OpenMM, incrOpenMM簡易マニュアル

大阪大学基礎工学研究科　山田一雄

本稿は、分子動力学シミュレーションライブラリOpenMM(<https://openmm.org/>)およびincrOpenMM(<https://github.com/yamada1988/mypythonpkg/tree/master/myOpenMM>) に関する情報を簡単にまとめたものである。OpenMMはVijay Satyanand Pandeらによって制作、管理された分子動力学計算ライブラリであり、PythonやFortran, C++等の言語に対応している。本稿では、主にPythonを用いた分子動力学計算のための情報について記述する。公式のユーザーガイドは<https://openmm.org/documentation>にあるので、そちらも参考にすること。

グラフィカル ユーザー インターフェイス, テキスト

自動的に生成された説明

目次

[インストール方法 3](#_Toc101429592)

[OpenMMスクリプトの例 5](#_Toc101429593)

[Gromacs形式で計算を行うOpenMMスクリプト例 6](#_Toc101429594)

[OpenMMで利用するクラス 7](#_Toc101429595)

[シミュレーションのパラメータ 11](#_Toc101429596)

[incrOpenMMの利用方法 20](#_Toc101429597)

[incrOpenMMのサンプルスクリプト 28](#_Toc101429598)

[ジョブスクリプト例 30](#_Toc101429599)

[incrOpenMM.MDConductorについての説明 32](#_Toc101429600)

[incremental chemical potentialをERmodで計算する方法 34](#_Toc101429601)

[ERmodでreweighting計算を行う方法 42](#_Toc101429602)

[Tips 45](#_Toc101429603)

インストール方法

OpenMMはCondaパッケージマネージャ(https://docs.conda.io)を使ってインストールすることが可能である。Conda は Anaconda Python ディストリビューションの一部として含まれており、https://docs.continuum.io/anaconda/installからダウンロードできる。Anacondaは特に科学的なアプリケーションのために設計されたPythonディストリビューションで、最も一般的な数学的および科学的パッケージの多くがプリインストールされている。

1. Anaconda または Miniconda のどちらかの最新の 64bit,Python 3.x バージョンをインストールする。手順は<https://docs.continuum.io/anaconda/install>を参考にすること。

2.インストールしたAnacondaのパスが通っていることを確認した後、コマンドラインで

conda install -c conda-forge openmm

と入力する。最近のconda(v4.8.4+)では、ドライバがサポートするCUDAの最新バージョンでコンパイルされたバージョンの OpenMM がインストールされる。

3.OpenMMが正常にインストールできているかの確認

次のコマンドを入力する。

python -m simtk.testInstallation

上記コマンドが”No module named simtk.testInstallation”等でエラーとなる場合、代わりに

python -m openmm.testInstallation

を実行する。このコマンドは、OpenMM がインストールされていることを確認し、GPU高速化が利用可能かどうかをチェックし（OpenCL/CUDA プラットフォームが利用可能かをチェックする）、すべてのプラットフォームが一貫した結果を生成することを確認するものである。スパコンでOpenMMをインストールする場合は、ジョブスクリプト内に上のpython -m …コマンドを書いてジョブ投入し、結果をout,errファイルで確認する。

4.（オプション）parmedのインストール

逐次導入法による高分子自由エネルギー計算を行う場合、pythonモジュールparmedをインストールする必要がある。<https://anaconda.org/conda-forge/parmed>にある通り、

conda install -c conda-forge parmed

でインストールを実行する。

5.（オプション）incrOpenMMのインストール

逐次導入法による高分子自由エネルギー計算を行う場合、山田による自作ライブラリincrOpenMMをインストールすると便利である。incrOpenMMは山田のGithubのmypythonpkg内に置いてある。（<https://github.com/yamada1988/mypythonpkg/>）

git clone https://github.com/yamada1988/mypythonpkg.git

で中のincrOpenMMごとmypythonpkg/をダウンロードしてくる。incrOpenMMのPythonPathを通すために、$HOME/.bashrc等の設定ファイルに以下の行を追記してsourceする。

export PYTHONPATH=$PYTHONPATH:mypythonpkgのパス

6.（オプション）MDTRAJのインストール

逐次導入法による高分子自由エネルギー計算を行う場合、pythonモジュールmdtrajをインストールし、内部のプログラムを一部書き換える必要がある。プログラム書き換えのために、ソースからコンパイルしてインストールする必要がある。

git clone https://github.com/mdtraj/mdtraj

mdtraj/format/xtc/xtc.pyxを修正（580行目付近, prec = 1000.0…をprec = 100000.0…に修正）

python setup.py install

$HOME/.bashrc等の設定ファイルに以下の行を追記してPythonPathを通す。

export PYTHONPATH=$PYTHONPATH:mdtrajのパス

7.（オプション）修正ERmodのインストール

逐次導入法を用いた高分子セグメントの溶媒和自由エネルギー計算をERmod（<https://sourceforge.net/p/ermod/wiki/Home/>）を用いて行う場合、ERmodの機能を一部変更した修正ERmod(ERmod\_mod)をインストールする必要がある。ERmod内のソースコード（engproc.F90）の1010行目付近を以下のようにコメントアウトして修正する。

修正前：

|  |
| --- |
| ! only for solute-solvent interaction in solution system  ! larger than the maximum of soft part (linear-graduation part)  if((slttype == SLT\_SOLN) .and. (pti > 0)) then  if(idpti > uvsoft(pti)) then  write(stdout, '(A,g12.4,A,i3,A)') ' energy of ', engval, ' for ', pti, '-th species'  call halt\_with\_error('eng\_sft')  endif  endif |

修正後:

|  |
| --- |
| ! only for solute-solvent interaction in solution system  ! larger than the maximum of soft part (linear-graduation part)  ! if((slttype == SLT\_SOLN) .and. (pti > 0)) then  ! if(idpti > uvsoft(pti)) then  ! write(stdout, '(A,g12.4,A,i3,A)') ' energy of ', engval, ' for ', pti, '-th species'  ! call halt\_with\_error('eng\_sft')  ! endif  ! endif |

以上の修正を行なったのち、通常のERmodのインストール作業を行う。

OpenMMスクリプトの例

OpenMMはpythonスクリプトでMDシミュレーション設定を構築、実行することができるライブラリである。MDシミュレーションのための力場や初期構造のインプットファイルとしては、Gromacs(.gro,.itp,.top)やAmber(.pdb,.xml),LAMMPS,NAMD,CHARMM等、様々なMDシミュレーションパッケージに対応している。以下のように、直感的に分かりやすく簡潔な形でシミュレーションを実行するスクリプトを書く事ができる。

**from** **openmm.app** **import** \*

**from** **openmm** **import** \*

**from** **openmm.unit** **import** \*

**from** **sys** **import** stdout

pdb = PDBFile('input.pdb')

forcefield = ForceField('amber14-all.xml', 'amber14/tip3pfb.xml')

system = forcefield.createSystem(pdb.topology, nonbondedMethod=PME,

nonbondedCutoff=1\*nanometer, constraints=HBonds)

integrator = LangevinMiddleIntegrator(300\*kelvin, 1/picosecond, 0.004\*picoseconds)

simulation = Simulation(pdb.topology, system, integrator)

simulation.context.setPositions(pdb.positions)

simulation.minimizeEnergy()

simulation.reporters.append(PDBReporter('output.pdb', 1000))

simulation.reporters.append(StateDataReporter(stdout, 1000, step=**True**,

potentialEnergy=**True**, temperature=**True**))

simulation.step(10000)

図 1:OpenMMのサンプルスクリプト(sample.py)

上のスクリプトは、初期構造として input.pdb という PDB ファイルを読み込み、Amber14 力場と TIP3P-FB 水モデルのパラメータを読み込んで力場パラメータを設定し、長距離相互作用をPME、非結合相互作用のカットオフ長を1 nm、H分子との結合距離の拘束化を行い、エネルギー最小化を行ったのち、 Langevin integratorを用いて温度300 Kの条件で10,000 ステップシミュレーションを行い、1000 ステップごとに output.pdb という PDB ファイルにスナップショットフレームを保存するスクリプトである。各行の詳細な解説は以下のURLに載っている。<http://docs.openmm.org/latest/userguide/application/02_running_sims.html>

上記スクリプト(sample.py)を実行する場合、

python sample.py

でMDシミュレーションを実行することができる。

Gromacs形式で計算を行うOpenMMスクリプト例

ここでは、Gromacsのインプットファイル(.gro, .top)を読み込んでシミュレーションを行うスクリプト例を紹介する。前頁と同じく、直感的にスクリプトを書くことが可能である。

**from** **openmm.app** **import** \*

**from** **openmm** **import** \*

**from** **openmm.unit** **import** \*

**from** **sys** **import** stdout

gro = GromacsGroFile('input.gro')

top = GromacsTopFile('input.top', periodicBoxVectors=gro.getPeriodicBoxVectors(),

includeDir='/usr/local/gromacs/share/gromacs/top')

system = top.createSystem(nonbondedMethod=PME, nonbondedCutoff=1\*nanometer,

constraints=HBonds)

integrator = LangevinMiddleIntegrator(300\*kelvin, 1/picosecond, 0.004\*picoseconds)

simulation = Simulation(top.topology, system, integrator)

simulation.context.setPositions(gro.positions)

simulation.minimizeEnergy()

simulation.reporters.append(PDBReporter('output.pdb', 1000))

simulation.reporters.append(StateDataReporter(stdout, 1000, step=**True**,

potentialEnergy=**True**, temperature=**True**))

simulation.step(10000)

図 2:Gromacs形式でMDシミュレーションを行うスクリプト。

上のスクリプトは、初期構造として input.groという GROファイルを読み込み、力場としてinput.topに記述されたパラメータを読み込んで力場パラメータを設定し、長距離相互作用をPME、非結合相互作用のカットオフ長を1nm、H分子との結合距離の拘束化を行い、エネルギー最小化を行ったのち、 Langevin integratorを用いて温度300Kの条件で10,000 ステップシミュレーションを行い、1000 ステップごとに output.pdb という PDB ファイルにスナップショットフレームを保存するスクリプトである。詳細は以下のURLを参照すること。[http://docs.openmm.org/latest/userguide/application/02\_running\_sims.html#using-gromacs-files](http://docs.openmm.org/latest/userguide/application/02_running_sims.html" \l "using-gromacs-files)

OpenMMで利用するクラス

OpenMMをpythonスクリプト形式で利用する場合、シミュレーションのための幾つかのクラスが用意されている。クラスとは、オブジェクト指向プログラミング言語であるPythonにおいて、データを入れる領域、データの処理の仕方等をまとめたオブジェクト（設計図）のことを指す。臨機応変にこのクラスを設計できるのがPythonの強みの一つである。OpenMMでは、シミュレーションを実行するためのパラメータ群や、シミュレーション処理などをひとまとめにしたクラスであるSimulationが一番大きなクラスとして存在している。

Simulation

Simulationクラスは、OpenMMでシミュレーションを実行し、結果を報告するための簡潔なAPIを提供する。これは、Topology、System、Integrator、Contextといった、シミュレーションを実行するために必要なオブジェクト同士を結び付けるものである。また、Simulationでは、MD計算で得られた各原子の座標をファイルに書き出すなど、シミュレーションの実行中にデータを記録したり解析したりするReporterオブジェクトのリストも管理している。例えば、次のように記述すると、"output.pdb "というファイルが作成され、1000タイムステップごとに構造が書き込まれる。

simulation.reporters.append(PDBReporter(‘output.pdb’, 1000))

