec(Mred,Vec,Mvcont).
matr_vec(_,_):-
    write('Erro na rotina matr_vec'),nl,
    write('Possivelmente a definição da matriz ou vetor está errada'),
    nl,fail.

```

```

%ROTINA QUE TRANSPOE A MATRIZ NXM
%observação: esta sendo usado a rotina maplist de forma
%inteligente para se extrair as colunas.
%transpoe(+Ment matriz de entrada,-Msai matriz de saída).
transpoe(Ment,Msai):-
    nth1(1,Ment,Mlin),
    length(Mlin,LenJ),!,
    transpoe(1,LenJ,Ment,Msai).

%transpoe(+1 - contador, +LenJ quantidade de colunas,
%      +Ment,-Msai).
transpoe(Ment,Msai):-
    nth1(1,Ment,X1),
    length(X1,Col),
    transpoe(1,Col,Ment,Msai).

transpoe(J,LenJ,_,[]):-
    J>LenJ,!.
transpoe(J,LenJ,Ment,[ColJ|Mcont]):-
    maplist(nth1(J),Ment,ColJ),
    J2 is J+1,
    transpoe(J2,LenJ,Ment,Mcont).

% ROTINA QUE MULTIPLICA MATRIZES NXM
% matr_matr(+Matriz de entrada 1,+Matriz de entrada 2,
%      +LenI linhas da matr 1, +LenJ colunas da matr 2,
%      +I contador =1,-Matriz de saída IxJ)
matr_matr(M1,M2,M1M2):-
    transpoe(M2,M2t),
    matr_matr_loop(M1,M2t,M1M2t),
    transpoe(M1M2t,M1M2).

matr_matr_loop(_,[],[]):-!.
matr_matr_loop(M1,[M2colj|M2cont],[M1M2colj|M1M2cont]):-
    matr_vec(M1,M2colj,M1M2colj),
    matr_matr_loop(M1,M2cont,M1M2cont).

% PEGA O TERMO LINHA I COLUNA J DE UMA MATRIZ
% pega_ij(linha I, coluna J, matriz de entrada, matr saída)
pega_ij(I,J,Ment,Tsai):-
    nth1(I,Ment,M1),
    nth1(J,M1,Tsai).

% ADICIONA O TERMO I A POSIÇÃO I DA LISTA.
% adiciona(Lista in, Imuleta=1,posição i a ser colocado,
%      Termo, Lista saída)
% se for maior coloca zero no caminho.
% !!! Se a linha for vazia vai dar um pau
%
adiciona([_|Lires],1,1,Term,[Term|Lires]):-!.
adiciona([X|Lires],I,Iterm,Term,[X|Lout]):-
    I2 is I+1,
    Iterm==I2,
    Lires=[_|B],
    Lout=[Term|B],
    !.
adiciona([X|Lires],I,Iterm,Term,[X|Lout]):-
    I2 is I+1,
    adiciona(Lires,I2,Iterm,Term,Lout).

% ADICIONA O TERMO TERM A POSIÇÃO IJ DE UMA MATRIZ
% adiciona_ij(linha I, coluna J, termo a ser adicionado,
%      matriz de entrada, matriz de saída).
adiciona_ij(I,J,Term,Ment,Msai):-
    nth1(I,Ment,LinI),
    adiciona(LinI,1,J,Term,Linsai),
    adiciona(Ment,1,I,Linsai,Msai).

```

```
% ROTINA QUE TRANSFORMA UMA COLUNA DE UMA MATRIZ DADA EM UM VETOR
```

```
%pega_col(matriz de entrada,coluna J saida,linha i comça de 1
```

```
%      ,J que eu quero,quantidade de linhas)
```

```
pega_col(Ment,[TerJ],Len,J,Len):-
```

```
  pega_ij(Len,J,Ment,TerJ),!
```

```
pega_col(Ment,[X|Colcont],I,J,Len):-
```

```
  pega_ij(I,J,Ment,X),
```

```
  I2 is I+1,
```

```
  pega_col(Ment,Colcont,I2,J,Len).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% UTILIDADE GERAL %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ROTINA QUE GERA UM VETOR DE ZEROS
```

