Tutorial CBMC DICE

Henrique Musseli Cezar

25 de junho de 2019

1 Introdução

Esse tutorial tem como objetivo guiar o usuário do DICE [1] a executar uma simulação da molécula de óxido mesityl (MOx, mostrada na Figura 1) flexível, utilizando o algoritmo Configurational Bias Monte Carlo (CBMC) com fragmentos, [2] em fase gasosa e em solução. No método CBMC a molécula é fragmentada em partes menores (exibidas na Figura 1) e através de um processo de reconstrução, a amostragem conformacional é realizada.

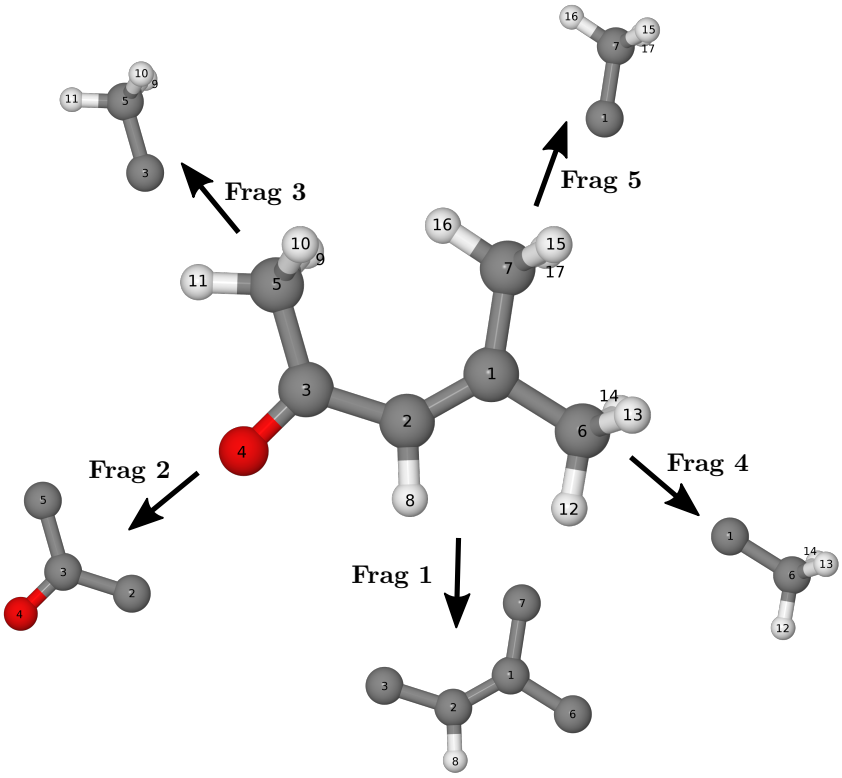


Figura 1: Molécula de óxido mesityl (MOx) e seus fragmentos. Figura da referência [3].

Partiremos da geometria da molécula e faremos todo o processo desde a parametrização até as simulações. Para a parametrização, mostraremos como os parâmetros de campos de força clássicos conhecidos como OPLS e AMBER podem ser refinados para reproduzir o potencial torcional de rotação de moléculas complexas. Esse refinamento

será feito através do cálculo da energia total com métodos de mecânica quântica para configurações referentes a rotação de fragmentos.

Concluída a parametrização, será mostrado como fazer a simulação com o DICE e como algumas análises podem ser realizadas.

2 Dependências

Para gerar os inputs necessários para fazer uma simulação CBMC com o DICE, utilizaremos o DICEtools (`https://github.com/hmcezar/dicetools`). O DICEtools é um conjunto de ferramentas escritas em Python 3 que auxiliam a gerar arquivos de entrada e analisar dados de simulações realizadas com o DICE.

Essas ferramentas tem algumas dependências. Recomendamos a utilização do Anaconda Python 3.x (`https://www.anaconda.com/distribution`) para utilizá-las, pois este facilita a instalação e manutenção dessas bibliotecas. A lista de bibliotecas atualizadas a serem instaladas se encontram na página do GitHub do DICEtools, contudo, replico aqui os requisitos na data deste tutorial.

```
$ conda install numpy
$ conda install -c openbabel openbabel
$ conda install matplotlib
$ conda install scipy
$ conda install sip
$ conda install pyqt=5
$ pip install rmsd
```

Caso já tenha o DICEtools em seu computador e tenha o git instalado em sua máquina, você pode atualizar para a última versão executando no diretório onde os programas de encontram o comando

```
$ git pull
```

3 Gerando `.dfr` e `.txt`

Para gerar os arquivos `.dfr` e `.txt`, necessários para uma simulação CBMC com o DICE, você precisará de um arquivo `.xyz`, `.pdb` ou em algum formato similar da geometria otimizada da molécula.

Se você pretende fazer uma simulação com fragmentos rígidos e parametrizar os potenciais de torção entre fragmentos, leia a subseção 3.1 "Usando o fragGen". Se preferir fazer uma simulação com fragmentos também flexíveis e/ou quiser um conjunto de parâmetros iniciais, leia a subseção 3.2 "Usando o LigParGen e gromacs2dice".

3.1 Usando o fragGen

No diretório onde se encontra o arquivo com a geometria da molécula, execute o comando:

```
$ python /path/to/dicetools/fragGen.py mesityloxide.xyz
```

Ele irá gerar um diretório contendo os fragmentos da molécula em arquivos separados para visualização, e os arquivos `.dfr` e `.txt`.