より詳細な情報は[http://docs.openmm.org/latest/api-python/generated/openmm.app.simulation.Simulation.html#openmm.app.simulation.Simulation](http://docs.openmm.org/latest/api-python/generated/openmm.app.simulation.Simulation.html" \l "openmm.app.simulation.Simulation)

を参照すること。

Topology

Topologyは、系のトポロジー情報を格納するクラスである。Topologyオブジェクトの構造はPDBファイルの構造に似ており、どの原子のペアが互いに結合しているかというリストと、結晶学的単位セルの寸法が格納されている。topologyクラスは

pdb=PDBFile(‘input.pdb’)

topology = pdb.topology

や

top=GromacsTopFile(‘input.top’)

topology = top.topology

などで得ることができる。より詳細な情報は[http://docs.openmm.org/latest/api-python/generated/openmm.app.topology.Topology.html#openmm.app.topology.Topology](http://docs.openmm.org/latest/api-python/generated/openmm.app.topology.Topology.html" \l "openmm.app.topology.Topology)を参照すること。

System

Systemは系の情報を表すクラスであり、Particle,Constraint,Force,PeriodicBoxVectorsの4つの要素がある。ParticleとConstraintはSystemオブジェクトによって直接定義され、ForceはOpenMMにある元のForceクラスを拡張したオブジェクトによって定義される。System を作成した後、各粒子ごとに addParticle()、各Constraintごとに addConstraint()、各 Force に対して addForce() を呼び出す。また、仮想サイトをParticleとして指定することができる。これは、tip4pモデルのように他の粒子の位置から自動的に位置が決まる相互作用点である。setVirtualSite()を呼び、その位置の計算規則を定義したVirtualSiteオブジェクトを渡すことで仮想サイトを定義することができる。より詳細な情報は[http://docs.openmm.org/latest/api-python/generated/openmm.openmm.System.html#openmm.openmm.System](http://docs.openmm.org/latest/api-python/generated/openmm.openmm.System.html" \l "openmm.openmm.System)を参考にすること。

Context

Contextには、シミュレーションの状態が全て保存されている。具体的には、

最新の時刻

各粒子の位置

各粒子の速度

System内のForceオブジェクトで定義された、設定可能なパラメーターの値

が保持されている。Contextオブジェクトの関数getState() を呼び出すことで、いつでも現在の状態のスナップショットを取得することができる。getState()は、各粒子にかかる現在の力とシステムの現在のエネルギーを取得するために使用することもできる。より詳細な情報は[http://docs.openmm.org/latest/api-python/generated/openmm.openmm.Context.html#openmm.openmm.Context](http://docs.openmm.org/latest/api-python/generated/openmm.openmm.Context.html" \l "openmm.openmm.Context)を参照すること。

State

Stateオブジェクトは、現在のシミュレーションの状態のスナップショットが記録されている。Contextに対してgetState()を呼び出すことで作成する。

state = Context.getState(getPositions=True)

上記のように、状態を作成する際にどのような情報を保存するかを指定することで、必要な情報のみを取り出すことができるため、時間とメモリを節約することができる。上の例では最新の粒子の位置情報がStateオブジェクトに保存される。他にも速度、力、エネルギー、力場パラメータ、系のシステムサイズ等を保存することができる。より詳細な情報は[http://docs.openmm.org/latest/api-python/generated/openmm.openmm.State.html#openmm.openmm.State](http://docs.openmm.org/latest/api-python/generated/openmm.openmm.State.html" \l "openmm.openmm.State)を参照すること。

Force

Forceオブジェクトは、系の粒子に新たな力を加えたり、挙動を変化させたりする。Forceは抽象クラスであり、そのサブクラスで具体的な力を定義する。

具体的には、Forceオブジェクトは以下のいずれか、またはすべてを行うことができる。

各粒子にかかる力への寄与を追加する

系の位置エネルギーへの寄与を追加する。

各タイムステップ開始時の粒子の位置と速度を変更する。

Contextに保存されている、ユーザーが変更可能なパラメータを定義する。

各タイムステップの開始時に、他のForceオブジェクトで定義されたパラメータの値を変更する。より詳細な情報は[http://docs.openmm.org/latest/api-python/generated/openmm.openmm.Force.html#openmm.openmm.Force](http://docs.openmm.org/latest/api-python/generated/openmm.openmm.Force.html" \l "openmm.openmm.Force)を参照すること。

NonbondedForce

NonbondedForceクラスは、クーロン力とファンデルワールス力という粒子間の非結合相互作用を実装したクラスである。オプションとして、周期境界条件と長距離相互作用のカットオフをサポートしている。Lennard-Jones相互作用はLorentz-Berthelot結合則で計算される。（2つの相互作用する粒子のシグマの算術平均とイプシロンの幾何平均を使用する。）

このクラスを使用するには、NonbondedForce オブジェクトを作成し、系の各粒子に対して一度 addParticle() を呼び出し、非結合パラメータ()を定義する必要がある。非結合パラメータを定義する粒子数は、系内の粒子数と正確に等しくなければならず、間違いがある場合、Context を作成しようとしたときにExceptionが呼び出される。粒子を追加した後、setParticleParameters()を呼び出すと、各粒子の非結合パラメータを変更することができる。ただし、すでに存在する Context に対しては updateParametersInContext() を呼び出さない限りパラメータが上書きされることはない。

NonbondedForce では、Exceptionを用いて粒子間相互作用の「例外」を指定することもできる。これは、個々の粒子に対して定義された相互作用パラメータとは異なるパラメータに基づいて相互作用を計算すべき、粒子間の特定のペアを指す。これにより、ある種の相互作用を力計算から完全に除外したり、相互作用の方法を変更したりすることができる。

この機能は主に分子内の1-4相互作用などでよく用いられる。

NonbondedForceでcutoffを使用する場合、デフォルトではLennard-Jones相互作用はカットオフ距離で不連続的に0となる。オプションとして、有限の距離範囲でスムーズに相互作用がゼロになるように、スイッチング関数を使用することができる。これを有効にするには、setUseSwitchingFunction()を呼び出し、setSwitchingDistance() を呼び出して、相互作用が減少し始める距離を指定する必要がある。切り替え距離は、カットオフ距離より小さくなければならない。（Gromacs他と同様。）

NonbondedForceクラスのもう一つのオプション機能は、周期的なシステムでカットオフを超えるすべてのLennard-Jones相互作用の効果を近似するエネルギーへの寄与を追加することである。圧力一定の条件でシミュレーションをおこなう場合、これは結果の質を向上させることができる。この機能はsetUseDispersionCorrection()を呼び出して使用するかどうかを設定する。（デフォルトで有効になっている。）

より詳細な情報は[http://docs.openmm.org/latest/api-python/generated/openmm.openmm.NonbondedForce.html#openmm.openmm.NonbondedForce](http://docs.openmm.org/latest/api-python/generated/openmm.openmm.NonbondedForce.html" \l "openmm.openmm.NonbondedForce)

を参照すること。

GromacsGroFile

GromacsGroFile は、Gromacs形式の.groファイルを読み込み、原子位置の情報を保持するものである。元の.groファイルには元素名や残基名などのトポロジー情報も含まれているが、完全なTopologyオブジェクトを構築するには（力場パラメータ等の情報がないため）十分ではない。この情報はオブジェクトのパブリックフィールドに記録・保存される。より詳細は[http://docs.openmm.org/latest/api-python/generated/openmm.app.gromacsgrofile.GromacsGroFile.html#openmm.app.gromacsgrofile.GromacsGroFile](http://docs.openmm.org/latest/api-python/generated/openmm.app.gromacsgrofile.GromacsGroFile.html" \l "openmm.app.gromacsgrofile.GromacsGroFile)を参照すること。

GromacsTopFile

GromacsTopFile は、Gromacs形式のtopファイルを解析し、そこからTopologyオブジェクトと (オプションで) Systemオブジェクトを構築するものである。Systemオブジェクトを構築する場合、createSystem()関数を用いる。これは、非結合相互作用の詳細や拘束条件等を設定することで、系の情報を保持するSystemオブジェクトを作成する関数である。より詳細は[http://docs.openmm.org/latest/api-python/generated/openmm.app.gromacstopfile.GromacsTopFile.html#openmm.app.gromacstopfile.GromacsTopFile](http://docs.openmm.org/latest/api-python/generated/openmm.app.gromacstopfile.GromacsTopFile.html" \l "openmm.app.gromacstopfile.GromacsTopFile)を参照すること。

シミュレーションのパラメータ

sample.pyのように、OpenMMでは用意されたパラメータを適宜変更することでシミュレーション条件をユーザーが好きなようにカスタマイズすることができる。詳細は公式ページのURLに記載されているが、ここではよく使うと思われるパラメータに絞って解説することにする。[http://docs.openmm.org/latest/userguide/application/02\_running\_sims.html#simulation-parameters](http://docs.openmm.org/latest/userguide/application/02_running_sims.html" \l "simulation-parameters)

Platforms

Simulationオブジェクトを作成する際に、オプションで使用するプラットフォームを指定することができる。OpenMMにはRefference、CPU、CUDA、OpenCLの4つのプラットフォームがある。プラットフォームを選択する方法は3つある。

1. デフォルトでは、OpenMM は利用可能な最も高速なプラットフォームを選択しようとする。通常、これが最も合理的である。

2. 明示的に設定したい場合、環境変数OPENMM\_DEFAULT\_PLATFORMに使用したいプラットフォームの名前を設定することも可能である。これはデフォルトのロジックをオーバーライドする。

3. 最後に、Simulation作成時にスクリプト上でプラットフォームオブジェクトを明示的に指定することができる。以下の行は、CUDAプラットフォームを使用するように指定している。

platform = Platform.getPlatformByName('CUDA')

simulation = Simulation(top.topology, system, integrator, platform)

platform名はRefference,CPU,OpenCL,CUDAで指定する。

さらに、計算方法をカスタマイズするためのプラットフォーム固有のプロパティを指定することができる。たとえば、以下は、2 つの異なる GPU（CUDA デバイス 0 および 1）で計算を並列化し、すべての計算を倍精度で行うように指定している。

platform = Platform.getPlatformByName('CUDA')

properties = {'DeviceIndex': '0,1', 'Precision': 'double'}

simulation = Simulation(prmtop.topology, system, integrator, platform, properties)

より詳細な情報は以下のURLの「3.6.1　Platforms」の記述を参考にすること。[http://docs.openmm.org/latest/userguide/application/02\_running\_sims.html#simulation-parameters](http://docs.openmm.org/latest/userguide/application/02_running_sims.html" \l "simulation-parameters)

Force Fields

XML形式で力場を作成する際には、力場定義の読み込み元となるXMLファイルを1つ以上指定する。また、Gromacsの.top形式で力場を作成する場合は、GromacsTopFile()を使って.topファイルを指定する。いずれの場合も、メインの力場を定義するファイルが1つあり、場合によっては水や具体的な分子モデルを定義する2つ目のファイル(.itp)がある。

より詳細な情報はPlatformsの欄にあるURLの「3.6.2 Force Fields」を参照すること。

Nonbonded Interactions

力場（またはprmtopファイル）からSystemオブジェクトを作成するときに、Nonbonded Interactionsの扱いに関するオプションを指定することができる。

system=top.createSystem(nonbondedMethod=PME,nonbondedCutoff=1\*nanometer)