```
% ROTINA QUE GERA UM VETOR DE ZEROS
```

```
% gera_val_ini(tamanho,vetor de saida).
```

```
gera_vec_zero(1,[0]):- !.
```

```
gera_vec_zero(Len,[0|Lres]):-
```

```
  Len2 is Len-1,
```

```
  gera_vec_zero(Len2,Lres).
```

```
% ROTINA QUE GERA A MATRIZ IDENTIDADE
```

```
% gera_ident(+1 contador,+Len tamanho desejado,-Iden matriz).
```

```
gera_ident(I,Len,[]):-
```

```
  I>Len,!
```

```
gera_ident(I,Len,[Li|Idencont]):-
```

```
  gera_li(I,1,Len,Li),
```

```
  I2 is I+1,
```

```
  gera_ident(I2,Len,Idencont).
```

```
% gera_li(+I posicao i corrente,+1 contador,+Len tamanho
```

```
%      do vetor,-Li lista com 1 na posicao I e 0 no resto).
```

```
gera_li(_,J,Len,[]):-
```

```
  J>Len,!
```

```
gera_li(I,J,Len,[Xli|Licont]):-
```

```
  kronecker(I,J,Xli),
```

```
  J2 is J+1,
```

```
  gera_li(I,J2,Len,Licont).
```

```
kronecker(X,X,1):- !.
```

```
kronecker(_,_,0).
```

Anexo C: Participação em artigo

A seguir está apresentado o artigo o qual participei e que foi submetido recentemente.

Genetic Algorithms Coupled with Quantum Mechanics for Refinement of Force Field applied to Nucleic Acids.

Rodrigo B. Kato,[†] Frederico T. Silva,[‡] Gisele L. Pappa,[†] and Jadson C. Belchior^{*,‡}

Department of Computer Science, Universidade Federal de Minas Gerais, Belo Horizonte, and Department of Chemistry, Universidade Federal de Minas Gerais, Belo Horizonte

E-mail: jadson@ufmg.br

Abstract

We reported the use of Genetic Algorithms (GA) as a method to refine force field parameters in order to determine nucleic acids energies (adenine, guanine, cytosine and uracil). Quantum calculations are carried out for these nucleic acids that are taken as reference data. In this particular study of torsion and electrostatic energies are reparametrized in order to test the proposed approach, i.e, GA coupled with quantum mechanics calculations. Overall, RMSE comparison with recent published results for describing nucleic acids energies showed an improvement, on average, of 50%. Finally, the new reparametrized potential energy was used to determine energy and structure of a PDB molecule (1r4h) that was not used to parametrization process. The structure was improved about 82% compared with previous published results.

*To whom correspondence should be addressed

[†]Department of Computer Science, Universidade Federal de Minas Gerais, Belo Horizonte

[‡]Department of Chemistry, Universidade Federal de Minas Gerais, Belo Horizonte

Introduction

It is well known that molecular dynamics techniques can provide accurate data for analyzing properties of a specified system.¹ However, this analysis is strongly dependent on the level of precision of the force field describing the system. The precision of the force field is inverse proportional to the size of the system as well as the amount of electrons taken into account.² Usually one needs to also consider correlation effects, which increase the demand for computational time.

The most effective way of obtaining the energy of a system is based on quantum calculations, and several methods with different levels of precision are available.^{3,4} Accordingly, almost all of them are highly computational time consuming, especially for larger systems. This computational time demand can be reduced by using empirical potentials (force fields). In this case, the procedure is generally based on the optimization of mathematical functions with several terms that contribute to the total energy, including physical contributions, such as attractive and repulsive terms (intermolecular forces), and others that determine the internal energy contribution (intramolecular forces). The development of empirical models to describe the system is an efficient alternative to provide a way to determine energies and structures by proposing force fields.

Potential energy calculations are common in the literature and, in general, researches try to improve the force field by reparametrization of all or part of the terms that contribute to the total energy. For example, CHARMM (all19, all22, all27) and Amber (ff94, ff96, ff98, ff99) force fields are commonly reparametrized.^{3,5-9} However, due to the approximation of pairwise additive empirical potentials taken into account to build up force fields, such a task usually decreases the precision of the results. Zgarbová *et. al.*¹⁰ stated that this imprecision is beyond the capabilities of simple force field approximations. Hence, they concluded that it is not surprising that different force fields often provide remarkably different descriptions of the same structure, a phenomenon known as “force field dependent polymorphism”.^{4,11-14} Among other works,^{3,5-9} Zgarbová *et. al.*¹⁰ recently reported a reparameterization of the

glycosidic torsion function of the Cornell *et. al.* Amber force field applied to RNA molecules. Although their work improved the description of the syn region and the syn-anti balance as well as enhance molecular dynamics (MD) simulations of various RNA structures, the results showed deviations from quantum calculations. This is quite common on the use of empirical force fields. Therefore, it seems that studies to improve parametrization are important in order to better compare semiclassical results against experimental or quantum calculations. Particularly, Banás *et. al.*¹⁵ recently showed tests using a reparametrized force field, and deviations for A-DNA and B-DNA structures using molecular dynamic were observed.