Para ver as opções disponíveis do fragGen e das outras ferramentas do DICEtools, execute o programa com a opção `-h`:

```
$ python /path/to/dicetools/fragGen.py -h
```

Depois de ter os arquivos de entrada, precisamos editar o `.txt` com as cargas dos átomos e os parâmetros do potencial de Lennard-Jones (LJ). As cargas podem ser obtidas a partir do campo de força escolhido, ou ainda utilizando algum método de fit eletrostático em cálculos de estrutura eletrônica como, por exemplo, o CHELPG. Os parâmetros de LJ devem ser escolhidos apropriadamente dado o campo de força escolhido. Para esse tutorial, um .txt com esses parâmetros é fornecido.

Já o `.dfr` deve ser editado com os parâmetros referentes aos torcionais entre os fragmentos. Caso tenha parâmetros para esses torcionais, basta preenche-los. Faremos isso para os potenciais de rotação dos fragmentos com CH_3. Para isso, preencha os valores no .dfr conforme ilustrado abaixo:

```
11 5 3 2 OPLS 0.0 0.0 0.275 0.0 0.0 0.0
10 5 3 2 OPLS 0.0 0.0 0.275 0.0 0.0 0.0
9 5 3 2 OPLS 0.0 0.0 0.275 0.0 0.0 0.0
11 5 3 4 OPLS 0.0 0.0 0.0 0.0 0.0 0.0
10 5 3 4 OPLS 0.0 0.0 0.0 0.0 0.0 0.0
9 5 3 4 OPLS 0.0 0.0 0.0 0.0 0.0 0.0
17 7 1 2 OPLS 0.0 0.0 -0.372 0.0 0.0 0.0
14 6 1 2 OPLS 0.0 0.0 -0.372 0.0 0.0 0.0
16 7 1 2 OPLS 0.0 0.0 -0.372 0.0 0.0 0.0
15 7 1 2 OPLS 0.0 0.0 -0.372 0.0 0.0 0.0
12 6 1 2 OPLS 0.0 0.0 -0.372 0.0 0.0 0.0
13 6 1 2 OPLS 0.0 0.0 -0.372 0.0 0.0 0.0
15 7 1 6 OPLS 0.0 0.0 0.3 0.0 0.0 0.0
13 6 1 7 OPLS 0.0 0.0 0.3 0.0 0.0 0.0
16 7 1 6 OPLS 0.0 0.0 0.3 0.0 0.0 0.0
14 6 1 7 OPLS 0.0 0.0 0.3 0.0 0.0 0.0
12 6 1 7 OPLS 0.0 0.0 0.3 0.0 0.0 0.0
17 7 1 6 OPLS 0.0 0.0 0.3 0.0 0.0 0.0
```

Você deve ter notado que não colocamos os parâmetros para as rotações entre os fragmentos 1 e 2, que giram em torno da ligação C2-C3. Neste tutorial, iremos encontrar esses parâmetros baseados em cálculos de estrutura eletrônica efetuados com o Gaussian, pois os campos de força convencionais não reproduzem corretamente as populações das conformações. Para isso utilizaremos na próxima seção uma outra ferramenta do DICEtools, o plot_eff_tors. Por hora, devemos apenas preencher os parâmetros no `.dfr` com 0.0 e escolher o tipo de torcional como "OPLS", ou seja, as linhas dos torcionais no arquivo, ficarão com a forma:

```
4 3 2 1 OPLS 0.0 0.0 0.0 0.0 0.0 0.0
4 3 2 8 OPLS 0.0 0.0 0.0 0.0 0.0 0.0
5 3 2 1 OPLS 0.0 0.0 0.0 0.0 0.0 0.0
5 3 2 8 OPLS 0.0 0.0 0.0 0.0 0.0 0.0
```

Os parâmetros já com os valores corretos podem ser encontrados no arquivo MSO_-dihedrals_opls.txt que acompanha esse tutorial. Caso deseje parametrizar todos os diedros entre fragmentos e, portanto, não utilizar nenhuma informação de campos de força, você pode definir todos os parâmetros de torcional como sendo nulos e realizar o scan e parametrização para cada uma das uniões entre fragmentos. Para fazer isso, realize o procedimento da seção 4 para cada torção entre fragmentos, até todos os parâmetros estarem preenchidos.

3.2 Usando o LigParGen e gromacs2dice

Você só precisa fazer essa seção de optou por partir de um conjunto inicial de parâmetros e não fez os passos da subseção 3.1.

O primeiro passo para fazer a parametrização utilizando o LigParGen é submeter ao servidor do LigParGen (`http://zarbi.chem.yale.edu/ligpargen/`) o `.pdb` da molécula otimizada. Atenção para os parâmetros da tela do LigParGen: caso esteja fornecendo uma geometria já otimizada, defina o "Molecule Optimization Iterations" como zero.

Depois que o LigParGen rodar, você será levado a uma nova página, onde há uma série de arquivos disponíveis para download. Faça download dos arquivos `.gro` e `.itp` do GROMACS. Esses arquivos contém, respectivamente, a geometria da molécula e os parâmetros do campo de força (conhecido como topologia). Vale ressaltar que muitas vezes essa parametrização não reproduz todas as propriedades da molécula, e por isso realizaremos um refinamento através do ajuste dos potencial de torção em torno da ligação de define as conformações *syn* e *anti* nas seções 4 e 5.

O LigParGen costuma mudar a ordem dos átomos do `.pdb` que o usuário envia ao servidor, o que pode ser inconveniente. Para garantir que a ordem dos átomos é a mesma do `.pdb` original utilize o reorder_ligpargen:

```
$ python /path/to/dicetools/reorder_ligpargen.py original.pdb ligpargen.
    ↪ gro ligpargen.itp
```

Serão gerados dois arquivos: um `.gro` e outro `.itp` com o prefixo "reordered_" no nome. São esses os arquivos que devem ser utilizados no restante do tutorial.