NonbondedMethodについては、主に以下のパラメータを設定することができる。

| **値** | **意味** |
| --- | --- |
| NoCutoff | cutoff処理を行わない。全てのペアについて力を計算する。 |
| Ewald | 周期的境界条件が適用された系で指定できる。クーロン相互作用の長距離成分の計算に Ewald Summation を使用する。(小さい系を除いては 下のPME の方がはるかに速いため、このオプションはほとんど使用されない。 |
| PME | 周期的境界条件が適用された系で指定できる。クーロン相互作用の長距離成分の計算に Particle Mesh Ewald法を使用する。主にこれを利用する。 |
| LJPME | 周期的境界条件が適用された系で指定できる。Lennard-Jones相互作用の長距離成分の計算に Particle Mesh Ewald法を使用する。 |

NoCutoffオプション以外を利用する場合、以下のようにカットオフ長をnonbondedCutoffオプションで指定する必要がある。

nonbondedCutoff=1.2\*nanometers

NonbondedMethodにEwald、PME、LJPMEを使用する場合、オプションで力計算の誤差許容範囲を指定することができる。例えば

system=top.createSystem(nonbondedMethod=PME,nonbondedCutoff=1\*nanometer,ewaldErrorTolerance=0.00001)

誤差の許容範囲は、Ewaldの総和を切り捨てることによる力の分数誤差にほぼ等しくなっている。指定しない場合は、デフォルト値の0.0005が使用される。

cutoffを使用する際のもう一つのオプションパラメータはswitchDistanceである。これにより、非結合相互作用はカットオフ距離で急激に切り捨てられるのではなく、ある有限の範囲にわたってスムーズにゼロになるようになる。これによって、計算中の系のエネルギー保存を改善することができる。これを使うには、相互作用が減少し始める距離を指定する。例えば

system=top.createSystem(nonbondedMethod=PME, nonbondedCutoff=1.2\*nanometer

switchDistance=1.0\*nanometer)

とすると、1 nmからスイッチング関数に従って非結合相互作用がスムーズに減衰していく。

スイッチング関数の具体的な関数やNonbondedFOrceの説明は以下のURLを参考にすること。[http://docs.openmm.org/latest/userguide/theory/02\_standard\_forces.html#nonbondedforce](http://docs.openmm.org/latest/userguide/theory/02_standard_forces.html" \l "nonbondedforce)また、NonbondedInteractionsに関するより詳細な情報は以下のURLを参照すること。[http://docs.openmm.org/latest/userguide/application/02\_running\_sims.html#simulation-parameters](http://docs.openmm.org/latest/userguide/application/02_running_sims.html" \l "simulation-parameters)

Constraints

Systemオブジェクトを作成する際、NonbondedMethodと同様、オプションとして系の原子間結合の距離や角度に拘束条件を加えることが可能である。

system = top.createSystem(nonbondedMethod=NoCutoff, constraints=HBonds)

拘束条件のパラメータは以下のものがある。

| **値** | **意味** |
| --- | --- |
| None | デフォルト値。なんの拘束もかかっていない。 |
| HBonds | 水素原子が関与する原子間結合の距離を固定する。 |
| AllBonds | 全ての結合距離を固定する。 |
| HAngles | すべての結合距離が拘束される。さらに、H-X-H または H-O-X (X は任意の原子) 形式のすべての角度が拘束される。 |

Constraintsを使用する主な理由は、計算時により大きな時間刻みを使用できるようになることである。拘束条件がない場合、AMBERやCHARMMのような典型的な生体分子力場では、時間刻みは通常約1fsのタイムステップに制限される。HBonds拘束条件を使用すると、Verletダイナミクスでは約2fs、Langevinダイナミクスでは約4fsまで時間刻みを増加させることができます。HAnglesでは、さらに大きくできる場合がある。このパラメータの値に関係なく、OpenMM では水分子は完全に剛体化され、結合長と角度の両方に拘束条件がかかる。(例えばtip3pモデルなど。)rigidWater パラメータでこの動作を無効にすることができる。より詳細な情報は以下のURLの「3.6.7 Constraints」の部分を参照すること。[http://docs.openmm.org/latest/userguide/application/02\_running\_sims.html#simulation-parameters](http://docs.openmm.org/latest/userguide/application/02_running_sims.html" \l "simulation-parameters)

Integrators

OpenMMでは、MD計算のためのIntegratorがいくつか存在し、その中から一つを選択することができる。すべてのIntegratorの詳細な説明は以下のURLに記載されているため、そちらを参照すること。<http://docs.openmm.org/latest/userguide/theory/04_integrators.html>

よく用いられるIntegratorを以下で説明する。

Langevin Middle Integrator

integrator = LangevinMiddleIntegrator(300\*kelvin, 1/picoseconds, 0.004\*picoseconds)

上の行の3つのパラメータ値は、シミュレーション温度（300 K）、摩擦係数（1 ps-1）、時間刻み（0.004 ps）をそれぞれ表しており、これらの値はユーザーが自由に変更することができる。ただし、すべての値で単位を指定する必要がある。例えば、時間刻みの値は0.004\*picosecondsまたは4\*femtosecondsのように書くことができ、これらはどちらも全く同じ意味を表す。LangevinMiddleIntegratorはleap-frog法で計算する手法なので、速度は位置から半時間ステップ分オフセットされていることに注意すること。

Langevin Inregrator

LangevinIntegrator は LangevinMiddleIntegrator と非常に似ているが、Langevin 方程式の離散化が異なる。LangevinMiddleIntegratorの方がより正確なサンプリングを行う傾向があるため、ほとんどのアプリケーションでそちらの方が好んで使用されている。また、LangevinIntegratorはLangevinMiddleIntegratorと同様にleap-frog法で計算されるため、速度は位置から半時間ステップ分オフセットされていることに注意すること。

Nosé-Hoover Integrator

NoseHooverIntegrator は、LangevinMiddleIntegrator と同じようなアルゴリズムを使用しているが、Langevin熱浴による確率的温度制御を、より正確な輸送特性を生み出す速度スケーリングアルゴリズムに置き換えている（Nose-Hoover熱浴）。この速度スケーリングは、各ステップにおける熱浴変数の計算が必要なため、LangevinMiddleIntegratorに対して効率が若干低下する。

integrator=NoseHooverIntegrator(300\*kelvin, 1/picosecond,0.004\*picoseconds)

第1引数で目標温度を指定し、第2引数で熱浴との相互作用のfrequencyを指定する。低い値の場合、系と熱浴の相互作用が小さくなり、周波数ゼロの極限でmicro canonicalアンサンブルに一致する。一方、大きなfrequencyは系とNose-Hoover熱浴の強いカップリングをもたらし、系の温度を目標温度により近づけようとする事になる。第3引数は時間刻み幅で、他の引数と同様に単位を指定する必要がある。

LeapfrogVerletIntegrator

LeapfrogVerletIntegratorは、エネルギー一定条件(NVE)ダイナミクスの実行に使用される。

integrator = VerletIntegrator(0.002\*picoseconds)

のように指定される。このIntegratorのオプションは時間刻み幅のみである。

これ以外にも様々なIntegratorが存在する。名前や機能は以下のURLを参考にすること。

<http://docs.openmm.org/latest/userguide/application/02_running_sims.html#simulation-parameters>

また、それぞれのIntegratorの理論的な背景は以下のURLを参考にすること。

<http://docs.openmm.org/latest/userguide/theory/04_integrators.html>

Temperature Coupling

NVTアンサンブルやNPTアンサンブルなど、系の温度を一定に保ってシミュレーションを行いたい場合、LangevinIntegratorを使用するのが最適である。これ以外のIntegratorを用いる場合、例えばVerletIntegratorでは、Andersen熱浴を系と接触させて温度カップリングを提供することができる。これを行うには、以下のようにaddForce()を呼んでAndersenThermostatオブジェクトをSystemに追加する。

system = top.createSystem(nonbondedMethod=PME, nonbondedCutoff=1\*nanometer,constraints=HBonds)

system.addForce(AndersenThermostat(300\*kelvin, 1/picosecond))

integrator = VerletIntegrator(0.002\*picoseconds)

AndersenThermostatのパラメータは2つで、第1引数が熱浴の温度、第2引数がfrequencyである。

Pressure Coupling

NPTアンサンブル計算のように、系の圧力を一定に保ってシミュレーションを行いたい場合、OpenMMではMonteCarkoBarostatをSystemに追加する必要がある。これは、上のTemperature CouplingでAndrsenThermostatを追加したのと全く同じ方法で行うことができる。

system=top.createSystem(nonbondedMethod=PME,nonbondedCutoff=1\*nanometer,Constraints=HBonds)

system.addForce(MonteCarloBarostat(1\*bar, 300\*kelvin))

integrator = LangevinMiddleIntegrator(300\*kelvin, 1/picoseconds, 0.004\*picoseconds)

MonteCarkoBarostatのパラメータは、圧力（1 bar）と温度（300 K）である。圧力浴はシミュレーションが一定の温度で実行されていることを前提にしていルガ、それ自身は温度を調節するようなことはしないため、熱浴は別途指定する必要がある。上の例ではLangevinIntegratorを用いることでLangevin熱浴を系とカップリングさせている。注意点として、MonteCarloBarostatとLangevinIntegratorやAndersenThermostatで指定する温度パラメータは互いに一致していなければならない。一致していない場合、間違った計算結果が得られてしまう点に留意すること。また、OpenMMでは非等方圧力浴による圧力制御も可能であり、それらを含むより詳細な情報は以下のURLの3.6.10を参考にすること。

<http://docs.openmm.org/latest/userguide/application/02_running_sims.html#simulation-parameters>

Energy Minimization

sample.pyでわかる通り、局所エネルギー最小化の実行は以下の1行で済む。

simulation.minimizeEnergy()

ほとんどの場合これだけで十分だが、最小化をさらに制御したい場合は、2つのオプションのパラメータを指定することができる。1つ目はエネルギーが収束したと見なす際の許容誤差であり、

simulation.minimizeEnergy(tolerance=5\*kilojoule/mole)

のようにする。このパラメータを指定しない場合、デフォルトの許容誤差10 kJ/moleが使用される。2つ目は最大反復回数であり、

simulation.minimizeEnergy(maxIterations=100)

のようにする。エネルギーがまだ収束していなくても、指定された反復回数に達すると、エネルギー最小化計算は終了する。このパラメータを指定しない場合、最小化計算は収束に達するまで継続する。これらのオプションは独立したものであり、必要であれば、両方を指定することができる。

simulation.minimizeEnergy(tolerance=0.1\*kilojoule/mole,maxIterations=500)

Removing COM motion

OpenMMのデフォルトでは、作成されたSystemオブジェクトには、各時間ステップごとにすべての重心(COM)運動を除去する CMMotionRemover が入っており、系全体が時間と共にドリフトしないようにしている。これはほとんどの場合必要な処理であるが、まれに、系が時間とともにドリフトすることを許容する場合がある。その場合、Systemオブジェクトの作成時にremoveCMMotionパラメータを指定することで実現できる。

system = forcefield.createSystem(pdb.topology, nonbondedMethod=NoCutoff,

removeCMMotion=False)