Techniques usually applied to optimize force fields are based on gradient methods such as quasi-Newton approach¹⁶ and Levenberg-Maquardt method.^{16,17} For example, Yildirim *et. al.*¹⁸ used steep descent method and all of the techniques aforementioned showed to be accurate in comparison with quantum results. More recently other heuristic approaches have been proposed and tested to refine parameters of force fields. For example, Sakae and Okamoto¹⁹ applied Monte Carlo and simulated annealing to calculate the parameters of force field.

Alternative approaches for optimization processes have been recently proposed for parametrizing force fields.²⁰ Similarly, we have recently applied genetic algorithms (GA) coupled with quantum calculations to determine structures and energies, however for small systems with particular attention to clusters alloy compounds.^{21,22} The use of GA method applied to atomic and molecular structures and energies optimizations is quite common.^{23,24} However, the application of GA to improve force fields for nucleic acids is rare in the literature and, therefore, the present work is dealing with its use in order to reparametrize force fields. In this way, we test the procedure considering, as a case study, only the torsion potential.

Aiming to improve the precision of force field glycosidic torsion profile, this work proposes a heuristic solution based on genetic algorithms²⁵⁻²⁷ to better estimate the reparametrization of force field to torsion energies based on the Darwin's principles of evolution and survival of the fittest. The approach works with a population of individuals, where each individual

represents a solution to the problem at hand. These individuals are evaluated according to a fitness function, which measures how well they solve the target problem. Based on the fitness values, individuals are selected to undergo crossover and mutation operations, producing a new population of individuals. On the lack of high level experiments on the system of nucleic acids (adenine, guanine, cytosine and uracil) quantum calculations are used in order to provide accurate data as a reference to test the present approach. This process is iterative, and goes on until a stopping criteria is met, as detailed later in this paper. A comparison of our refined results with previous publication as well as with PDB databank is finally analyzed.

Methods

Model System

The methodology used to optimize the dihedral term of the glycosidic torsion parameter of the force field in order to describe RNA nucleotides is showed in Figure 1. The procedure is carried out based on a set of quantum calculations to obtain a reference data (energy and structure), which are used as a template for the parametrization of the force field using other methods,² in this case, a genetic algorithm.

The inputs of quantum calculations are the isolated structures of each RNA base (Figure 2). These structures are optimized using quantum calculations, performed by the software GAMESS.²⁸ The outputs are: an optimized structure for the input bases, a set of charges for the involved atoms and the quantum energy (E^{QM}), which are used in next step of the proposed approach.

Obtaining the Torsion Profiles

The torsion profiles are obtained using the optimized structures calculated by GAMESS. We also map a subset of possible dihedral angles of the molecule by making 50 degrees rotations

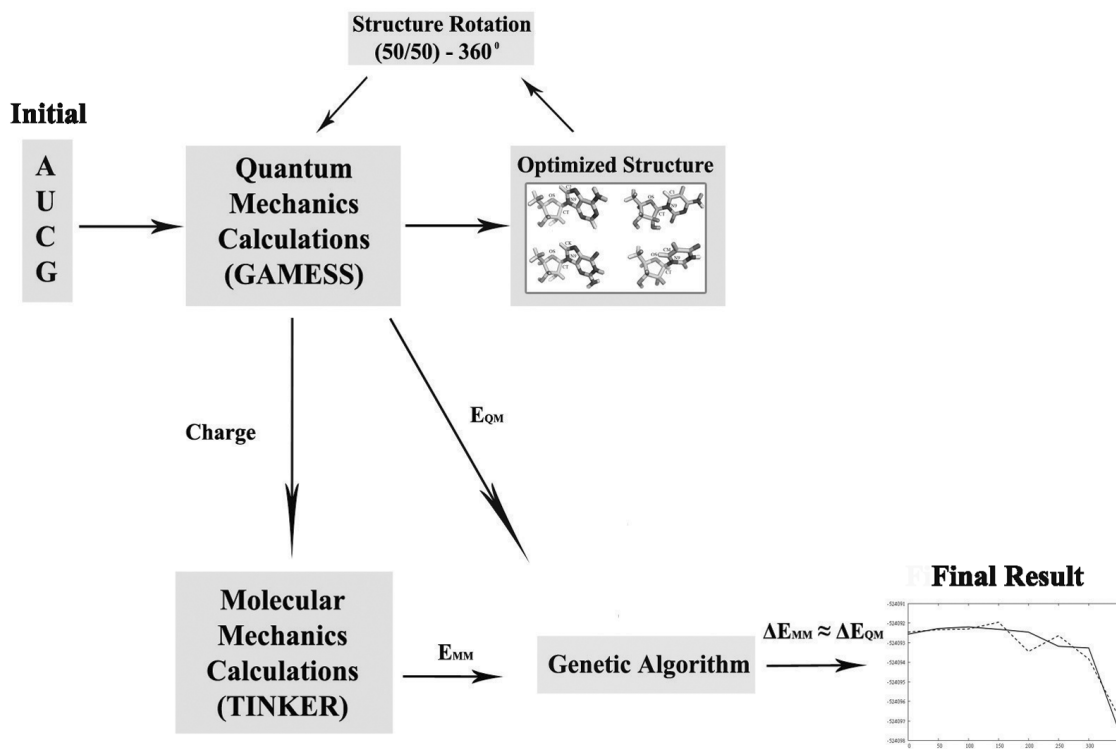


Figure 1: Scheme of the proposed methodology, where A is adenine, U is uracil, C is cytosine, G is guanine, E^{MM} is molecular mechanics energy, E^{QM} is quantum energy and ΔE^{MM} is the difference between each rotation $\sum_{i=1}^n \frac{E_{i-1}^{MM} - E_i^{MM}}{n}$.

in the glycosidic bonds in the range of 0 to 360 degrees. Other works in the literature use 10 degrees step when performing the same process.¹⁰ Due the time spent to perform this quantum calculations, here we increased the steps to 50. For each of the 7 new structures generated with the rotations (360 degrees in 50 degrees steps), we perform again a set of quantum calculations to recalculate them.

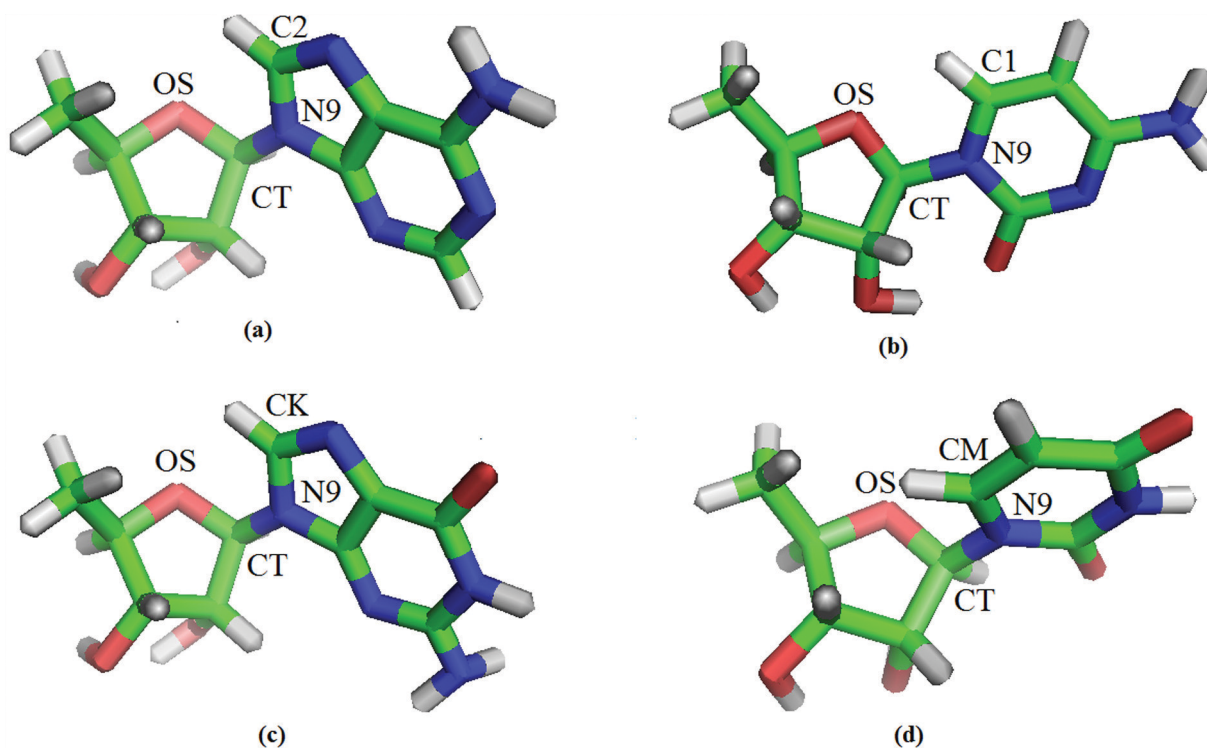


Figure 2: Nucleotides used as input for the method (a) Adenine, (b) Cytosine, (c) Guanine e (d) Uracil. The dihedral angle is defined by the OS-CT-N9-C2 atoms for adenine, by the OS-CT-N9-C1 atoms for cytosine, by the OS-CT-N9-CK atoms for guanine and by the OS-CT-N9-C1 atoms for uracil (this definition is used throughout this work).

The GA receives as reference data seven optimized structures rotated and one without rotation for each nucleotide. The algorithm evolves the parameters for four different periods of the cosine function that describes its dihedral energy. The refinement of parameters are evaluated according to Eq. 1 using the same geometry for both the molecular mechanics and the quantum calculations.

$$RMSE = \sqrt{\sum_{i=1}^n \frac{[(E_{i-1}^{QM} - E_i^{QM}) - (E_{i-1}^{MM} - E_i^{MM})]^2}{n}} \quad (1)$$

The force field energy (E^{QM}) is calculated using Amber force field (ff99), which uses the parameters returned by the GA. Different rotation energies (0 to 360 degrees with 50 degrees steps) are compared with quantum energies (E^{QM}) for each nucleotide (adenine, guanine, cytosine and uracil) calculated from GAMESS.

Quantum Calculations