Com os arquivos do GROMACS tendo os átomos na ordem correta, faça a conversão dos arquivos de entrada do GROMACS para arquivos de entrada do DICE com o gromacs2dice

```
$ python /path/to/dicetools/gromacs2dice.py reordered_ligpargen.itp
    ↪ reordered_ligpargen.gro -g -p /usr/share/gromacs/top/
The files reordered_mox_ligpargen.txt and reordered_mox_ligpargen.dfr
    ↪ were successfully generated.
```

A opção `-g` é utilizada para garantir que as distâncias e ângulos de equilíbrio do campo de força são as da geometria fornecida inicialmente e a opção `-p` aponta para o diretório "top" da instalação do GROMACS, onde os parâmetros dos campos de força são definidos.[1]

Se a conversão for bem sucedida, você terá no diretório um arquivo `.txt` e um `.dfr`, necessários para a simulação com o CBMC. Renomeie esses arquivos para `mesityloxide.txt` e `mesityloxide.dfr`, e edite o `mesityloxide.txt` substituindo o nome da molécula de reordered_ligpargen para mesityloxide na terceira linha. Feito isso, podemos dar prosseguimento ao tutorial.

[1] Basta ter uma instalação simples do GROMACS que pode ser até do repositório de sua distribuição Linux (para Ubuntu, Debian e derivados, instalar com "sudo apt-get install gromacs")

4 Gerando o scan dos ângulos de torção com o plot_eff_tors

Utilizaremos o plot_eff_tors para gerar o scan em torno da ligação C2-C3 e obter o perfil de energia da rotação com cálculos de estrutura eletrônica, pois os parâmetros dos campo de força mais comuns encontrados na literatura não reproduzem as populações corretamente. Para obter o perfil de energia referente a rotação, iremos calcular qual a energia total do sistema fazendo cálculos *single point* em 36 configurações. As geometrias e arquivo de entrada para esse cálculo são obtidos utilizando o plot_eff_tors, disponível no DICEtools.

O nível de cálculo utilizado e informações referentes ao número de processadores e memória são informados em um arquivo de texto (que chamarei de `topmp2.txt`) que contém o cabeçalho referente ao cálculo do Gaussian a ser realizado:

```
%nprocshared=6
%mem=54GB
#MP2/aug-cc-pVDZ Density=Current Pop=MK

MOx scan

0 1
```

Quando necessário, também é possível utilizar um arquivo `.txt` cujo conteúdo é adicionado ao final de cada geometria para o cálculo.

Nos basearemos no ângulo de diedro definido por C1-C2-C3-O4 para a definição do ângulo de torção. Utilizando os arquivos `.dfr` e `.txt` produzidos na seção 3, rodamos os plot_eff_tors com as opções:

```
$ python /path/to/dicetools/plot_eff_tors.py mesityloxide.txt
   ↪ mesityloxide.dfr 1 2 3 4 36 --gausstop topmp2.txt --printxyz
```

Os números que aparecem em seguida dos arquivos `.txt` e `.dfr` são os números dos átomos usados para definir o ângulo de torção, e o número 36 é o número de configurações geradas (o número de cálculos a ser realizado com o Gaussian). O `--gausstop` especifica o arquivo que contém o cabeçalho do Gaussian. A opção `--printxyz` gera um arquivo .xyz com as rotações, que deve ser visualizado para garantir que as geometrias utilizadas no cálculo do Gaussian são as esperadas. Novamente, rodar o script com a opção `-h` mostra as opções disponíveis.

Como saída, são gerados alguns arquivos `.pdf` mostrando o potencial total, torcional e não ligado, e o principal nesse momento, um arquivo `.gjf` contendo o input para o Gaussian. Note que os dois lados da molécula durante a rotação são mantidos rígidos. Isso significa que a geometria não é relaxada em cada ponto (fazemos um cálculo *single point*), o que pode fazer com que os mínimos da curva não sejam perfeitamente descritos. Para boa parte das moléculas, a rotação de partes rígidas é uma aproximação bem razoável, mas em alguns casos, a otimização da molécula em torno de cada mínimo pode ser necessária, para que as diferenças de energia entre os confôrmeros sejam obedecidas pelo potencial torcional. No caso do MOx, o arquivo `.log` fornecido utilizou esse procedimento: primeiro foi feito o *scan* partindo do confôrmero *syn* em torno de $0°$ e os estados de transição; depois um scan em torno do mínimo da geometria *anti* otimizada foi realizado entre os ângulos em torno de $\pm 180°$. Esses arquivos foram unidos no arquivo `.log` fornecido como exemplo, e nele podemos ver que o

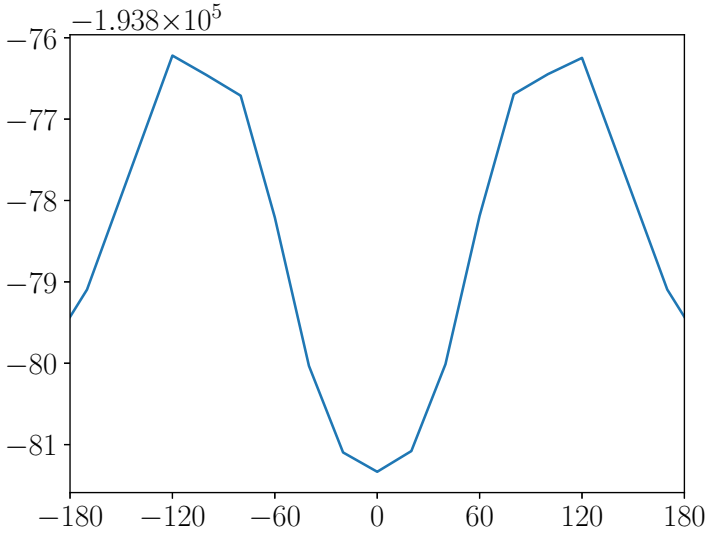


Figura 2: Gráfico produzido pelo plot_en_angle_gaussian_scan à partir do `.log` do scan do ângulo de diedro.

fato do confôrmero *syn* ter a energia cerca de $1.9\,\mathrm{kcal/mol}$ menor que o confôrmero *anti* em fase gasosa é obedecido.