Writing Trajectory

OpenMMでは、シミュレーションのtrajectoryをいくつかのフォーマットで保存することができる。PDB,PDBx/mmCIF,DCD,また、mdtrajをimportすることでXTCやTRR形式で保存可能となる。書き出されtrajectoryは世間で用いられているほトンドの解析・可視化プログラムで読み込むことができる。trajectoryの保存には、以下のスクリプト例に示すように、Simulationに "reporter "を追加するだけである。

simulation.reporters.append(PDBReporter('output.pdb', 1000))

PDBReporterの2つのパラメータは、出力ファイル名と出力構造を書き込む頻度（タイムステップ数）である。PDBx/mmCIFやDCD形式を使うには、PDBReporterをPDBxReporterやDCDReporterに置き換えるだけであり、その際のパラメータはPDBReporterの場合と同じである。

simulation.reporters.append(DCDReporter('output.dcd', 1000))

XTC形式やTRR形式で保存したい場合、山田の自作ライブラリであるincrOpenMM（事前にinstallする必要あり）をimportしてXTCReporterやTRRReporterを利用可能にした後でSimulationにreporterを追加する。

import incrOpenMM as mm

simulation.reporters.append(mm.XTCReporter(‘output.xtc’, 1000))

PDBReporter等のOpenMM組み込みオブジェクトについてより詳細な情報は以下のURLを参照すること。<http://docs.openmm.org/latest/api-python/search.html?q=reporter>

Recording Data

系のtrajectoryだけでなく、系のポテンシャルエネルギーや温度など、シミュレーションの過程で他の物理量を保存したい場合がある。OpenMMでは、このような場合にも対応できるように、StateDataReporterオブジェクトを用意している。以下のようにStateDataReporterを作成し、Simulationに追加する。

simulation.rpporters.append(StateDataReporter('data.csv', 1000, time=True,kineticEnergy=True,

potentialEnergy=True))

最初の2つのパラメータは、出力ファイル名と値を書き込む頻度（時間ステップ数）である。拡張子名は任意の名前が利用可能であり、テキストファイル形式で保存される。残りの引数でどのような値を書き込むかを指定する。使用可能なオプションはstep（現在のタイムステップのインデックス）、time、progress（シミュレーションの何パーセントが完了したか）、remainingTime（シミュレーションが完了するまでの推定時間）、potentialEnergy、kineticEnergy、totalEnergy、temperature、volume（周期的ボックスの体積）、density（システム全体の質量を周期的ボックスの体積で割ったもの）、speed（シミュレーションがどのくらい速く動いているかの推定）である。progressまたはremainingTimeオプションを指定する場合、totalStepsパラメータも同時に指定し、シミュレーションに含まれる時間ステップの総数を指定する必要がある。要求された値を1行につなげたテキストが逐次ファイルに書き込まれていく。デフォルトでは、値はカンマ区切り値（CSV）フォーマットで書き込まれる。separatorパラメータを使用すると、別のセパレータを選択することができる。たとえば、以下のようにすると値がカンマではなくスペースで区切られます。

simulation.reporters.append(StateDataReporter('data.txt', 1000, progress=True,

temperature=True, totalSteps=10000, separator=' '))

Saving Simulation Progress and Results

OpenMM には、シミュレーションの結果を保存する方法が 3 つ組み込まれている。（その他にも、結果を保存するスクリプトを自分で書いたり、mdtraj や parmed などの他のパッケージをimportして書き出すことも可能である）。単に最終構造を保存したいだけなら、PDBFile.writeFile()を使ってPDBファイルとして書き出すことができる。もし、位置以外の情報も保存したいのであれば、simulation.saveState()を使用することができる。これは、位置、速度、系の大きさなど、シミュレーションの全状態をXMLファイルに保存する関数である。このファイルは、再度OpenMMでロードして、シミュレーションを継続することが可能である。重要なのは、このファイルがテキストファイルであるため、異なるプラットフォームや異なるバージョンのOpenMM間で転送できることである。この方法の欠点は、Stateファイルが非常に大きくなりがちで、すべてのユースケースに対応できない可能性があることである。以下に、アウトプットの仕方や再度読み込む方法を記す。

simulation.saveState('output.xml')

シミュレーションを再び読み込むには

simulation.loadState('output.xml')

とする。シミュレーションを保存するもう一つ別の方法として、チェックポイントファイルを書き出すというものがある。これはシミュレーション全体をバイナリファイルとして保存するものであり、これを用いることで、必要なときにシミュレーションを正確に継続することができる。（シミュレーションが決定論的であるかどうかは、Platformや手法に依存する）。ただし、この方法には重要な注意点がある。書き出したバイナリファイルは、保存したマシンと同じハードウェア、同じOpenMMのバージョンでのみ正確にシミュレーションを再開することができる。したがって、それが問題にならないことが明らかな場合にのみ、使用する必要があるチェックポイントファイルの書き出しにはsaveCheckpoint()を用いる。

simulation.saveCheckpoint('state.chk')

そして、このように再び読み込むことができる。

simulation.loadCheckpoint('state.chk')

また、OpenMM にはチェックポイントを保存するための CheckpointReporter が組み込まれており、予期せぬ事態や外部要因（サーバクラッシュなど）で失敗したシミュレーションを再開する際に役に立つ。5,000ステップごとにチェックポイントファイルを保存するには、例えば、以下のようにする。

simulation.reporters.append(CheckpointReporter('checkpnt.chk', 5000))

チェックポイントレポーターは、最後のチェックポイントファイルを上書きすることに注意すること。

incrOpenMMの利用方法

incrOpenMMは、Gromacs形式のファイルをOpenMMでシミュレーションを実行するために作成されたパッケージである。MD計算条件を記したインプットファイル(.inp)と初期構造(.gro)、力場ファイル(.top, .itp)を用意するだけで簡素化されたスクリプトを投げてシミュレーションを実行することができる。さらに、高分子の自由エネルギー計算を高速に行う手法である逐次導入法(chain-increment method)に必要な平衡状態サンプリングのためのMD計算(ghost状態計算)を行うこともできるという特徴を持っている。利用には本稿「インストール方法」の1.から7.を全て行い、必要なパッケージをインストールする必要がある。

input.inpファイル

上で解説した、OpenMMでシミュレーションを行うためのパラメータ群を1つにまとめたテキストファイルであり、Gromacsにおける.mdpファイルと同じ役割を果たす。以下に.inpファイルの例を示す。incrOpenMMを用いる際、通常のMD計算であれば、input.inpファイルだけ作成しておけば、このファイルを読み込むことでシミュレーションに必要なパラメータを自動で設定してシミュレーションを実行するスクリプト(sample.py)が用意されているため、非常に簡単にMD計算を行うことが可能である。#で始まる行はコメント行であり、見やすさのために書かれている。

#OpenMM Input Parameters

integrator = Langevin

dt = 0.0010

friction\_const= 1.0e0

mdensemble = npt

temperature= 483.0e0

pressure = 1.0e0

pbc = True

constraints = AllBonds

sysfile = system.gro

forcefield = topol.top

fileformat = GRO

# Equilibriation Process

nptflag = True

nptstep = 250000

# Record Input

nptrecflag = True

npteqrecstep= 25000

verbose = True

forcerecflag= False

reporterformat= XTC

# NonBonded Method

nonbondedmethod = PME

nonbobded\_cutoffflag = True

nonbonded\_cutoff = 1.20e0

nonbonded\_switchflag = True

nonbonded\_switch = 1.00e0

# PME Parameters

setPME = True

pme-alpha = 2.6

pme-nx = 64

pme-ny = 64

pme-nz = 64

pme-ftol = 5.0e-4

# Advanced Simulation

ghost\_particle = True

path\_ndxfile = system.ndx

remdflag = False

annealingflag= False

heavyhydrogen = False

removeCMMotion = False

# Platform Input

platform = OpenCL

precision = single

図3: input.inpの例。OpenMM計算のためのインプットパラメータ群が1つにまとまっている。

以下にパラメータの説明と指定可能な値を羅列して紹介する。

#OpenMM Input Parameters

integrator

運動方程式の数値計算アルゴリズムを指定する。利用可能な値はLangevin,Brownian,Verlet。

それぞれのIntegratorの説明は本稿「シミュレーションのパラメータ」のIntegratorsの部分を参照すること。

dt

数値計算の時間刻み幅を指定する。（単位:nano second）実数で指定する。

friction\_constant

integratorにLangevinを指定した場合に有効となる。Langevin方程式における抵抗係数であり、damping factorとも呼ばれ、熱浴との相互作用の強さと言い換えることもできる。（単位:）実数で指定する。

mdensemble

MD計算を行うアンサンブルを指定する。利用可能な値はNPT,NVT,NVE。上記のintegratorや後述のnptflag,Pressure等の値と矛盾のないようにする必要がある。値の矛盾がある場合でもシミュレーション自体は動いてしまうため、正しい計算結果が得られなくなってしまうことに注意。

temperature

系と接する熱浴の温度を実数で指定する。（単位はK）NPT, NVTアンサンブル計算で必須。デフォルト値は300 K。

pressure

系と接する圧力浴の圧力を実数で指定する。（単位はatm）NPTアンサンブル計算で必須。デフォルト値は1 atm。

pbc

系に周期境界条件を課すかどうかをTrue/Falseで指定する。デフォルト値はTrue。

constraints

原子間の拘束条件を文字列で指定する。利用可能な値はAllBonds,Hbonds,None。デフォルト値はNone。

sysfile

系の初期構造ファイル名を文字列で指定する。input.groのように拡張子も明記する必要がある。利用可能な拡張子名は.groおよび.pdbであり、これは後述のfileformatと一致している必要がある。デフォルト値はsystem.gro。

forcefield

系の力場情報が書かれたファイル名を文字列で指定する。topol.topのように拡張子も明記する必要がある。現在利用可能な拡張子名は.topのみ。デフォルト値はtopol.top。

fileformat

系の初期構造および計算終了後に書き出される構造データのファイルフォーマット（拡張子名）を文字列で指定する。利用可能な値はGROおよびPDBであり、それぞれ拡張子名.groと.pdbに対応している。デフォルト値はNoneであり、必ずGROかPDBのどちらかを指定する必要がある。

#Equilibriation Process

nptflag

MD計算をNPTアンサンブルで行うかどうかをTrue/Falseで指定する。デフォルト値はFalse。mdensembleと重複している。

nvtflag

MD計算をNPTアンサンブルで行うかどうかをTrue/Falseで指定する。デフォルト値はFalse。mdensembleと重複している。

nptstep

NPTアンサンブル計算の総ステップ数を整数で指定する。nptflagがTrueの時のみ有効となる。デフォルト値は0。

nvtstep

NVTアンサンブル計算の総ステップ数を整数で指定する。nvtflagがTrueの時のみ有効となる。デフォルト値は0。

nptreqflag

NPTアンサンブルのMD計算で得られたトラジェクトリを保存するかどうかをTrue/Falseで指定する。mdensembleでNPTを指定した場合に有効である。デフォルト値はFalse。

nvtreqflag

NVTアンサンブルのMD計算で得られたトラジェクトリを保存するかどうかをTrue/Falseで指定する。mdensembleでNVTを指定した場合に有効である。デフォルト値はFalse。

npteqreqstep

NPTアンサンブルのMD計算中にトラジェクトリを1回書き出す間のステップ間隔を整数で指定する。上記サンプルでは25000ステップに1回トラジェクトリが書き出される。nptreqflagがTrueの時のみ有効。デフォルト値は0。

nvteqreqstep

NVTアンサンブルのMD計算中にトラジェクトリを1回書き出す間のステップ間隔を整数で指定する。nvtreqflagがTrueの時のみ有効。デフォルト値は0。

verbose

現状意味のないパラメータ。.logファイルを書き出すか書き出さないかを決めるつもりだったもの。現在のデフォルトでは.logファイルが自動で書き出されるようになっている。

forcereqflag

現状意味のないパラメータ。MD計算中の各原子に働く力をnxteqreqstepごとに別のファイルに書き出すかどうかを決めるつもりだったもの。現在のデフォルトではFalseとなっている。

reporterformat

トラジェクトリのフォーマットを文字列で指定する。利用可能なフォーマットはXTC,TRR,HDF5であり、それぞれ.xtcと.trr,.h5ファイルに対応している。デフォルト値はNoneであり、トラジェクトリを書き出したい場合は必ずどれかを指定する必要がある。