The choice of quantum methods was based on works published in the literature. Few parametrization based on Amber force field (ff94,⁵ ff98²⁹ and ff99⁷) have considered the Hatree-Fock method. However, in this method the electrons interact without considering electron correlation. This fact generates great consequences in the interpretation of wave functions.³⁰ Hence, recent studies focused on more effective methods. For example, the last improvement over the CHARMM force field optimizes geometry based on MP2/6-31+G* and RI-MP2/cc-pVTZ calculations using Density Functional Theory (DFT). In turn, the calculations are used to improve the torsion potential of α/γ for the Amber force field.^{10,31,32}

In this work the same level of theory (DFT) is used, i.e, the PBE functional and the basis 6-31++G (3df, 3pd) as recently also used in reference [10]. The latter work showed excellent results compared against quantum data.

Molecular Mechanics Calculations

As previously mentioned, the parameters evolved by the GA are tested with the Amber force field, defined in Eq. 2.¹⁰ It is represented by the sum of several energy contributions, namely, the bond stretching (E_{bond}), angle bending (E_{angle}), dihedral (E_{dih}), nonbonded electrostatic (E_{elst}) and van der Waals terms (E_{vdW}). The GA focused on the optimization of the dihedral angle (see Eq. 3), and is defined as a cosine series, where n is the periodicity of the torsion,

V_n is the rotational barrier, φ is the torsion angle, and γ is the phase angle.

$$E_T = E_{bond} + E_{angle} + E_{dih} + E_{elst} + E_{vdW} \quad (2)$$

$$E_{dih} = \frac{V_n}{2}[1 + \cos(n\varphi - \gamma)] \quad (3)$$

The quantum calculations returned the charges of the atoms, and we also used this information to reparametrize the electrostatic term E_{elst} , described in Eq. 4, where q_1 and q_2 are the charges between two atoms, r_{12} is the distance between them and ϵ is the dielectric constant. The contribution of the electrostatic interaction is one the most important in the dynamical stability of the molecule.³³ We used the charges average over 8 different nucleotide conformations for each atom to refine the parametrization of this term. The charge that will be used as parameter in the force field is written as:

$$E_{elst} = \frac{q_1 q_2}{\epsilon r_{12}} \quad (4)$$

The molecular mechanics calculations were performed by the software Tinker.³⁴ This software is widely used in the literature, and examples of studies based on its calculations are described elsewhere.³⁵⁻³⁷

Optimizing the Dihedral Term

Genetic algorithms are methods based on Dawin’s principles of evolution and survival of the fittest. They are well-known for performing a global search while analyzing multiple solutions of the search space at each iteration. These algorithms work with individuals, where each individual represents a solution to the problem to be solved. In our case, an individual is a vector of 8 positions, as illustrated in Figure 3, representing the terms V_n and γ , previously defined in Eq. 3, for four different periods of the cosine function.

The standard GA search procedure is showed in Figure 4. First, an initial population

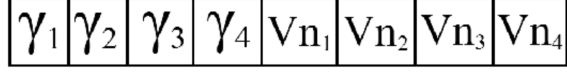


Figure 3: Representation of individual in the genetic algorithm.

of individuals is randomly generated, following a normal distribution with average 0 and standard deviation defined as 1. Next, individuals are evaluated according to how well they solve the problem at hand. As our individuals are torsion parameters, they are evaluated according to how far the molecular mechanics energy E^{MM} (obtained by Tinker with the GA optimized parameters) approaches to the quantum energy E^{QM} (obtained by GAMESS). Eq. 5 describes the fitness function, which is the RMSE (Root Mean Square Error) of the difference of energies for each conformation i (which here n is equal to 8). The lower the RMSE, the better the force field approximation is performed.

$$Fitness = RMSE = \sqrt{\sum_{i=1}^n \frac{[(E_{i-1}^{QM} - E_i^{QM}) - (E_{i-1}^{MM} - E_i^{MM})]^2}{n}} \quad (5)$$

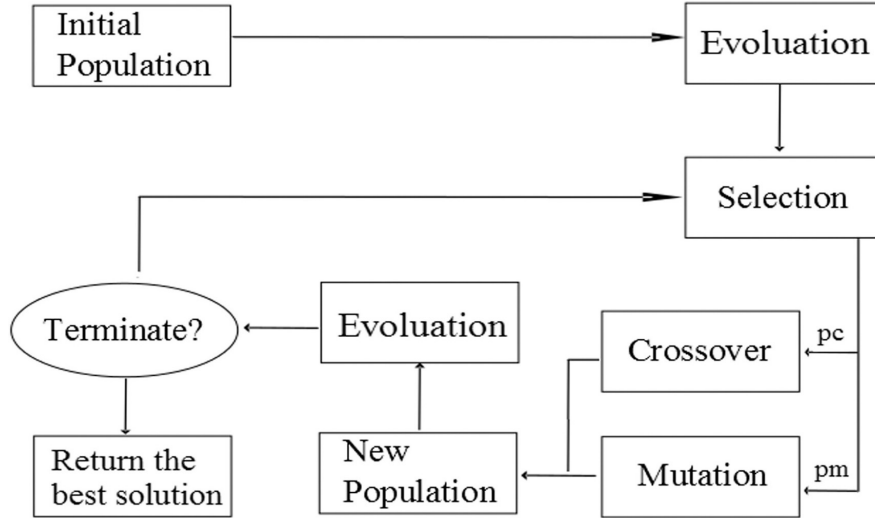


Figure 4: Flowchart of a standard genetic algorithm, where p_c is crossover probability and p_m is mutation probability.

Following the GA structure shown in Figure 4, after evaluation, individuals are selected

to undergo crossover and mutation operations according to the probabilities namely, p_c (crossover probability) and p_m (mutation probability). The tournament selection method is used in the present GA.²⁶ In this procedure, k individuals are randomly selected from the population, and compete with each other for a chance to undergo genetic operations. The individual with the best fitness (in our case, lowest RMSE), is selected. Note that, in this case, individuals with higher fitness have greater probability of surviving, and this probability varies according to the size of tournament (k) selected.

Individuals selected through consecutive tournaments can then be modified using crossover or mutation before being inserted into the new population. Crossover takes two individuals, and its main objective is to exchange material between them, creating two children. Here we used uniform crossover, that generates a mask to determine the positions of the two individuals that will be exchanged, as showed in Figure 5.

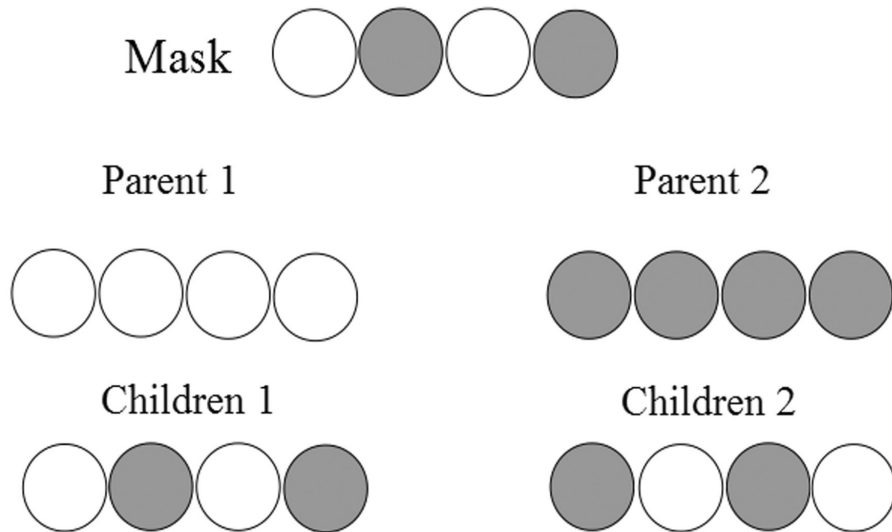


Figure 5: Schematic of uniform crossover operation.

Mutation, in turn, is performed over a single individual, and its objective is to cause a small random modification into the individual, in order to leave a local stagnation in the search region (local optimum). The mutation used here changes the parameter at a randomly selected position by $+(-) 0.02$.