Depois que os cálculos forem concluídos, você pode gerar um gráfico da energia total pelo ângulo torcional de referência utilizando o script plot_en_angle_gaussian_-scan

```
$ python /path/to/dicetools/plot_en_angle_gaussian_scan.py tors_1-2-3-4
    ↪ _scan.log > curve_tors_1-2-3-4_scan.dat
```

O script irá gerar um gráfico chamado scan_gaussian.eps (Figura 2) contendo a curva calculada. Compare esse gráfico com o tors_1-2-3-4_total.pdf, gerado pelo plot_eff_tors e que contém o potencial clássico utilizando os parâmetros que temos até o momento (parâmetros torcionais iguais a zero se fez a seção 3.1). A diferença entre esses gráficos deve vir dos parâmetros dos torcionais, que serão obtidos através do ajuste da curva calculada com o Gaussian.

5 Ajuste dos parâmetros do campo de força para reproduzir cálculos quânticos

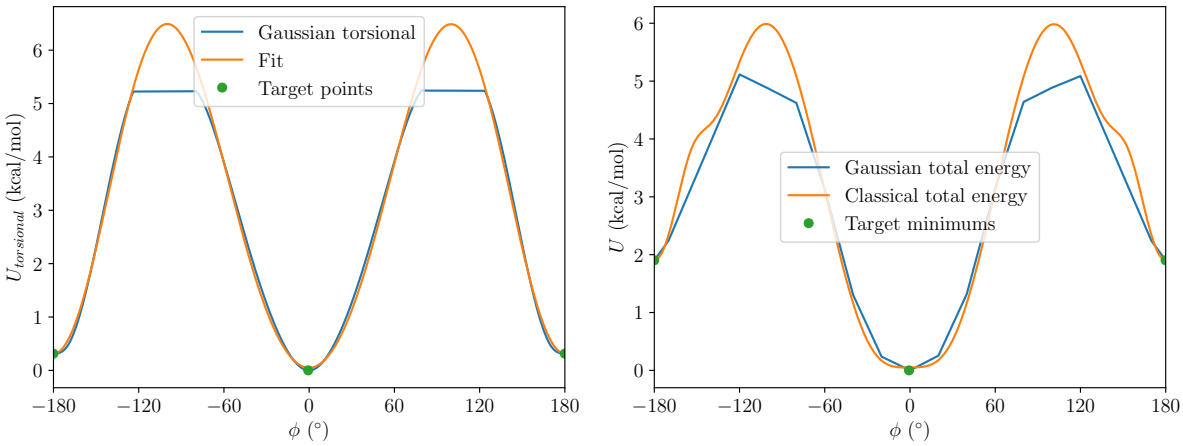
Para fazer o ajuste dos parâmetros dos torcionais e obter a energia o mais próximo possível dá energia calculada com o Gaussian, utilizaremos o fit_torsional.py, que está disponível no DICEtools. Esse programa, recebe como parâmetros o `.log` do Gaussian, o `.dfr` com os parâmetros referentes a torção nulos (ou fornecendo um chute inicial) e o `.txt` contendo os parâmetros corretos para as cargas e potencial de Lennard-Jones. Além disso, o programa recebe os números dos átomos que definem o diedro de referência (o mesmo utilizado no plot_eff_tors).

Com todos os arquivos preparados, o ajuste dos parâmetros é feito com o comando

```
$ python /path/to/dicetools/fit_torsional.py tors_1-2-3-4_scan.log
    ↪ mesityloxide.dfr mesityloxide.txt 1 2 3 4 --fit-to-spline >
    ↪ fit_mesityloxide.dfr
```

A opção `--fit-to-spline` é utilizada para ajustar a curva à uma curva interpolada dos dados do `.log`. Nesse caso, utilizamos essa opção pois não temos muitos pontos entre o estado de transição e os mínimos em $\pm 180°$ referentes a conformação *anti*.

Ao executar o comando, dois gráficos como os mostrados na Figura 3, com os nomes fit_torsional_tors_1-2-3-4_scan.pdf e fit_total_en_tors_1-2-3-4_scan.pdf contendo o seu ajuste e a energia total final, respectivamente, serão gerados:



(a) Potencial torcional

(b) Energia potencial total

Figura 3: Ajuste do potencial torcional em cima do potencial quântico utilizando o fit_torsional. No gráfico da esquerda são mostrados a energia potencial torcional (energia total menos a de Lennard-Jones e Coulomb) e no da direita a energia potencial total.

Os gráficos representam representam:

(a) Em azul: energia torcional obtida com os cálculos de mecânica quântica. Essa curva é obtida subtraindo a energia potencial não ligada (termos de Lennard-Jones e Coulomb), o que resulta na energia torcional. Note que os máximos nessa curva são "achatados" para que o ajuste (que é feito sobre essa curva) não tente ajustar os máximos, que não são se grande importância nas simulações com CBMC. Como utilizamos a opção `--fit-to-spline`, essa curva é uma interpolação sobre os pontos da curva quântica.
Em laranja: curva ajustada a curva azul. Durante o ajuste, os pontos em verde tem um peso maior, buscando descrever os mínimos e diferenças de energia entre mínimos mais precisamente.