#NonBonded Method

nonbondedmethod

非結合相互作用(NonbondedForce)の長距離成分の計算メソッドを文字列で指定する。利用可能なメソッドはPME(Particle Mesh Ewald法を利用する。)とCutoff（カットオフ以降の原子間相互作用は考慮しない。）の2つである。PMEの場合、後述するいくつかのパラメータを指定する必要がある。Cutoffの場合もそれらのうちの一部のパラメータを指定する必要がある。デフォルトはPMEとなっている。

nonbonded\_cutoffflag

非結合相互作用(NonbondedForce)の静電相互作用成分について、短距離力のcutoffを決めるか決めないかをTrue/Falseで指定する。nonbondedmethodがPMEの場合も指定する必要がある。デフォルト値はTrueとなっている。

nonbonded\_cutoff

非結合相互作用(NonbondedForce)の静電相互作用の短距離cutoff長を実数で指定する。（単位:nm）nonbonded\_cutoffflagがTrueの時のみ有効となる。デフォルト値は1.2 nm。

nonbonded\_cutoff

非結合相互作用(NonbondedForce)の静電相互作用の短距離cutoff長を実数で指定する。（単位:nm）nonbonded\_cutoffflagがTrueの時のみ有効となる。デフォルト値は1.2 nm。

nonbonded\_switchflag

非結合相互作用(NonbondedForce)の静電相互作用の短距離成分のcutoff長での力の打ち切りを連続的にするため、cutoff長以前のある距離からスイッチング関数を用いて相互作用を連続的に減衰させていくスイッチング関数を利用するかしないかをTrue/Falseで指定する。スイッチング関数の詳細は本稿「シミュレーションのパラメータ」Nonbonded Interactions部分および[http://docs.openmm.org/latest/userguide/theory/02\_standard\_forces.html#nonbondedforce](http://docs.openmm.org/latest/userguide/theory/02_standard_forces.html" \l "nonbondedforce)を参考にすること。デフォルト値はTrueである。

nonbonded\_switch

非結合相互作用(NonbondedForce)の静電相互作用の短距離成分のスイッチング関数を印加しはじめる距離を実数で指定する。（単位:nm）。デフォルト値は1.0 nm。cutoffとswitchingの性質上、上記のnonbonded\_cutoffで指定した距離よりも小さな値にする必要がある点に注意すること。

#PME Parameters

setPME

非結合相互作用(NonbondedForce)の静電相互作用の長距離成分計算をPMEとした時の、PME計算のためのパラメータを自身で設定するかしないかをTrue/Falseで指定する。Trueにした場合、下記のPMEのためのパラメータを.inpファイルに明記する必要がある。デフォルト値はFalseとなっている。

pme-alpha

PME計算のためのパラメータの一つであるseparation factorαを実数で指定する。（単位はnm-1。）αの詳細は以下のURLを参照すること。デフォルト値は2.6である。

<http://docs.openmm.org/latest/userguide/theory/02_standard_forces.html#coulomb-interaction-with-particle-mesh-ewald>

pme-nx

PME計算のためのパラメータの一つであるx方向のメッシュサイズを整数で指定する。素数は指定しない方がよく、2や3,5などの倍数がよく利用される。MD計算セルサイズ(単位:nm)を*L*とした時、が0.10程度になる値を設定すると良いと言われる。デフォルト値は64である。

pme-ny,ny

PME計算のためのパラメータの一つであるy,z方向のメッシュサイズを整数で指定する。素数は指定しない方がよく、2や3,5などの倍数がよく利用される。MD計算セルサイズ(単位:nm)を*L*とした時、が0.10程度になる値を設定すると良いと言われる。デフォルト値は64である。

pme-ftol

PME計算のためのパラメータの一つであるδ（error tolerance）を実数で指定する。デフォルト値は1.0e-5である。詳細はpme-alphaに記載されているURLを参照すること。

#Advanced Simulation

ghost\_particle

高分子の自由エネルギーを効率よく計算するための逐次導入法（chain-increment method）計算のためには、溶質ポリマーを構成する一部の繰り返しセグメント（最小単位はモノマー）と周囲の溶媒分子の間の相互作用をoffにする必要がある。ghost\_particleは、このような溶質の一部分の原子集団のみについて溶媒分子との間の相互作用をoffにする計算(ghost状態計算)を行うかどうかをTrue/Falseで指定するパラメータである。デフォルト値はFalseである。ghost状態計算を行う場合、後述のpath\_ndxfileに「どの原子をghostとして扱うか」を原子インデックスで指定したテキストファイル名を明記する必要がある。

path\_ndxfile

上で述べた通り、ghost状態計算を行うために必要な、「溶質分子内のどの原子をghost particle（周囲の溶媒分子との相互作用をoffにした原子）とみなすか」を明記したファイル名を文字列で指定するパラメータ。デフォルト値はsystem.ndxである。このファイルの書式は以下の通りである。現在のパッケージでは、溶質分子は1分子目であることを前提としている。初期構造や力場ファイルもそうなるように系を作成する必要がある。テキストファイル内で、[ Core ]は溶質分子中で周囲の溶媒分子との相互作用がonである通常の粒子群を指す。

この原子インデックスをA-Bの形で指定する。[ Ghost ]がghost状態、つまり周囲の溶媒分子との相互作用がoffである通常の粒子群であり、こちらもA-Bの形で指定する。[ Core ], [ Ghost ]どちらも「[半角スペースCore半角スペース]」、「[半角スペースGhost半角スペース]」とする必要がある。また、A-Bの方は「数字-数字」のようにハイフンの間に半角スペースなどの余計なものが入らないようにする必要がある。

下に記す図4の例では、溶質分子（1~852の852原子で構成される。）のうち、1~681原子目までは溶媒分子との相互作用がonであり、682原子以降が溶媒分子との相互作用がない状態でMD計算を行うという設定になっている。このため、ghost原子は溶媒分子と激しくoverlapしており、このghost状態計算から、Core末端のセグメント部分（X~681原子）の溶液状態（soln）およびGhost先端(682~Y原子)の参照溶媒状態（refs）のトラジェクトリを得ることができる。この.ndxファイルを、Core,Ghost原子インデックスをセグメント1つ分ずつずらしてそれぞれのGhsot状態を指定することで、Ghost状態計算から（溶液系のエネルギー密度汎関数理論であるエネルギー表示法を用いて自由エネルギーを計算するソフトウェアであるERmodを用いた）自由エネルギー計算に必要なトラジェクトリを得ることができる。

[ Core ]

1-681

[ Ghost ]

682-852

図 4: system.ndxの例。溶質分子(原子インデックス1~852)中の1~681原子までを周囲の溶媒分子との相互作用のあるCore部分,682~852原子を相互作用のないGhost部分と見做してMD計算が行われる。

remdflag

レプリカ交換MD計算(REMD)を行うかどうかをTrue/Falseで指定する。現在未実装。

annealingflag

熱浴の温度を逐次的に変化させていくアニーリング計算を行うかどうかをTrue/Falseで指定する。現在未実装。

heavyhydrogen

水素原子の質量を本来の4倍に設定するかどうかをTrue/Falseで指定する。最も軽い水素原子の運動によってMD計算の時間刻み幅が制限されているため、水素原子の運動を質量を増やすことで鈍らせて時間刻み幅をより大きなものに設定できるようにする効果がある。デフォルト値はFalse。

removeCMMotion

本稿「シミュレーションのパラメータ」Removing COM motionにあるように、系の重心運動が起こらないようにMD計算の各ステップで調整するかどうかをTrue/Falseで指定する。現在未実装で、OpenMMのデフォルトはTrueである。

#Platform Input

platform

本稿「シミュレーションのパラメータ」Platformsにあるように、OpenMMでMD計算を行うプラットフォームを指定する。OpenMMにはReference、CPU、CUDA、OpenCLの4つのプラットフォームがあり、GPUを使って計算を高速化したい場合、CUDAかOpenCLのどちらかを選択する必要がある。どちらを用いるかは利用するマシンに依存する。利用するマシンでどのプラットフォームが利用可能か確かめるためには、本稿「インストール方法」にある通り

python -m simtk.testInstallation

を実行する。計算ノードで確かめたい場合、ジョブスクリプトに上記コマンドを書いてジョブを投入し、出力された.outや.errを読めば良い。

precision

MD計算における力の計算等の計算精度を指定する。利用可能な値はdouble,single,mixedである。doubleはdouble precisionで、singleはsingle precisionで力と積分計算が行われる。mixedは特殊で、力の計算はsingleで、積分計算はdoubleで行われる。精度はdoubleが最もよく、mixedはsingleよりもよく系のエネルギーが保存される。一方計算速度はsingleが最も速く、doubleが最も遅い。平衡化計算はdouble かmixedで行うのが無難である。

incrOpenMMのサンプルスクリプト

incrOpenMMパッケージをインポートしてMD計算を行う場合のサンプルスクリプト例を以下に示す。

1 from myopenmm import \*

2 import mdtraj as md

3 from simtk.openmm.app import \*

4 from simtk.openmm import \*

5 from simtk.unit import \*

6 from sys import stdout

7 import sys

8 import os

9

10 args = sys.argv

11 stage = args[1]

12 index = args[2]

13 gpuindex = args[3]

14

15 md\_npt = MDConductor(index)

16 inpf = '../inpdir/' + stage + '/npt\_ghost.inp'

17

18 # Production Run

19 md\_npt.loadFile(inpf)

20 sysgro, systop = md\_npt.preparation(sysdir='SYS/', mddir='MD')

21 simulation\_npt = md\_npt.setup(sysgro, systop, deviceindex=gpuindex)

22

23 simulation\_npt, \_ = md\_npt.mdrun(simulation\_npt, 'npt', index, mddir='MD/')

図 5: incrOpenMMをインポートしてMD計算を行うスクリプト例。（sample.py）

図5のsample.pyは、高分子系のMD計算を行うためのサンプルスクリプトである。各行について解説すると、まず1~8行目で必要なパッケージのimportを行っている。10~13行目は、以降必要となる文字列変数をスクリプト実行時に与えられた引数から読み込むための処理である。stageはMD計算の何段階目の計算なのかを(.inpファイルをinpdir/stageX/のようなインプットファイル達を各ステージごとに分けたディレクトリ内に置くことを前提としている。)、indexは系のインデックスを、（より詳細を述べると、高分子の熱力学計算では、熱力学的に等価かつ初期条件の異なる互いに独立な系を多数作成し、それらを全てMD計算し、サンプリング時につなぎ合わせることで初期構造に依存しない、サンプリング空間における大域的な平衡状態サンプリングを行うことがしばしば行われる。indexはこの互いに独立な系のインデックスを表している。）gpuindexはGPU計算の際、計算ノードに搭載されている何番目のGPUを利用するかを表す変数である。一般的に、スーパーコンピュータ等の計算ノードでは、1つのノードにつき複数のGPUが搭載されていることが多い。（例えば、九州大学のItoでは1ノードあたり4つのGPUが利用可能である。）しかし、OpenMMを用いたMD計算では、１つのGPUを利用したときの計算速度に対して、複数のGPUを利用した時の計算速度がほとんど変化しない。そのため、1つの計算に対して1つのGPUのみを利用するように指定した方がより多くの計算資源を同時に利用可能となる。そこで、GPUの番号を指定するようにして、1計算で1GPUのみを利用するようにしている。