All individuals generated by the above operators are inserted into a new population, together with the best of all individuals, which is preserved from one generation to the next (procedure known as elitism). This process goes on until a stopping criteria is met, which is usually a minimum error or maximum number of generations.

Note that the algorithm uses a set of parameters, including number of individuals, number of generations, crossover and mutation probabilities and tournament size. An appropriate choice of these parameters is crucial for a good coverage of the search space. Number of individuals and generations are parameters highly dependent on the size of the search space, determined by the problem understudied. Here we tested population sizes of 50, 100, 150 and 200, and generations of 200, 500 and 800. Many combinations were tested, and the best results obtained are with 200 individuals and 800 generations.

Crossover and mutation probabilities have more standard values, with high crossover and low mutation rates. We tested crossover probabilities of 0.9 and 0.95, and mutation rates of 0.01, 0.1, 0.2, 0.25 e 0.3. All possible combinations were tested, and the best results obtained are 0.95 and 0.3 for crossover and mutation, respectively. For tournament size, values of 8, 10, 20 and 30 were tested, with 8 showed the best results. All parameter tested were run five times, and the best results selected.

Results and Discussion

The Optimization.

The dihedral energy parameter optimization is done by a genetic algorithm, which uses as reference the nucleotide structure provided by quantum calculations. The GA evolves the parameters for four different periods of the cosine function that describes the dihedral energy, optimizing the rotational barrier V_n and the phase angle γ (see Eq. 3 for further details). The parameters being refined are evaluated according to the value of the differences of energy variation returned by quantum calculation and molecular mechanics calculation. The Amber

force field (ff99) uses the parameters obtained by the GA, and the energy comparisons involve eight different rotations (0 to 360 degrees with 50 degrees steps) for each nucleotide (adenine, guanine, cytosine and uracil), in order to map a subset of possible torsions of the glycosidic bond (see Eq. 5).

The values of the parameters found by the GA, followed by those reported in reference [10], are showed in Table 1. As GA is a heuristics method, it was executed 5 times in order to show the algorithm did not find an appropriate solution by chance. As the variance of the results is small, we report the parameters obtained with the execution that produced the smallest RMSE between the differences of quantum energy (E^{QM}) and molecular mechanics energy (E^{MM}) calculated using different nucleotide rotations (Eq. 5). The computational time of each GA generation is, on average, 60 seconds. Each generation evaluated a set of 200 different parameters, and 800 generations are executed. Hence, a complete experiment runs for about 13 hours.

The evolution of the individuals along the generations in terms of RMSE, that is, the differences of E^{QM} and E^{MM} in different nucleotides rotations, is showed in Figure 6. From now on, we refer to this measure simply as RMSE. The curves shown in these plots represent the best individual and the evolution of the RMSE improved along the generations. Note that, for guanine, the curve stabilizes fast, while for cytosine the improvements happen until the end of the search process. The average RMSE is an indicative of when the search converges, as at this point all individuals (which represent the set of parameters being optimized) will have very similar values of fitness. In the present analysis convergence starts around 300 generations.

Comparison with Literature.

The RMSE reported in reference [10] and the results calculated by the GA approach used in the present work are compared in Figure 7. We were able to improve the RMSE and hence, reduce the relative global energy in 37% for adenine, 67% for guanine, 21% for cytosine and

Table 1: Parameters found by the Genetic Algorithm for the dihedral energy

Nucleotide	Torsion	n^a	GA		Reference [10]	
			$Vn/2^b$	γ^c	$Vn/2^b$	γ^c
Adenine	OS-CT-N9-C2*	1	0.9660	69.00	0.6956	68.79
		2	0.9465	15.89	1.0740	15.64
		3	0.0597	24.95	0.4575	171.68
		4	4.0000	308.85	0.3092	19.09
Guanine	OS-CT-N9-CK*	1	0.5327	50.60	0.7051	74.76
		2	0.0002	0.80	1.0655	6.23
		3	0.0001	1.96	0.4427	168.65
		4	0.3919	3.91	0.2560	3.97
Cytosine	OS-CT-N9-C1*	1	0.0092	0.87	1.2251	146.99
		2	2.9787	277.70	1.6346	16.48
		3	3.5213	278.05	0.9375	185.88
		4	0.1253	10.03	0.3103	32.16
Uracil	OS-CT-N9-CM*	1	1.9634	186.52	1.0251	149.88
		2	1.4182	137.21	1.7488	16.76
		3	1.0670	69.31	0.5815	179.35
		4	1.4675	96.34	0.3515	16.00

^a The periodicity of the torsion.

^b Magnitude of torsion in kcal/mol.

^c Phase offset in deg.

* see Figure 2 for details

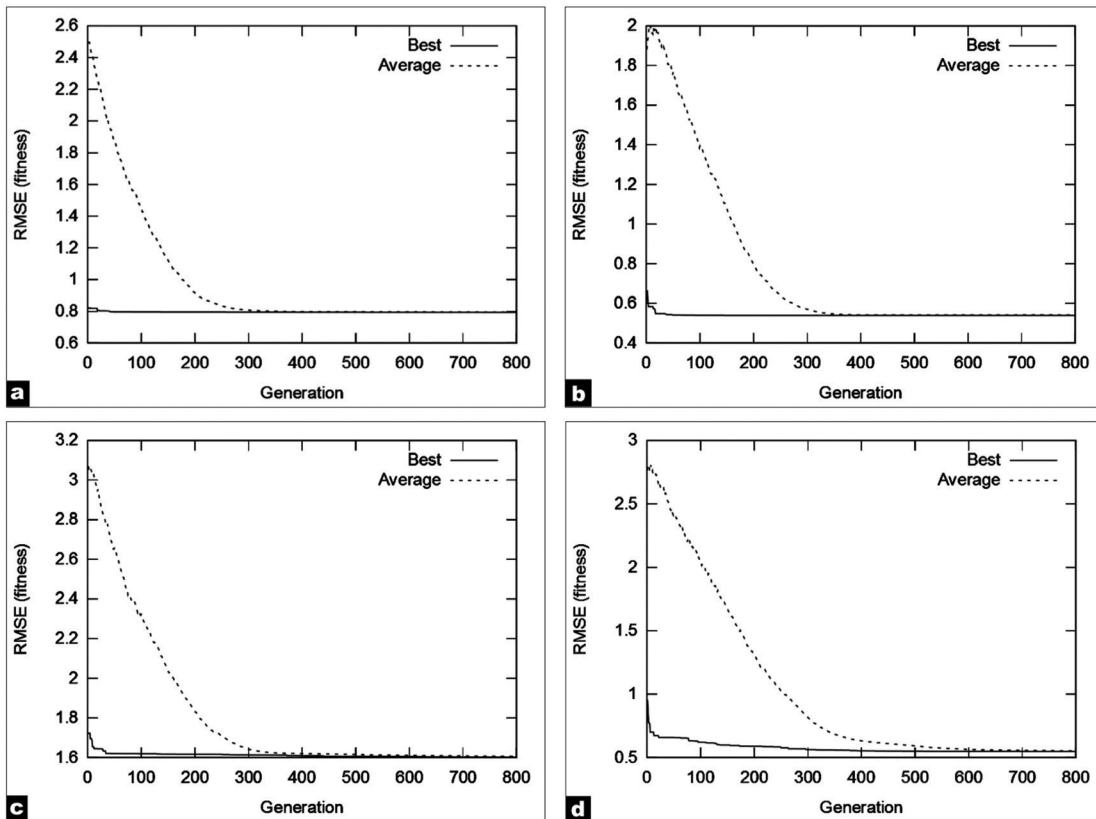


Figure 6: RMSE of the differences of E^{QM} and E^{MM} with different nucleotides rotations of the individuals evolved by the GA, considering the best individual and the population average: (a) Adenine; (b) Guanine; (c) Cytosine and (d) Uracil.

72% for uracil when compared to the results obtained in reference [10]. These numbers show that the method of parametrization performed by GA can be more effective, specially for uracil.