(b) Em azul: energia total dos cálculos de mecância quântica. O potencial total clássico obtido com o ajuste deve ser similar a esse, especialmente os mínimos. Os pontos verdes indicam os mínimos, que tem um peso maior para o ajuste da curva.
Em laranja: energia potencial total clássica após o ajuste dos parâmetros. Deve ser similar à curva azul, especialmente em torno dos mínimos.

Veja que, idealmente, as curvas azul e laranja coincidiriam perfeitamente em ambos os gráficos. Como isso não é possível na maioria dos casos, o ajuste do fit_torsionals prioriza uma melhor concordância com os mínimos, que dão as populações durante as

simulações. Isso é feito atribuindo um peso maior aos mínimos da energia total quântica, representados pelos pontos verdes, durante o ajuste. Você pode desligar esse peso atribuído aos mínimos com a opção `--no-force-min`, e modificar o peso atribuído aos mínimos com a opção `--weight-minimums`. A porcentagem de pontos de alta energia descartados (gerando o "achatamento" da curva verde) é controlado pela opção `--cut-energy`, que tem como padrão o valor 0.3 (descartar $30\,\%$ dos pontos de mais alta energia).

Após a execução do programa, verifique no arquivo para o qual você redirecionou a saída, no caso o fit_mesityloxide.dfr, se os parâmetros foram devidamente incluídos. Você deve ter encontrado um conjunto de parâmetros para esse torcional parecido com os mostrados abaixo:

```
4 3 2 1 OPLS -4.954 4.274 -3.920 0.0 0.0 0.0
4 3 2 8 OPLS 4.999 4.436 5.000 0.0 0.0 0.0
5 3 2 1 OPLS -4.686 -3.862 -5.000 0.0 0.0 0.0
5 3 2 8 OPLS 3.922 1.258 5.000 0.0 0.0 0.0
```

Caso os parâmetros tenham sido devidamente incluídos no seu novo `.dfr`, substitua o arquivo anterior mesityloxide.dfr com esse novo arquivo. Se estiver fazendo a parametrização de mais de um torcional, use esse novo `.dfr` como base para a nova rodada do fit_torsional, e repita o procedimento até a parametrização de todos os torcionais ser concluída.

6 Simulações

Com os arquivos `.dfr` e `.txt` devidamente definidos é possível iniciar as simulações com o CBMC no DICE. Inicialmente, é recomendado fazer uma breve simulação em fase gasosa para identificar possíveis problemas com arquivos de entrada e campo de força. Por rodar com apenas uma molécula, essa simulação termina rapidamente, e pode evitar horas de CPU desperdiçadas se a parametrização do campo de força não estiver condizente com o esperado. Nas próximas subseções serão mostradas as palavras-chave utilizadas pelo algoritmo CBMC e como uma simulação utilizando esse método é feita. Para manter o tutorial sucinto, assumirei que o usuário já possui algum conhecimento prévio sobre simulações de moléculas rígidas com o DICE.

6.1 Fase gasosa

Para a simulação em fase gasosa, criaremos um arquivo `.in` (um `.ter` para a termalização não é necessário nesse caso) para uma simulação no ensemble NVT de uma única molécula de MOx. Utilizamos também uma densidade baixa ($0.001\,\mathrm{g/cm^3}$) para ter uma única molécula em uma caixa grande. Um exemplo de arquivo `.in` pode ser visto abaixo.

```
title = mesityloxide_gas
ljname = mesityloxide.txt
outname = outmox
ncores = 4
init = yes
coolstep = 0
nmol = 1
dens = 0.001
```

```
temp = 300.0
press = 1.0
vstep = 0
nstep = 500000
iprint = 1
isave = 50
irdf = 0
seed = 80466

# CBMC keywords
flex = mesityloxide
pcbmc = 1.0
```

Apenas duas palavras chave são, a princípio, necessárias para a simulação com o CBMC. A palavra-chave "flex" indica a lista de nomes de moléculas flexíveis a serem simuladas com o CBMC. Nesse caso, temos somente a molécula de MOx, cujo nome está especificado no arquivo `.txt` sendo a palavra que aparece logo em seguida do número de átomos:

```
17 mesityloxide (generated with fragGen)
```

Já a palavra-chave "pcbmc" define a probabilidade de, dado que uma molécula de um tipo flexível é selecionada, o movimento CBMC é realizado. O restante de probabilidade ($1.0-$pcbmc) é de que a molécula selecionada sofra uma rotação e translação mantendo a molécula rígida. Na simulação em fase gasosa (uma única molécula), a rotação e translação não alteram a energia do sistema, e por isso utilizamos pcbmc = 1.0.

Com esse arquivo de entrada `.in` á simulação em fase gasosa já pode ser realizada. Contudo, antes de executar o DICE, execute no terminal (ou adicione no seu script de submissão de job antes do comando de execução) os seguintes comandos:

```
$ export OMP_STACKSIZE=32M
$ ulimit -s unlimited
```

Esses comandos aumentam o tamanho da pilha e evitam que Falhas de Segmentação (Segmentation Fault) ocorram.

Feito isso, o DICE pode ser executado normalmente:

```
$ dice3 < nvt.in > nvt.out
```

Para os 5×10^5 passos definidos no `.in` a simulação deve rodar em pouco mais de um minuto. Após a execução, verifique no arquivo de saída (no comando acima, nvt.out) se a execução terminou normalmente.