15行目でMDConductor()というクラスが登場している。incrOpenMMではMDConductorがシミュレーションを行う役割を果たしている。上で説明した通り、高分子計算利用のためのスクリプトなので、indexを渡すことがデフォルトとなっている。サンプルスクリプトでは、indexを渡されたクラスがmd\_nptという名前のオブジェクトとして以降で操作されていく。md\_nptは例であり、好きな名前をつけることができる。

16行目では.inpファイルのパスを指定している。10~13,16行目は自身でさまざまに変更可能である。引数渡しが嫌な場合は、sample.py内の変数に具体的な値を代入しても構わない。ただし、15行目の

(任意の名前)=MDConductor(任意のインデックス)

は必ず行う必要がある。これによってincrOpenMMでMD計算を行うための準備が始められる。

19行目で.inpファイルを読み込む関数であるloadFile()を用いて.inpファイルの内容をmd\_nptに渡している。この操作も必須である。

（任意の名前）.loadFile(任意のファイル名)

20行目は.groファイルおよび.top,.itpファイル等のGromacsの入力ファイルをmd\_nptに渡し、groオブジェクトとtopオブジェクトを作成する処理である。preparation()関数がこの処理を行なっており、sysdirで.groファイルと.top,.itpファイルが置いてあるディレクトリ名を、mddirでMD計算の結果出力されるトラジェクトリファイルと.groファイルを書き出すディレクトリ名を指定する。入力ファイルである.groファイルはinput.inpファイルのsysfileで指定した名前+$index.groという形、$sysfile$index.groという名前で、.topファイルは.inpファイルのforcefieldで指定した名前.topで読み込まれるようになっている。

gro, top =（任意の名前）.preparation(sysdir=XX,mddir=XX)

21行目でOpenMMのSimulationオブジェクトを作成している。Simulationオブジェクトの詳細は本稿「OpenMMで利用するクラス」Simulationの項を参照してもらいたい。Simulationオブジェクト作成のために.setup()関数を用いている。20行目で作成したgro,topオブジェクトおよび13行目で読み込んだGPUインデックス番号gpuindexを渡すことでSimulationオブジェクトを作成することができる。

（任意のSimulation名）=（任意の名前）.setup(gro, top, deviceindex=gpuinedx)

23行目で、input.inpファイルで書かれたパラメータ群で指定された設定で実際のMD計算を行なっている。mdrun()関数はmdrun後のsimulationオブジェクトおよび系のエネルギーを(check\_eneflag=Trueとした場合のみ)返す関数であり、二つの変数を読み込むために

simulation\_npt, \_=…という形をとっている。

（任意のSimulation名）, \_=（任意の名前）.mdrun(simulation名, 任意のトラジェクトリ名, index番号,mddir=トラジェクトリを書き出すディレクトリ名)

トラジェクトリはmdrun()関数内のmddirで指定したディレクトリに、最終構造はpreparation()関数内のsysdirで指定したディレクトリに書き出される。

ジョブスクリプト例

上で解説したsample.pyを用いて計算ノード上で計算を実行するジョブスクリプトを以下に記載する。

1 #!/bin/bash

2 #PJM -N "V08051"

3 #PJM -L "rscunit=ito-b"

4 #PJM -L "rscgrp=hp-g-4-dbg"

5 #PJM -L "vnode=1"

6 #PJM -L "vnode-core=36"

7 #PJM -L "elapse=01:00:00"

8 #PJM -X

9 #PJM -e "OUT/npt1.e.1"

10 #PJM -o "OUT/npt1.o.1"

11

12 #---- Program Execution --------#

13 module load cuda/8.0

14 module load intel/2017

15

16 i=1

17 date > OUT/smplog$i.tt

18 num1=`expr 4 \\* $i - 3`

19 num2=`expr $num1 + 1`

20 num3=`expr $num1 + 2`

21 num4=`expr $num1 + 3`

22 num1=`printf %04d $num1`

23 num2=`printf %04d $num2`

24 num3=`printf %04d $num3`

25 num4=`printf %04d $num4`

26

27 mkdir DAT

28

29 echo "#Progress (%) Time (ps) Total Energy (kJ/mole) Temperature (K) Density (g/mL) Speed (ns/day) Time Remaining" > OUT/density$n1.txt

30

31 for n1 in $num1 $num2 $num3 $num4

32 do

33 gpuindex=`expr $n1 % 4`

34 python ./script/sample.py sample1 $n1 $gpuindex >> OUT/log$n1.txt &

35 done

36 wait

図 6:九州大学Ito(system B)でincrOpenMMを実行するジョブスクリプト(run.sh)

１~10行目はItoでジョブを実行する際に使用するジョブ名、リソース名、ノード数、CPUコア数、最大実行時間、アウトプット・エラーファイル名などを指定している。13,14行目はcudaライブラリとintelライブラリをmoduleコマンドを使って計算環境にロードしている。16~25行目は1つのジョブ内で計算する系のindexを指定おり、0001~0004がnum1からnum4までのシェル変数に格納される。27,29行目は読み飛ばしてよい。31~36行目が実際にsample.pyを実行してincrOpenMMを使ってMD計算を実行している部分である。Itoには1つの計算ノードに4基のGPUが搭載されているため、1つのジョブスクリプトで4つの系(0001~0004)の計算を行うためにfor文と&（バックグラウンド実行）、wait文が組み合わされている。33行目は0001から0004の4つの系の計算にそれぞれGPUインデックス0から3までの4つを重複なしで割り振る処理を表しており、シェル変数gpuindexに算術文exprと%（剰余）を組み合わせてこの割り振りを行なっている。実質的なincrOpenMM計算の命令は34行目の一行分で完了する。

python sample.py $stage $index $gpuindex

$stageは.inpファイルを格納しているディレクトリ名を意味しており、上の例ではinpdir/stageX/npt\_ghost.inpという名前で各インプットファイルが置かれている。$index,$gpuindexの意味は上で説明した通り、系のインデックスとGPU番号である。サンプルではOUT/log$n1.txtファイルに出力結果がリダイレクトされている。sample.pyを実行すると、.inpファイルの内容を読み込んで設定した計算内容がアウトプットされる仕様になっており、通常はジョブスクリプトで指定したアウトプットファイルに書き出される。上のジョブスクリプト例では、この内容をlog$n1.txtファイルにリダイレクトするという形をとっている。38~41行目で計算後のSYS/npt$n1.groをSYS/system$n1.groにコピーしており、これによって最終構造を次のMD計算の初期構造とする処理を行なっている。これらの手続きは必ずしなければならないものではなく。sample.pyを走らせる際に必ず必要なのは

python sample.py $stage $index $gpuindex

のみである。

incrOpenMM.MDConductorについての説明

*class incrOpenMM.MDConductor(\*args,\*\*kwargs)*

テキスト形式でシミュレーション設定が書かれたインプットファイルとGromacs形式の構造ファイル(.gro),力場情報ファイル(.top)を読み込んでOpenMMのSimulationオブジェクトを作成したり、MD計算を行うことを可能とするオブジェクト。

\_\_init\_\_(\*args,\*\*kwargs)

MDConductorをインスタンス化するメソッド。

**メソッド**

|  |  |
| --- | --- |
| \_\_init\_\_(self,\*\*kwargs) | MDConductorインスタンスを作成する。 |
| loadFile(self, file) | .inpファイルからMD計算のパラメータ情報を読み込み、インスタンスの属性に格納する。 |
| create\_platform(self, deviceindex) | .inpファイルのplatformとprecision情報を読み込み,それがOpenCLかCUDAの場合GPUのインデックスを読み込む。 |
| preparation(self, sysdir, mddir) | .groファイルと.topのパスと出力先のパスを読み込む。sysgro変数には.inpファイルのX.groに系のindexを足したX$index.groが格納され、これがインプットファイルとなる。 |
| create\_top(self, systop, gro) | 読み込んだ.groファイル名と.top名からOpenMMで利用可能なtopologyオブジェクトを作成する。 |
| create\_system(self, top) | topologyオブジェクトと.inpファイルの情報を読み込み,その情報が書き込まれたsystemオブジェクトを作成する。 |
| create\_integrator(self) | .inpファイルの情報を読み込み,その情報が書き込まれたIntegratorオブジェクトを作成する。 |
| setup(self, sysgro, systop, deviceindex, sysdir) | create\_X()を実際に実行するメソッド。インプット情報をloadしたSimulationオブジェクトを作成する。 |
| minimize(self, simulation, emname, index, sysdir, mddir,max\_iter) | 与えられたSimulationオブジェクトのエネルギー最小化計算を行う。 |
| mdrun(self, simulation, ensname, index, mddir, sysdir, assert\_system, check\_eneflag, nrep, remdflag, niter, niter\_tot) | 与えられたSimulationオブジェクトのMD計算を行う。現状check\_eneflag以降のキーワード引数に関しては未実装である。 |
| convert\_gro2pdb(self, grofile) | grofileを.pdb形式に変換する。parmedパッケージを用いているため、parmedをあらかじめインストールする必要がある。 |
| getIntegratorInfo(self, integrator) | integratorオブジェクトのパラメータ（時間刻み, 抵抗係数, 温度）を取得する。現状LangevinIntegratorのみ対応している。 |

incrOpenMMを利用する場合、実際に用いるメソッドはサンプルにある通り

loadFile() 🡺 preparation() 🡺 setup() 🡺 (minimize()) 🡺 mdrun()

の順番である。

**アトリビュート**

|  |  |
| --- | --- |
| sysindex | 系のindex。 |
| InpDict | .inpファイルの情報をまとめて辞書型に格納している。 |

incremental chemical potentialをERmodで計算する方法

ここでは、ERmod(<https://sourceforge.net/p/ermod/wiki/Home/>)を用いてIncrOpenMMで計算したGhost状態トラジェクトリからincremental chemical potentialを計算する方法について説明する。まず、incrOpenMMでGhost状態計算したトラジェクトリは、system.ndxで指定した溶質内原子グループ（[ Ghost ]グループ）に関して周囲の溶媒分子と相互作用がない状態にある。例として、溶質中の1セグメントあたりの原子数をとする。元のGhost状態計算トラジェクトリA（system\_A.ndx）と、[ Ghost ]グループのインデックスをだけ減らしたsystem\_B.ndxでGhost状態計算を行なったトラジェクトリBを用意する。

1 [ Core ] 1 [ Core ]

2 1-50 2 1-40

3 [ Ghost ] 3 [ Ghost ]