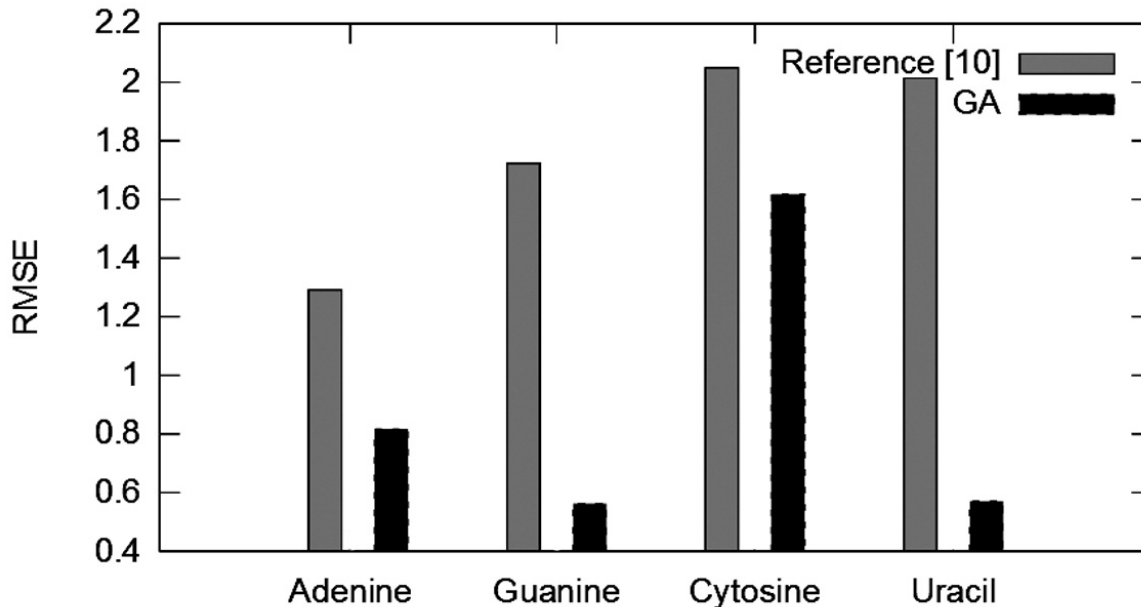


Figure 7: RMSE reported in reference[10] vs. RMSE found by the GA.

Figure 8 shows that the torsion potential energy that represents the classical energy for all nucleotides is closer to the quantum results. The actual study shows an improvement in the energy calculation based on GA optimization. A re-scale procedure, previous used³⁸ for this type of analysis was adopted in order to compare literature and present calculations against quantum results taken as reference data.

As already discussed, the best results obtained by the actual methodology are for uracil. Observing the curves in Figure 8 (d), the shape of the E^{MM} function (Eq. 2) is closer to the E^{QM} . The parametrization shown in reference [10] has demonstrated a high level of precision. However, our new approach demonstrates an improvement of such parametrization. The consequence of small deviations usually observed in force field is pointed out by Banás *et al.*¹⁵ as a justification for the underestimation of base pairing energies. As showed in this figure, the proposed methodology reduced this problem, making the RNA structure closer

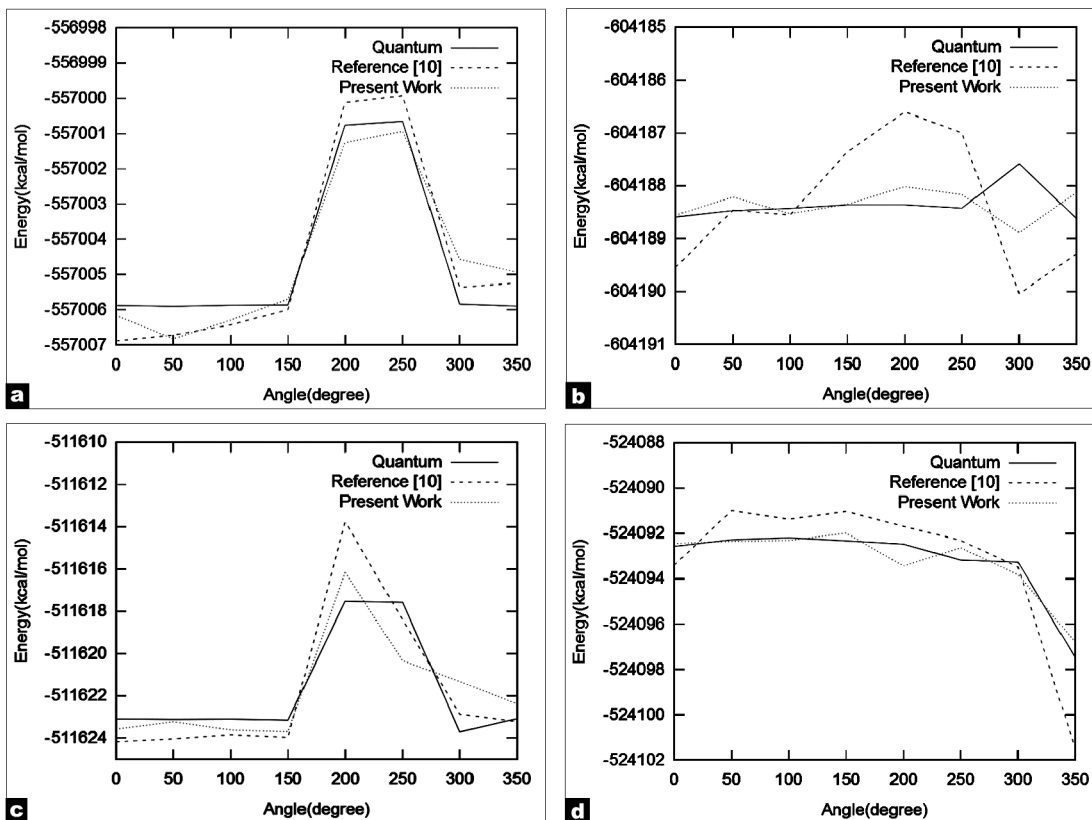


Figure 8: Comparison of E^{QC} with the E^{MM} previously reported in reference [10] and the E^{MM} obtained by the proposed methodology. Energies for (a) Adenine, (b) Guanine, (c) Cytosine, (d) Uracil.

to the data assumed as reference (quantum calculations).

RNA Crystal Simulation using New Parameters.

In order to better validate the procedure proposed in this work considering the refinement process of force field parameters, we selected the RNA 1r4h from PDB (Protein Data Bank)³⁹ to show that the parameters can be applied to optimize the representation of RNA structure. RNA 1r4h was obtained using nuclear magnetic resonance and contains ten nucleotides. It was used in the structural analysis of the IIIc domain of GB human virus. Due to its relatively small size, this structure provides a reasonable computer simulation with no difficulties for testing. In addition, if our proposed method of using GA to refine the torsion potential parameters is effective (for instance, a different molecule that was not consider in the refine process), then we believe that the application of this method to other terms (Eq. 2) can even better improve the force field, at least for nucleic acids.

We compare the structure obtained using the parameters reported in reference [10] and the results calculated here with experimental data. Tinker³⁴ was used to perform the molecular mechanics calculations for both sets of parameters. In order to simulate similar conditions to those applied in the experiment 1r4h, a water box was used to solvate the molecule (dimensions 41 x 41 x 41 Å with 333 molecules of water for one 1r4h). The simulation was performed using the software DICE,⁴⁰ and the total charges was neutralized with Na++ ions.⁴¹

A fundamental point to perform the comparisons above described is to select and calculate an appropriate measure of similarity between different structures.⁴² Here we choose RMSD (Root Mean Square Deviation), a measure widely used in the literature to represent the root mean square deviation of the atoms. RMSD is defined in Eq. 6, where x , y and z are the experimental data coordinates, x' , y' and z' are the coordinates found by Tinker and n defines the number of atoms in the molecule. As pointed out in reference [43], RSMD is a good indicator of the precision of the atomic coordinates. The lower its value, the better is

the precision. It can be written as:

$$RMSD = \sqrt{\sum_n^{i=1} \frac{(x_i - x')^2 + (y_i - y')^2 + (z_i - z')^2}{n}} \quad (6)$$

The results of RMSD considering the structure generated with GA parameters and reference [10] are compared in Table 2. One observes that the RMSD is reduced in one order of magnitude with respect to previous publication.¹⁰ Although, Zgarbová *et. al.*¹⁰ have a good value of RMSD, we were able to improve this value of in 78%. Figure 9 shows the optimized molecule structure produced by the proposed methodology (red) and the structure obtained with experimental data (green). The structures were aligned using the software Pymol.⁴⁴ In order to make visualization easier, the water molecules making solvation were removed. The results show the improvement of structure using new parametrization when compared against experimental data.

Table 2: Root Mean Square Deviation from experimental data

	RMSD
Reference [10]	1.5
GA	0.33

A Final Refinement

As observed in Figure 9, in 3D view the nucleotides in the extremity are those that present the greatest errors, specially the oxygen of the sugar for the guanine nucleotide (marked with a square in Figure 9). A better view of the guanine in the left-hand side of this figure is presented in isolation in Figure 10.

Aiming to further analyze the nature of error in the oxygen conformation, Figure 11 shows in dotted lines the hydrogen bonds for 1r4h. One of the reasons for the error is the missing hydrogen bonds, marked with squares, which is the interaction that holds the oxygen in its correct position. This problem occurs because the dihedral force constant (V_n) is too

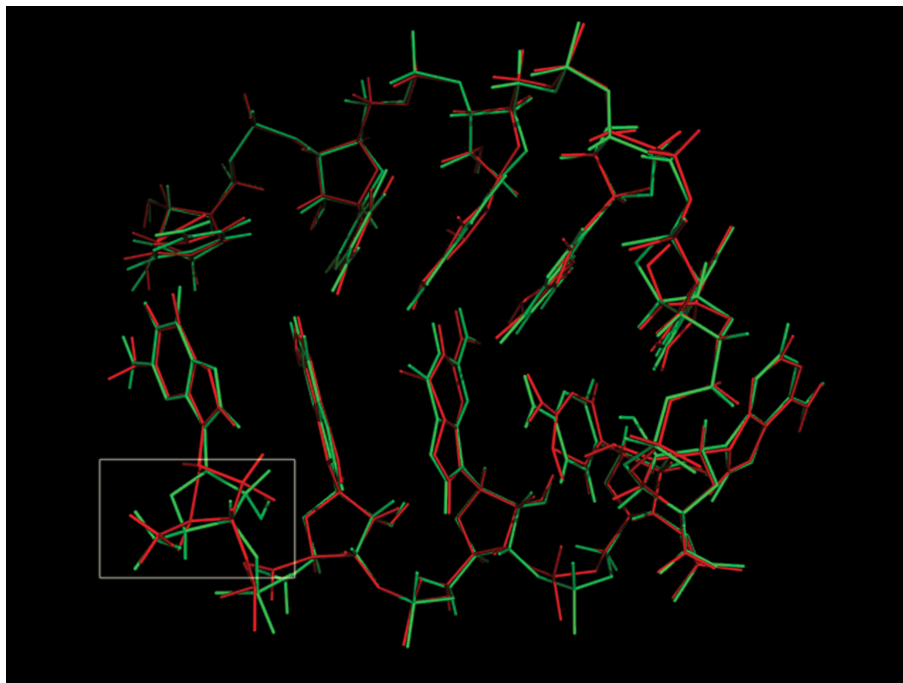


Figure 9: Comparison of the RNA 1r4h structures found by the proposed methodology (red) and experimental data (green).

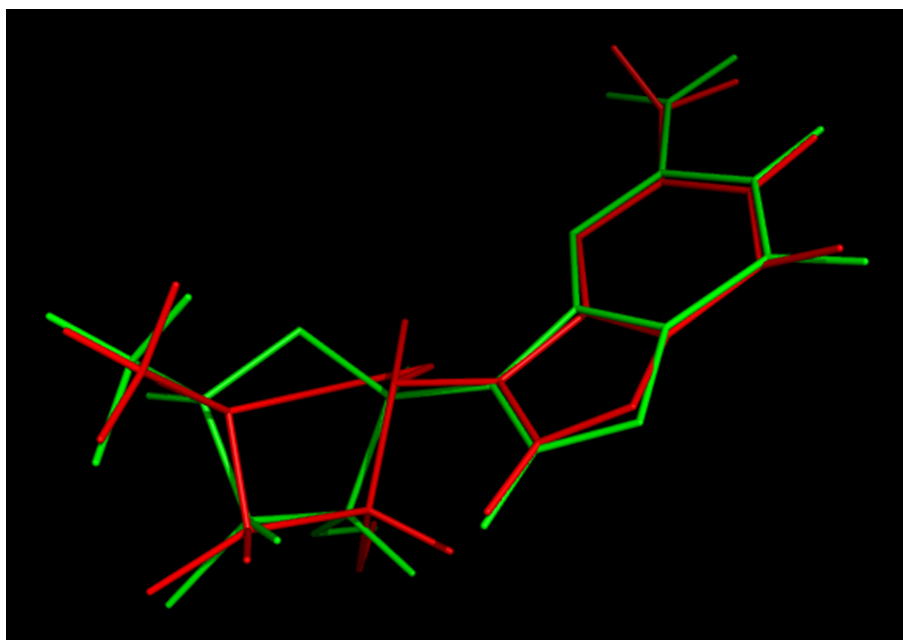


Figure 10: Alignment of the guanine marked with a square in RNA 1r4h (Figure 9) and experimental data.