Caso tenha terminado, verifique qual foi o resultado da amostragem com o CBMC, por exemplo, olhando a distribuição do ângulo de diedro definido pelos átomos C1-C2-C3-O4. Para isso, calcule o ângulo de diedro para todas as configurações da trajetória simulada e faça o gráfico da evolução do ângulo de diedro com os passos de simulação:

```
$ python /path/to/dicetools/calculate_dihedrals.py outmox.xyz 1 2 3 4 >
    ↪ died_C1-C2-C3-O4.dat
$ python /path/to/dicetools/dihedral_step_evolution.py died_C1-C2-C3-O4.
    ↪ dat 50
```

No segundo comando, o valor 50 indica o intervalo de passos entre cada snapshot (valor da keyword isave do `.in`). Você irá obter um gráfico similar ao da Figura 4.

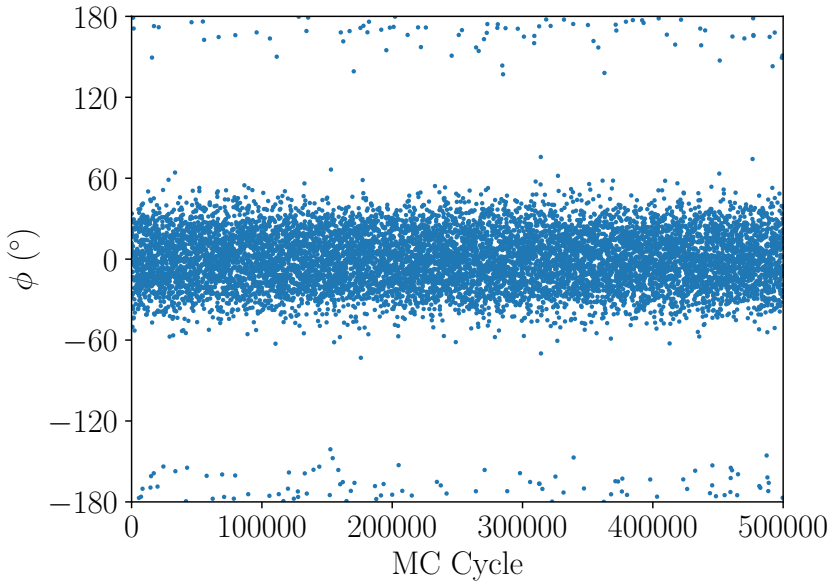


Figura 4: Evolução do ângulo de diedro C1-C2-C3-O4 com os passos de simulação para a simulação do MOx em fase gasosa.

Veja que a amostragem dos dois confôrmeros (região em torno de $\pm 180°$ e região em torno de $0°$) ocorre de maneira uniforme. Vemos também que a população de conformações *syn* (ângulo em torno de $0°$) é maior que a de conformações *anti* (ângulo em torno de $\pm 180°$). Essas populações podem ser obtidas com o script probability_interval do DICEtools, que conta o número de conformações com um ângulo de diedro em um certo intervalo e dá a porcentagem de configurações desse tipo. Para obter o número de conformações *syn*, considerando como sendo desse tipo conformações com o ângulo no intervalo $[-60°, 60°]$, rodamos:

```
$ python /path/to/dicetools/probability_interval.py died_C1-C2-C3-O4.dat
    ↪  -60 60
The probability of getting one value in the interval [-60.000000,
    ↪ 60.000000] is 98.11 %
```

Que retorna que cerca de $98\,\%$ das conformações amostradas foram *syn*, indicando que o restante (outros cerca de $2\,\%$) são de conformações *anti*.

6.2 Em água

Para fazer a simulação em solução (no caso em água) primeiro precisamos de uma caixa termalizada, considerando a molécula rígida na conformação inicial escolhida. Essa condição não é estritamente necessária, contudo, como a simulação CBMC roda de maneira mais lenta, iniciar de uma caixa termalizada é altamente recomendado. Caso deseje realizar a simulação de inicialização da caixa e termalização, utilize o arquivo npt.ter, que contém as palavras chave necessárias para a simulação de moléculas rígidas. Utilizaremos um sistema pequeno (com 500 moléculas de água) apenas para acelerar a simulação, mas note que na prática, caixas maiores devem ser necessárias. Lembre de acrescentar a parametrização da molécula de água no seu arquivo `.txt`

```
3 SPC/E (JCP,91,6269 (1987))
```

```
1 8 0.0000 0.0000 0.0000 -0.8476 0.155 3.165
2 1 0.5774 0.8165 0.0000 0.4238 0.000 0.000
2 1 0.5774 -0.8165 0.0000 0.4238 0.000 0.000
```

Nesse caso, utilizamos a parametrização SPC/E para as moléculas de água.

Introduziremos para essa simulação duas palavras-chave do CBMC que não foram utilizadas na simulação em gás: o ntrialphi e o equiphi. O ntrialphi define o número de ângulos tentativas de inserção que são utilizados na inserção de cada fragmento. De uma maneira geral, quanto maior esse número, maior a taxa de aceitação. Contudo, o custo computacional também cresce conforme maior o número de ângulos tentativa de inserção e por esse motivo, deve-se ter um compromisso entre taxa de aceitação e custo computacional. Um valor de ntrialphi de 16 costuma fornecer um bom compromisso nesse sentido. Já o equiphi, faz com que esses ângulos sejam distribuídos de maneira uniforme no intervalo $[0, 2\pi]$.