4 51-100 4 41-100

図 7: 端、端以外のセグメントに関して、1セグメントの原子数であり、10個のセグメントから構成される溶質分子の.ndxファイルの例。.ndxファイルは溶質内のCore原子とGhost原子を区別するファイルであり、system\_A.ndx(左)は溶質中の6セグメント以降がGhost原子であることを表しており、 system\_B.ndx(右)は溶質中の5セグメント以降がGhost原子であることを表している。

この時、AとBでは、溶質分子内の原子インデックス41~50までの溶質-溶媒間相互作用*V*が異なっており、Bはであるのに対し、Aは*V*が入っている。A系はGhost部分も周囲の溶媒分子と相互作用するために溶質部分と溶媒分子の間のオーバーラップはなく、エネルギー表示法における溶液系に対応している。Bでは*V*がないために41~50までの溶質分子内原子（以降,溶質Ghost部分と呼ぶ。）は周囲の溶媒分子と激しくオーバーラップしており、B系はこの溶質Ghost部分に関してエネルギー表示理論における参照溶媒系に対応している。

グラフ が含まれている画像

自動的に生成された説明



図 8:system\_A（左）とB（右)の構造サンプル例。赤と青色の原子が溶質分子を、緑色が溶媒分子を表す。赤色の原子は緑色の溶媒分子と相互作用するのに対して、青色が溶媒と相互作用しない[ Ghost ]グループであり、溶媒と激しくオーバーラップしている。ピンクで囲んだ部分がAとBで色が異なっており、このグループが“溶質Ghost部分”である。エネルギー表示法などの自由エネルギー計算を行う立場で見ると、“溶質Ghost部分”に関してAが「溶液系」、Bが「参照溶媒系」である。

A系はエネルギー表示法の意味での「溶液系」の平衡サンプリング計算そのものであり、“溶質Ghost”部分を溶質とみなしてERmodの溶液系(soln)計算を行うことで目的のエネルギーヒストグラムが得られる。一方B系はA系のようにエネルギー表示法における「参照溶媒系」のトラジェクトリそのものではない。ERmodの参照溶媒系(refs)計算では通常、溶質分子の構造（あるいはトラジェクトリ）と溶媒分子系のトラジェクトリをそれぞれ独立に用意し、ermodコマンドを用いることで溶質を溶媒分子環境に（あたかもであるように）ランダムにinsertionしてその際の溶質-溶媒相互作用のエネルギーヒストグラム計算を行うことで「参照溶媒系」のエネルギーヒストグラムを得る。これに対してB系では「溶質分子が溶媒分子環境とというGhost状態」が既に実現している。つまり、B系のトラジェクトリは「溶質が溶媒分子環境に相互作用でinsertionされている状態」のサンプルそのものを表している。そのため、溶質の溶媒環境insertionのような操作を行う必要がなくなる。こうして、B系のトラジェクトリに対してERmodのrefs計算ではなくsoln計算を行うことで、目的である“溶質Ghost”部分を溶質とする「参照溶媒系」のエネルギーヒストグラムを得ることができる。B系のsoln計算によって、溶液系で観察されるようなエネルギーヒストグラムに加えて、溶質分子と溶媒分子がオーバーラップする事による高エネルギー領域（排除体積領域）のヒストグラムもサンプリングすることが可能である。ただし、通常のERmodでは、soln系のエネルギーヒストグラム計算で排除体積領域のような高エネルギー状態を検知した場合、異常終了する仕様になっている。そのため、B系のエネルギーヒストグラム計算のためには、この検知機能をoffにした修正ERmod(ERmod\_mod)を用いる必要がある。修正ERmodのインストール方法は本稿「インストール方法」7.を参照すること。まとめると、

A系:soln計算を行う。計算は通常のERmodを用いる。

B系:soln計算でrefs系のヒストグラム計算を行う。修正したERmodを用いる。

となる。

大まかな流れは上記の通りだが、incremental chemical potential計算のためには、さらに以下に記す2つの手続きをしなければならない。長大になってしまうので先に簡単に列挙すると、

1. 正しいエネルギーヒストグラム計算のための再重みづけ

2. B系のrefs系ヒストグラム計算のためのパラメータ追加

が必要となる。以下でそれぞれを詳しく説明する。

1.Ghost状態の再重み付け

実は、相互作用に長距離成分（例えば静電相互作用をPMEで計算している場合）が存在する場合、**上で説明したGhost状態はA系とB系どちらも真にGhost部分と溶媒分子間の相互作用が０である平衡状態になっていない**。これはincrOpenMMでGhost状態を再現しようとするときの操作に原因がある。Ghost部分と溶媒分子の間の相互作用のみをoffにするために、incrOpenMMでは次のような操作を行なっている。

1.Ghost部分の非結合相互作用パラメータ（電荷およびLJ相互作用のエネルギーパラメータ）を全て0にする。これによってGhost-溶媒分子間相互作用は完全に0になるが、同時に溶質内相互作用（図8の赤-青間相互作用）も０になってしまう。

2.溶質内相互作用をonにするために、赤-青ペアの非結合相互作用を例外(Exception)として入れる。(NonbondedForceを追加する。)このとき、非結合相互作用のポテンシャル形はLJポテンシャルとクーロン相互作用()の和として表現する。このExceptionには長距離成分を考慮する機能がないため、本来PME等で考慮すべき相互作用が抜け落ちてしまっているという問題点がある。正しい赤-青ペアの非結合相互作用エネルギーを含む系全体のエネルギーは、後ろで示す手順で再計算が可能である。幸い、赤-青ペアの非結合相互作用のエネルギーのずれは溶質内相互作用の長距離成分のみであり、系全体のエネルギーと比較すると相対的にかなり小さい。具体的には、鎖長50のポリビニルピロリドン(PVP)で1~3 kcal/mol程度の値をとる。500 Kの高分子溶融系の熱ゆらぎのエネルギーがおよそ1 kcal/molであることを考えると、差はあまり大きくないものであることがわかる。

このエネルギー差を計算して、**真のGhost状態にあるA系とB 系のエネルギーサンプリングを行うためにincrOpenMMで計算したA’系とB’系のエネルギーサンプリングの再重み付け(reweighting)を行う。**再重み付けは、X’系とX系のエネルギー差のボルツマン因子（を以下のように規格化したもの）を各フレーム*t*ごとに掛けることで実行する。

上式のように、各タイムステップでこの重みをかけることで正しいGhost状態の平衡アンサンブルを得ることができる。

正しい系のエネルギー*E*(*X*)の計算方法

*E*(*X*)を計算する方法を図9にまとめる。Ghost状態計算で書き出したトラジェクトリをOpenMMとMDTRAJを用いてrerunしてことで再度エネルギーを計算していく。その際、図9にあるような3つの異なるtopologyファイルを使ってそれぞれのエネルギーを計算する。

エネルギー再計算, *E*exactの計算方法

異なるtopologyファイルを3つ作成

グラフ

中程度の精度で自動的に生成された説明1. B (ghost)部分の*e*, *q*を全て0にしたtopology

*E*1 = *EAA* + *EAS* + *ESS*

2. S (solvent)部分の*e*, *q*を全て0にしたtopology

*E*2 = *EAA* + *EAB* + *EBB*

3. A (core)部分以外の*e*, *q*を全て0にしたtopology

*E*3 = *EAA*

*E*exact = *E*1 + (*E*2 – *E*3) で計算可能

3つのtopologyファイルを使い、3回rerunしてエネルギー計算

1 + (2–3) から*E*exact を算出

OpenMMの結果である E\_OpenMM との差が補正エネルギーD*E*

図 9: 正しいエネルギー*E*(*X*)の計算方法の説明。はLJポテンシャルのエネルギー、*q*は原子電荷。

3つのtopologyの作り方

POLYMER\_A/Bが1:1で混ざっているポリマーブレンド系を例に説明する。元々のtopologyファイルがPOLYMER\_A.itpとPOLYMER\_B.itpをincludeする形式を採用していると仮定する。POLYMER\_Aの溶質分子にPOLYMER\_A\_sltという名前をつけ、この溶質のセグメント部分のincremental chemical potentialを計算することを考える。

**topology\_01.top**

図9で説明した1つ目のtopologyファイルを下図に載せる。POLYMER\_A\_sltのGhost部分のと*q*を全て0にする必要がある。

[ defaults ]

;nbfunc comb-rule gen-pairs fudgeLJ fudgeQQ

1 2 yes 0.5 0.83333333

[ atomtypes ]

; atom type mass q particle\_type sigma[nm] epsion[kJ mol^-1]

c3 12.01 0.0 A 0.3399669508 0.4577296000

ho 1.008 0.0 A 0.0 0.0

h1 1.008 0.0 A 0.2471353044 0.0656888000

oh 16.0 0.0 A 0.3066473387 0.8803136000

hc 1.008 0.0 A 0.2649532787 0.0656888000

c 12.01 0.0 A 0.3399669508 0.3598239999

n 14.01 0.0 A 0.3249998523 0.7112800000

o 16.0 0.0 A 0.2959921901 0.8786399999

c30 12.01 0.0 A 0.3399669508 0.000000000

ho0 1.008 0.0 A 0.0 0.0

h10 1.008 0.0 A 0.2471353044 0.000000000

oh0 16.0 0.0 A 0.3066473387 0.000000000

hc0 1.008 0.0 A 0.2649532787 0.000000000

c0 12.01 0.0 A 0.3399669508 0.000000000

n0 14.01 0.0 A 0.3249998523 0.000000000

o0 16.0 0.0 A 0.2959921901 0.000000000

#include "POLYMER\_A\_slt.itp"

#include "POLYMER\_A.itp"

#include "POLYMER\_B.itp"

[ system ]

;Name

example

[ molecules ]

POLYMER\_A\_slt 1

POLYMER\_A 99

POLYMER\_B 100

図 10:topology\_01.top。[ atomtypes ]行のc30〜o0までの原子がGhost原子群のパラメータであり、ここでepsilonを全て0にする。

図10のように、溶質であるPOLYMER\_A\_sltが1分子、そのほかのPOLYMER\_AとBがそれぞれ99,100分子をincludeする形でtopologyファイルを用意する。[ atomtypes ]行でPOLYMER\_A\_slt, A,Bの原子の基本的なパラメータ（。部分電荷*q*は.itpファイルで上書きする。）を決める。“c3”〜“o”までがPOLYMET\_A,Bの原子であり、通常のtopologyファイルにはこの部分までが記されている。以降の“c30”〜“o0”がPOLYMER\_A\_sltのGhost部分の原子であり、この部分のLJポテンシャルのエネルギーパラメータを0にする。部分電荷*q*はここでは修正せず、溶質分子の.itpファイルであるPOLYMER\_A\_slt.itpを直接操作して0にする。以下にPOLYMER\_A\_slt.itpの中身を示す。

**POLYMER\_A\_slt.itp**

; And here is the topology.