low to keep the oxygen atom in its correct position. This problem can be solved by changing the torsion parameter of the AMBER force field.

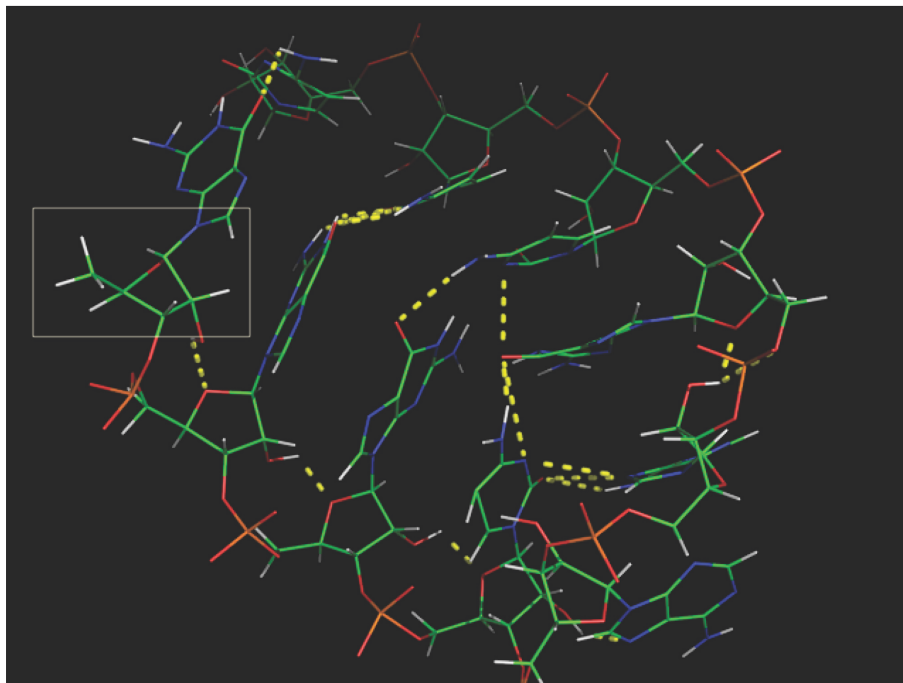


Figure 11: Hydrogen bonds in 1r4h showed in dotted lines.

Instead of running the genetic algorithm again to improve the values of this specific parameter, we took advantage of the fact that the GA works with multiple solutions simultaneously, and searched for a new individual in the final population according to two criteria: the RMSE of the new set of parameters should be as close as possible to the best parameter set, and the value of V_n of the guanine had to be higher than the ones of the returned parameter set.

The new selected parameter set, the third best in terms of RMSE in the final population, provides an RMSE of about 0.02 kcal/mol higher than the original parameter set and the values of V_n are, on average, 0.33 kcal/mol higher. These new parameters are presented in Table 3. Although the values are quite different from the first selected parameter set, this difference is not significantly reflected in the RMSE value. However, when we run Tinker again with these new data, the new structure found is more similar to the experimental data,

as shown in Figure 12. Comparing this new alignment with the one showed in Figure 10 one can notice that it is more precise, and that the oxygen atoms are now at their expected positions, regardless the missing hydrogen bonds.

The new higher precise alignment is reflected in the RMSD values showed in Table 4. Note that the values were further improved (0.27 against 0.33 in the first GA parametrization). This final result shows an improvement of about 82% (over 1Å) using GA in comparison with the results obtained in reference [10].

Table 3: New parameters selected from the GA for guanine

Nucleotide	Torsion	n^a	GA-New		GA-Old	
			$Vn/2^b$	γ^c	$Vn/2^b$	γ^c
Guanine	OS-CT-N9-CK*	1	1.1954	113.57	0.5327	50.60
		2	0.0996	5.76	0.0002	0.8
		3	0.0004	13.11	0.0001	1.96
		4	0.7215	4.27	0.3919	3.91

^a The periodicity of the torsion. ^b Magnitude of torsion in kcal/mol.

^c Phase off set in deg.

* see Figure 2 for details.

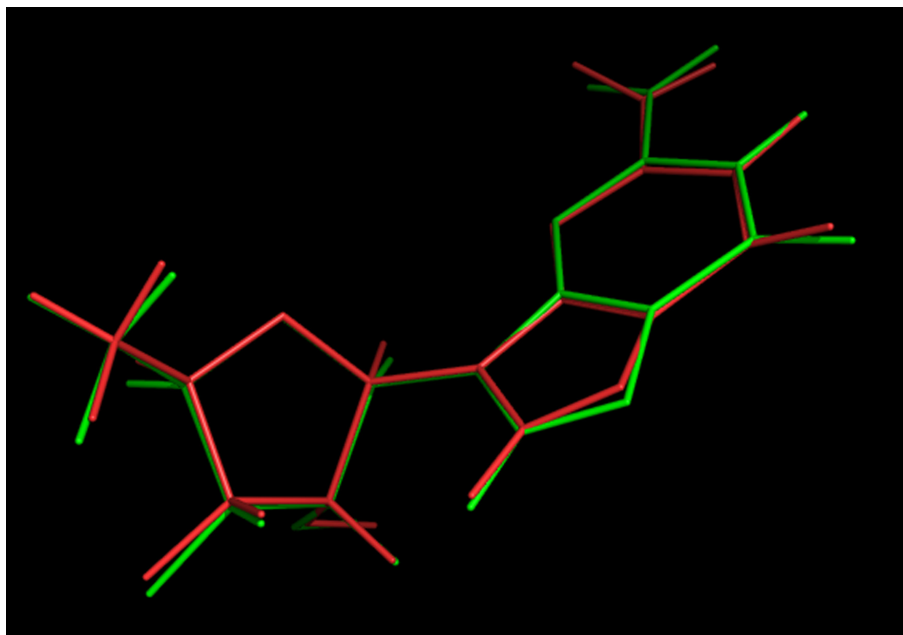


Figure 12: Alignment of the new guanine parameters obtained by the GA (red) compared with experimental data (green).

Table 4: Root Mean Square Deviation for the 1r4h with the new adenine parameters

	RMSD
Reference [10]	1.5
GA - Original param.	0.33
GA- Modified param.	0.27

The new plot of the 1r4h molecule compared against experimental data (green) is showed in Figure 13. This figure compares the updated genetic algorithms structure (A) and the one found in reference [10] (B) against PDB experimental data. As observed, the GA alignment is closer to experimental data than the results obtained by reference [10]. Notice that the problem with the guanine (showed in Figure 9) was solved. One can also observe that the rings of the nucleotides bases are better represented than the structure skeleton considering the final refinement. This happens because the algorithm focused its optimization in the glycosidic bonds, improving the RMSD value with respect to the experimental data with computer-generated structures and reducing the parameter value from 1.5 to 0.27.

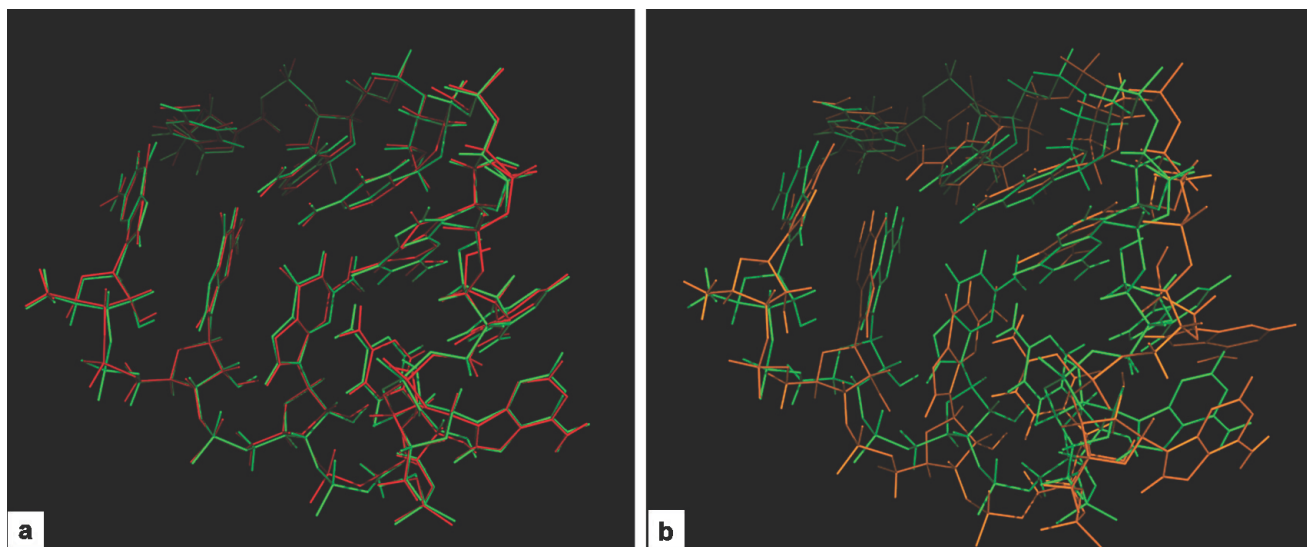


Figure 13: Comparison of the structures found by the two computational optimization and experimental data. (a) Optimization generated with parameters found by the GA (red) and experimental data (green). (b) Optimization generated with parameters found in reference [10] (orange) and experimental data (green).

Conclusion

In summary, the present study has proposed the use of chemical quantum calculations as references for a genetic algorithm to optimize the parameters of the dihedral term of the glycosidic torsion for better accuracy in representation of RNA. According to Lankas *et al.*⁴⁵ this is the most relevant bond in nucleic acids. We refined the parameters of the Amber force field (ff99).

The objective of the methodology was to further reduce the error of the computational calculations involved in the parametrization of the force field with particular attention to nucleic acids torsion energies. The new parametrization reduced, on average, the RMSE calculated with the differences of quantum and molecular mechanics energies in 50%. As demonstrated in the present work, the parameters found led to a better structural representation of the 1r4h molecule when compared to parametrizations previously reported in the literature [10]. In the latter case it is important to emphasize that the actual approach was able to go further and predict with a reasonable level of accuracy an experimental structure (NMR) that was not taken into account in refinement process.

This new way of optimizing parameters will bring improvements in studies of molecules behaviours and properties, since it showed to be more precise than other methodologies previously proposed. In addition, the method can be also applied to refine parameters of other terms of the force field, allowing us in the future to have a completely parametrized and more precise force fields.

Acknowledgement

We thank CENAPADs (UFRS, UFMG, UFPE) for help us in the quantum calculations and Dr. Lucas Bleicher for help in the final refinement section. We also thank the Brazilian National Council of Research (CNPq), Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) and the FAPEMIG Foundation for supporting this work.