Quando temos somente o soluto flexível e as moléculas do solvente são rígidas, é interessante ter certeza que o soluto será modificado um número de vezes o suficiente para uma boa estatística. Para garantir isso, utilizamos nesse tipo de simulação o "preferential sampling", [4, 5] que seleciona a molécula de soluto e as moléculas de solvente ao seu redor mais frequentemente. Dessa maneira, mais movimentos CBMC são realizados e o número de mudanças conformacionais da molécula é maior. Apesar de desejarmos que um grande número de mudanças conformacionais sejam feitos para ter uma boa estatística, também é interessante que o ambiente em torno do soluto seja modificado, e por isso utilizamos pcbmc = 0.8, o que dá uma probabilidade de $20\,\%$ de realizar um movimento de rotação/translação do soluto. Ficamos então com um arquivo `.in` como o abaixo:

```
title = mesityloxide_wat
ljname = mesityloxide.txt
outname = outmox
ncores = 4
init = no
temp = 300.0
press = 1.0
accum = no
vstep = 37500
nstep = 4
iprint = 1
isave = 50
irdf = 0
iratio = 10
vratio = 10
seed = 92047

# preferential sampling (select solute and surrounding molecules more
    ↪ often)
sampling = 2

# CBMC keywords
flex = mesityloxide
pcbmc = 0.8
ntrialphi = 16
equiphi = yes
```

Mesmo se tratando de uma simulação de uma caixa com poucas moléculas de água e poucos passos, asimulação deve rodar em cerca de 40 minutos. Ao término da simu-

lação, iremos fazer uma análise da evolução do ângulo diedro como feita no caso da simulação em gás. Para acelerar o cálculo do ângulo de diedro em cada *snapshot* salvo da simulação, iremos primeiro extrair as configurações do soluto com o get_solute

```
$ python /path/to/dicetools/get_solute_xyz.py outmox.xyz 17 >
    ↪ solute_confs.xyz
```

o primeiro argumento (outmox.xyz) é o nome do arquivo `.xyz` contendo os snapshots e o 17 indica o número de átomos da molécula de soluto. Nesse caso, as configurações do soluto foram extraídas para o arquivo solute_confs.xyz, que usaremos para criar um gráfico como o da Figura 4

```
$ python /path/to/dicetools/calculate_dihedrals.py solute_confs.xyz 1 2
    ↪ 3 4 > died_C1-C2-C3-O4.dat
$ python /path/to/dicetools/dihedral_step_evolution.py died_C1-C2-C3-O4.
    ↪ dat 50
```

O que gera o arquivo died_C1-C2-C3-O4.pdf exibido na Figura 5.

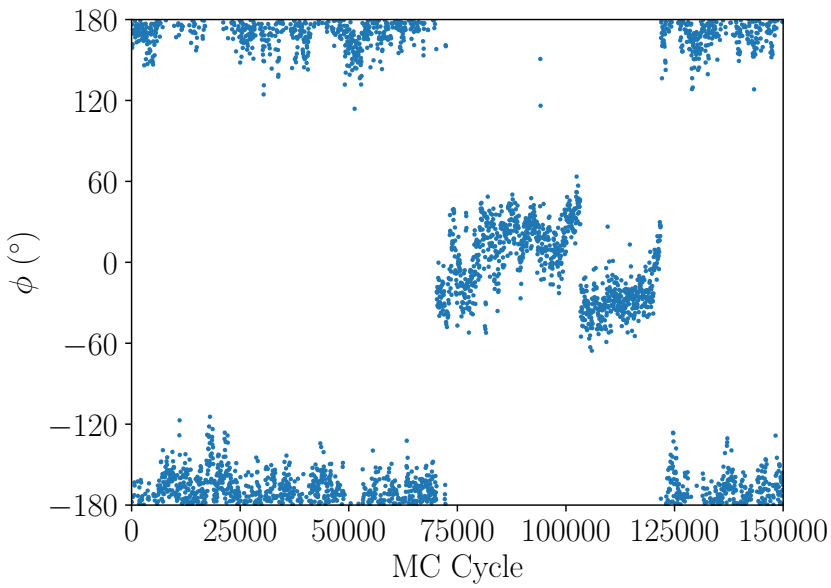


Figura 5: Evolução do ângulo de diedro C1-C2-C3-O4 com os passos de simulação para a simulação do MOx em água.

Vemos que nesse caso, o confôrmero dominante foi o confôrmero *anti*! Iremos investigar porque isso aconteceu, mas primeiro vamos verificar o quão mais frequente foi a amostragem da conformação *anti* com o probability_interval:

```
$ python /path/to/dicetools/probability_interval.py died_C1-C2-C3-O4.dat
    ↪  -60 60
The probability of getting one value in the interval [-60.000000,
    ↪ 60.000000] is 34.03 %
```

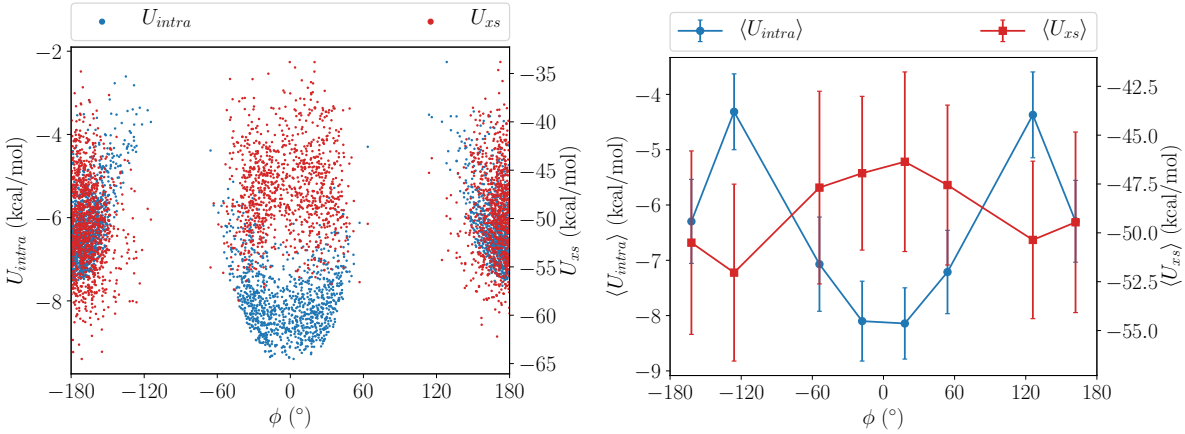
Vemos que $34\,\%$ das conformações amostradas foram *syn*, e o restante (outros cerca de $66\,\%$) são de conformações *anti*. Esses valores são dependentes da semente utilizada pois a simulação que fizemos não foi longa o suficiente. Para ter um resultado mais consistente, tente rodar a simulação para 10^6 ciclos de MC.