References

- (1) Ditzler, M A and Otyepka, M and Sponer, J and Walter, N G, *Accounts of Chemical Research*, 2010, 43, 40-47.
- (2) Sponer, J and Mládek, A and Sponer, J E and Svozil, D and Zgarbová, M and Banás, P and Jurecka, P and Otyepka, M, *Physical Chemistry Chemical Physics*, 2012, 14, 15257-15277.
- (3) Perez, A and Marchan, I and Svozil, D and Sponer, J and Cheatham, T E and Laughton, C A and Orozco, M, *Biophysical Journal*, 2007, 92, 3817-3829.
- (4) Auffinger, P and Westhof, E, *Current Opinion in Structural Biology*, 1998, 8, 227-236.
- (5) Cornell, W D and Cieplak, P and Bayly, C I and Gould, I R and Merz, K M and Ferguson, D M and Spellmeyer, D C and Fox, T and Caldwell, J W and Kollman, P A, *Journal American Chemistry Society*, 1995, 117, 5179-5197.
- (6) Foloppe, N and MacKerell, A D, *Journal of Computational Chemistry*, 2000, 21, 86-104.
- (7) Wang, J M and Cieplak, P and Kollman, P A, *Journal of Computational Chemistry*, 2000, 21, 1049-1074.
- (8) Bosch, D and Foloppe, N and Pastor, N and Pardo, L and Campillo, M, *Biophysical Journal*, 2007, 92, 3817-3829.
- (9) Foloppe, N and MacKerell, A D, *Journal Physical Chemistry*, 1998, 34, 6669-6678.
- (10) Zgarbová, M and Otyepka, M and Sponer, J and Mládek, A and Banás, P and Cheatham, T. E and III and Jurecka, P, *Journal Chemistry Theory Computational*, 2011, 7, 2886-2902.
- (11) Reddy, S Y and Leclerc, F and Karplus, M, *Biophysical Journal*, 2003, 84, 1421-1449.

- (12) Besseova, I and Otyepka, M and Reblova, K and Sponer, J, *Physical Chemistry Chemical Physics*, 2009, 11, 10701-10711.
- (13) Deng, N J and Cieplak, P, *Biophysical Journal*, 2010, 98, 627-636.
- (14) Ricci, C. G and de Andrade, A S C and Mottin, M and Netz, P A, *Biophysical Journal*, 2010, 98, 627-636.
- (15) Banás, P and Mladék, A and Otyepka, M and Zgarbová, M and Jurecka, P and Svozil, D and Lankas, F and Sponer, J, *Journal Chemistry Theory Computational*, 2012, 8, 2448-2460.
- (16) Cheng, Y B and Sykulski, J K, *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, 1996, 9, 59-69.
- (17) Buchvarova, M and Velinov, P I Y, *Advances in Space Research*, 2010, 45, 1026-1034.
- (18) Yildirim, I and Stern, H A and Kennedy, S D and Tubbs, J D and Turner, D H, *Journal of Chemical Theory and Computation*, 2010, 6, 1520-1531.
- (19) Sakae, Y and Okamoto, Y, *Journal of Theoretical and Computational Chemistry*, 2004, 3, 339358.
- (20) Kästner, J and Carr, J M and Keal, T W and Thiel, W and Wander, A and Sherwood, P, *Journal Physical Chemistry*, 2009, 113, 11856-11865.
- (21) Silva, M X and Galvão, B R L and Belchior, J C, *Physical Chemistry Chemical Physics*, 2014, 16, 8895-8904.
- (22) Rodrigues, D D C and Nascimento, A M and Duarte, H A and Belchior, J C, *Chemical Physics*, 2008, 349, 91-97.
- (23) Guimarães, F F and Belchior, J C and Johnston, R L and Roberts, C, *Physical Chemistry Chemical Physics*, 2002, 19, 8327-8333.

- (24) Büyükata, M and Borges, E and Belchior, J C and Braga, J P, Physical Chemistry Chemical Physics, 2002, 19, 8327-8333.
- (25) Alden H. Wright, Genetic Algorithms for Real Parameter Optimization, Foundations of Genetic Algorithms, 1991, 205-218, Morgan Kaufmann.
- (26) Thomas Back, Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms, Oxford University Press, 1996, Oxford, UK.
- (27) A A Freitas, Data Mining and Knowledge Discovery with Evolutionary Algorithms, Springer: Germany, 2002.
- (28) Schmidt, M W and Baldrige, K K and Boatz, J A and Elbert, S T and Gordon, M S and Jensen, J H and Koseki, S and Matsunaga, N and Nguyen, K A and Su, S J and Windus, T L and Dupuis, M and Montgomery, J A, Journal Computational Chemistry, 1993, 14, 1347-1363.
- (29) Cheatham, T E and Cieplak, P and Kollman, P A, Journal Biomolecular Structure Dynamic, 1999, 16, 845-862.
- (30) C Froese-Fischer, The Hartree-Fock Method for Atoms, Wiley: New York, 1997.
- (31) Bhandarkar S M and Zhang H, IEEE Transactions on Evolutionary Computation, 1999, 3, 1-21.
- (32) Zhou, P-P and Qiu, W-Y, Journal Physical Chemistry, 2009, 113, 10306-10320.
- (33) Yao, X X and Ji, C G and Xie, D Q and Zhang, J Z H, Journal of Computational Chemistry, 2013, 34, 1136-1142.
- (34) Rubenstein, S and Kundrot, C and Huston, S and Dudek, M and Kong, Y and Hart, R and Hodsdon, M and Pappu, R and Mooij, W and Loeffler, G and Vorobieva, M

- and Sokolova, N and Bagossi, P and Ren, P and Carlsson, A and Kutepov, A and Grossfield, A and Schnieders, M and Gohara, D and Darden, T.
- (35) Shah, P H and Batra, R C, Computational Materials Science, 2014, 83, 349-361.
- (36) Zhong, A and Jiang, X and Hu, Y and Du, C, Journal of the Society of Leather Technologists and Chemists, 2013, 97, 121-124.
- (37) Saito, S and Ohno, K and Suzuki, T and Sakuraba, H, Molecular Genetics and Metabolism, 2012, 105, 244-248.
- (38) Guvench, O and MacKerell, A D, Journal Molecular Modeling, 2008, 14, 667-679.
- (39) An Information Portal to Biological Macromolecular Structures, 2014, <http://www.rcsb.org/pdb/home/home.do>.
- (40) K Coutinho, Computer Simulation of Liquids using DICE, Symposium in Memory of Michael C. Zerner, 2000.
- (41) Aqvist, J, Journal Physical Chemistry, 1990, 94, 8021-8024.
- (42) A M Lesk, Introdução à Bioinformática, Artmed: São Paulo, 2005.
- (43) P E Bourne and H Weissig, Structural bioinformatics, John Wiley & Sons: New Jersey, 2003.
- (44) Schrödinger, L L C, 2010.
- (45) Lankas, F and Spackova, N and Moakher, M and Enkhbayar, P and Sponer, J, Nucleic Acids Research, 2010, 38, 3414-3422.