Mostraremos que a origem da inversão do confôrmero mais frequentemente amostrado é devido a introdução da interação com o solvente, que penaliza o confôrmero

syn que tem um momento de dipolo menor. Para isso utilizamos o solute_en_vs_torsion do DICEtools, que para cada ângulo de diedro, analisa a energia intramolecular U_{intra} e a energia soluto solvente U_{xs}. Fazendo a média dos valores de U_{intra} e U_{xs} para intervalos de ângulo levantamos uma curva que dá o valor médio e desvio padrão para diferentes valores de ângulo amostrados. Rodamos o solute_en_vs_torsion da seguinte maneira:

```
python /path/to/dicetools/solute_en_vs_torsion.py died_C1-C2-C3-O4.dat
    ↪ outmox.ien outmox.e12
```

E obtemos dois gráficos:



(a) Energia para cada ângulo amostrado

(b) Valores médios de energia para intervalos de ângulo.

Figura 6: Gráficos de saída do solute_en_vs_torsion.

O segundo gráfico é o mais interessante por mostrar os valores médios, onde vemos que a energia soluto solvente U_{xs} é cerca de $3\,\mathrm{kcal/mol}$ maior no confôrmero *syn* do que no *anti*. Por esse motivo, o aumento de cerca de $2\,\mathrm{kcal/mol}$ no confôrmero *anti* vindo da energia intramolecular é compensado, fazendo o confôrmero *anti* mais estável.

7 Além do básico: opções dentro do método CBMC

Nas seções anteriores realizamos as simulações CBMC com o DICE utilizando o mínimo possível de palavras chave. Contudo, existem opções dentro do método que podem ser ativadas com outras palavras chave. Por exemplo, não informamos o tamanho da biblioteca de fragmentos, e nem informamos como essa é gerada. As palavras chave abaixo podem ser utilizados no seu `.in` em alguns casos para melhorar a eficiência do método ou resolver problemas pontuais de sistemas específicos.

- `savefralib = yes`
  Salva as bibliotecas de fragmentos em arquivos. Útil para verificar a parametrização dos graus de liberdade rígidos.

- `biasphi = yes`
  Ao invés de gerar os κ_ϕ ângulos tentativa de inserção aleatoriamente (ou uniformemente distribuídos quando o `equiphi` é utilizado), gera os ângulos baseados

no potencial torcional, com um *bias* que favorece os ângulos próximos aos mínimos da energia torcional. Esse bias é removido no critério de aceitação. Essa palavra chave pode ser útil em casos que a topologia da superfície de energia potencial tenha barreiras que o método CBMC tenha dificuldades de transpor.

- `fudgelj` e `fudgeclb`
  Esses comandos são utilizados para definir a atenuação das interações 1-4 de Lennard-Jones (fudgelj) e Coulomb (fudgeclb). Para o campo de força OPLS esses dois valores devem ser iguais a 0.5 (valores padrão, utilizados quando a palavra chave não é especificada). Para o campo de força AMBER, o valor do fudgeclb deve ser igual a 0.8333 e o do fudgelj 0.5.

- `nsf` e `sfint`
  Indicam o tamanho da biblioteca de fragmentos e (nsf) e o intervalo entre o armazenamento de cada snapshot na simulação do fragmento em fase gasosa que gera a biblioteca de fragmentos (sfint). Isto é, para cada fragmento uma simulação com nsf $\times$ sfint passos é realizada, salvando as configurações na biblioteca de fragmento a cada sfint passos. Tanto nsf quanto sfint devem ser iguais a números inteiros. Por padrão nsf = $10\,000$ (ou o maior valor suportado pela compilação do DICE) e sfint = 500.

- `lambdac`
  Fator de atenuação utilizado no bias eletrostático. Quando um lambdac é definido com um valor entre 0.0 e 1.0, esse fator é aplicado no termo eletrostático da energia no momento em que as configurações são geradas. O bias é removido no critério de aceitação. Por padrão lambdac = 1.0, que significa que o bias eletrostático não é utilizado.

Referências

[1] Henrique M. Cezar, Sylvio Canuto, and Kaline Coutinho. Dice: A monte carlo program for molecular liquid simulation, university of são paulo, v. 3.0, 2018.

[2] Jindal K Shah and Edward J Maginn. A general and efficient Monte Carlo method for sampling intramolecular degrees of freedom of branched and cyclic molecules. *The Journal of Chemical Physics*, 135(13):134121, 2011.

[3] Henrique M. Cezar, Sylvio Canuto, and Kaline Coutinho. Solvent effect on the syn/anti conformational stability: A comparison between conformational bias Monte Carlo and molecular dynamics methods. *International Journal of Quantum Chemistry*, 119(1):e25688, 2019.

[4] John C. Owicki. Optimization of Sampling Algorithms in Monte Carlo Calculations on Fluids. pages 159–171. jun 1978.

[5] Bernard Bigot and William L. Jorgensen. Sampling methods for Monte Carlo simulations of n-butane in dilute solution. *The Journal of Chemical Physics*, 75(4):1944–1952, 1981.