[ moleculetype ]

;Name nrexcl

POLYMER\_A\_slt 3

[ atoms ]

; nr type resnr residu atom cgnr charge

1 c3 1 POLYMER\_A\_slt c3 1 -0.5294384388892324

2 h1 1 POLYMER\_A\_slt h12 2 0.09524686111076762

3 c 1 POLYMER\_A\_slt c1 3 0.5213595611107676

4 c3 1 POLYMER\_A\_slt c3 4 0.07468756111076762

5 c3 1 POLYMER\_A\_slt c3 5 -0.12029543888923239

6 c3 1 POLYMER\_A\_slt c3 6 0.02223256111076761

7 n 1 POLYMER\_A\_slt n4 7 -0.2723564388892324

8 o 1 POLYMER\_A\_slt o5 8 -0.5634284388892323

9 c3 1 POLYMER\_A\_slt c3 9 0.48690356111076766

10 hc 1 POLYMER\_A\_slt hc3 10 0.0016434388892323896

11 hc 1 POLYMER\_A\_slt hc3 11 0.02537156111076761

12 hc 1 POLYMER\_A\_slt hc3 12 0.06717056111076762

13 hc 1 POLYMER\_A\_slt hc3 13 0.04536056111076761

14 hc 1 POLYMER\_A\_slt hc3 14 0.00449156111076761

15 hc 1 POLYMER\_A\_slt hc3 15 0.022700561110767608

16 h1 1 POLYMER\_A\_slt h12 16 -0.06885643888923239

17 h1 1 POLYMER\_A\_slt h12 17 0.09524686111076762

18 h1 1 POLYMER\_A\_slt h12 18 0.09524686111076762

19 c30 1 POLYMER\_A\_slt c3 19 0.00

20 h10 1 POLYMER\_A\_slt h12 20 0.00

21 c0 1 POLYMER\_A\_slt c1 21 0.00

22 c30 1 POLYMER\_A\_slt c3 22 0.00

23 c30 1 POLYMER\_A\_slt c3 23 0.00

24 c30 1 POLYMER\_A\_slt c3 24 0.00

25 n0 1 POLYMER\_A\_slt n4 25 0.00

26 o0 1 POLYMER\_A\_slt o5 26 0.00

27 c30 1 POLYMER\_A\_slt c3 27 0.00

28 hc0 1 POLYMER\_A\_slt hc3 28 0.00

29 hc0 1 POLYMER\_A\_slt hc3 29 0.00

30 hc0 1 POLYMER\_A\_slt hc3 30 0.00

31 hc0 1 POLYMER\_A\_slt hc3 31 0.00

32 hc0 1 POLYMER\_A\_slt hc3 32 0.00

33 hc0 1 POLYMER\_A\_slt hc3 33 0.00

34 h10 1 POLYMER\_A\_slt h12 34 0.00

35 h10 1 POLYMER\_A\_slt h12 35 0.00

図 11:セグメント数*N*seg=18,２番目以降のセグメントがGhost部分である場合のPOLYMER\_A\_slt.itp。原子インデックス1~18まではPOLYMER\_Aと同じ“c3”等の原子であり、epsilonも*q*もPOLYMER\_Aと同じだが、19番目以降は topology\_01.topで指定したepsilon=0の原子群に置き換わっている。この部分のcharge部分を全て0にすることで部分電荷を0にする。

備考：“c30”のように名前を付け替える必要があるのは[ atoms ]のnr列であり、atom列に関しては修正する必要はない。

**topology\_02,03.top**

2つ目と3つ目のtopologyファイルはどちらも溶媒分子（今の例では溶質分子以外のPOLYMER\_AとPOLYMER\_B）の非結合相互作用をすべてoffにした系である。topology\_01.topの時と同様に、溶媒分子のepsilonと部分電荷*q*を0にすればよい。その上で、

・topology\_02: POLYMER\_A\_sltのepsilonと*q*を一切操作しない（Ghost部分なし）

・topology\_03: POLYMER\_A\_sltのepsilonとqを操作する（Ghost部分あり）

のようにしてtopologyを作成する。ちなみに、topology\_02と03は「溶質分子（Ghost部分なしとGhost部分あり）のみが存在する系」と考えることもできる。そのため、Gromacsのtrjconv等のコマンドを用いてGhost状態計算のトラジェクトリから溶質部分のみを抜き出したsolute.xtcを作成し、溶質分子のみ（Ghost部分なしとGhost部分あり）のみのtopologyをtopology\_02,03.topとすることもできる。こちらの方がrerun計算にかかる時間が短くなるというメリットがある。

OpenMMを用いたrerun計算の例

上で作成したtopology\_01,02,03を使って、Ghost状態のトラジェクトリ(.xtc)を読み込んでエネルギー再計算を行うスクリプトの例を図12に示す。例では

・上で作成したtopologyとgroファイル、OpenMMのインプットを読み込む

・エネルギー再計算を行いたいトラジェクトリであるMD/nptXXXX.xtc(Xには任意の数字が入る。)をmdtrajを使ってロードする

・各タイムステップでの原子配置と基本セルのbox長（周期境界条件を考慮するために必要。）を読み込む

・その状態のエネルギーを計算し、アウトプットファイルに書き出す

という処理をしている。これをtopology\_01,02,03に関して行うことで図9のを計算することで、正確なGhost状態エネルギー*E*(*X*)を得ることができる。また、元のghost状態のエネルギーに関しても、.logファイルに書き出された値ではなく、図12のスクリプトでrerunすることを推奨する。（.logファイルに書き出されたエネルギー値では数kJ/mol程度のずれが生じることが確認されたため。理由は不明だが、.xtcの構造情報から再計算したエネルギー値で統一した方が正確であると考えられるため、*E*(*X’*)もrerunで計算することを推奨する。）以上から、各タイムステップにおけるエネルギーのずれ*E*(*X’*)-*E*(*X*)を計算できる。

1 from incropenmm import \*

2 import mdtraj as md

3 from simtk.openmm.app import \*

4 from simtk.openmm import \*

5 from simtk.unit import \*

6 from sys import stdout

7 import sys

8 import os

9 import parmed as pmd

10

11 args = sys.argv

12 stage = args[1]

13 index = args[2]

14 gpuindex = args[3]

15

16 md\_npt = MDConductor(index)

17 inpf = '../inpdir/' + stage + '/npt\_ghost.inp'

18

19 # Production Run

20 md\_npt.loadFile(inpf)

21 sysgro, systop = md\_npt.preparation(sysdir='SYS/', mddir='MD')

22 simulation\_npt = md\_npt.setup(sysgro, systop, deviceindex=gpuindex)

23

24 context = simulation\_npt.context

25

26

27 ofname = 'DAT/npt' + index + '\_ghost.dat'

28 fname = 'MD/npt' + index + '.xtc'

29 #fname = 'MD/energy' + index + '.trr'

30 with open(ofname, 'wt') as f:

31 f.write('# step\tEnergy(kJ/mol)\n')

32

33 # load trajcetory

34 traj = md.load(fname,top=sysgro)

35 # read box informations

36 ucell\_vs = traj.unitcell\_vectors

37

38 for i in range(len(traj)):

39 # read and set positions

40 context.setPositions(traj.openmm\_positions(i))

41 # set periodicbixvectors

42 ucell\_v = ucell\_vs[i]

43

44 context.setPeriodicBoxVectors(ucell\_v[0],ucell\_v[1],ucell\_v[2])

45 state = context.getState(getEnergy=True)

46 energyval = state.getPotentialEnergy()

47 energyval /= kilojoules/mole

48 #print(i, energyval)

49 with open(ofname, 'a+') as f:

50 f.write('{0:04d}\t{1:10.3f}\n'.format(i, energyval))

図 12: トラジェクトリを読み込んでエネルギー再計算を行うサンプルスクリプト(rerun.py)。変数inpfにincrOpenMMのインプットファイルを代入してロードし、MD/npt$index.xtcを読み込んでエネルギーのみ再計算を行い、DAT/npt${index}\_ghost.datにエネルギー値を書き出している。

ERmodでreweighting計算を行う方法

reweightingを行うには、ERmodでermod計算を行う際にparameters\_erファイルのwgtsysパラメータを1に設定する必要がある。parameters\_erはERmodでエネルギーヒストグラム計算を行うermodコマンドを実行するためのパラメータ群をまとめたファイルである。（<https://sourceforge.net/p/ermod/wiki/parameters-ermod03/>を参照すること。）

該当箇所を抜き出す。

wgtsys : weight of a trajectory snapshot of the solution or (reference-)solvent system

0 (default) : no 1 : yes

If yes, a file called SysWght is separately prepared in the form of

(snapshot number, integer) (weight, real)

before ermod runs.

上にある通り、ERmodのermod計算を行うディレクトリ(soln/, refs/)において、parameters\_erファイルでwgtsys=1と設定した上で、SysWghtファイルに各時間ステップと、その重みを書き出して保存する。この状態でermodコマンドを実行することでエネルギーヒストグラムのreweighting計算を実施する。

2.Ghost状態計算のための参照溶媒系ermod計算

グラフ が含まれている画像

自動的に生成された説明



図 13: 　system\_A（左）とB（右)の構造サンプル例の再掲。ピンク色で囲んだ“溶質Ghost部分”に関してAが「溶液(soln)系」、Bが「参照溶媒(refs)系」となる。P.34で説明した通り、ermod計算はsystem\_AもBもsolnの作法で行う。system\_Bのermod計算後、slvfe計算のために出力されたファイル等をrefs系の名前に変更する手続きが必要となる。

P.37で説明した通り、incremental chemical potential計算のためのermod計算は、「溶液(soln)系」でも「参照溶媒(refs)系」でもsoln系の方法で計算を行う必要がある。soln系の方法でrefs系のエネルギーヒストグラムおよびエネルギー相関行列を得るというのは通常と異なる特殊な計算であるため、追加の手続きが必要となる。ERmodについてはERmod wiki(<https://sourceforge.net/p/ermod/wiki/Home/>)

を参考にすること。

parameters\_erの修正

solnのermod計算を行う場合、通常ではエネルギーヒストグラムengsln.XXおよび各ブロックの平均相互作用エネルギーaveuv.tt,各時刻の溶質-溶媒相互作用エネルギーflcuv.tt,および各ブロックの重みweight\_solnのみが出力される。refsではこれに加えて相互作用エネルギー相関行列corref.XXとself partを含んだ重みweight\_refsが出力されてslvfe計算に利用される。incremental chemical potential計算ではsoln計算からrefs系の情報を得なければならないため、corref.XXとweight\_refsを得るために追加のオプションを指定する必要がある。そのために、parameters\_erの&eneparam部分に以下の行を追記する。

#parameters\_er

corrcal = 1,

wgtslf = 1,

これで修正したERmodを用いてsoln計算を「参照溶媒系」で実行する。上のようにパラメータcorrcalとwgtslfを設定することで、エネルギー相関行列corsln.XXとweight\_solnのself partを得ることができる。最後に、slvfe計算のために、得られたcorsln.XX, engsln.XX, wight\_soln等の名前を全てrefs系のものに変更する。

engsln.XX 🡺 engref.XX

corsln.XX 🡺 corref.XX

weight\_soln 🡺 weight\_refs

以上が、incremental chemical potential計算におけるrefs系の計算方法である。

incremental chemical potential計算のermod計算の流れ

最後に、incremental chemical potentialを計算する際のermod計算の流れをまとめる。

1. OpenMMで「溶液(A)系」と「参照溶媒(B)系」のGhost状態計算を行う

system.ndxを作成してGhost原子を指定する

.inpファイルのghost\_particlesフラグをTrueにする

1. A系とB系でそれぞれエネルギーのずれをrerun計算する

topology\_01,02,03.topおよび.itpファイルを作成してrerun.pyを実行する

1. ずれのボルツマン因子を計算してsoln(A)/, refs(B)/にSysWghtを書き出す

計算したSysWghtをERmod\_XX/soln/,refs/に置く

1. A系を通常のERmodでreweighting計算(solnの方法)

soln/parameters\_erのwgtsys = 1にして、通常のERmodでermodを行う

1. B系を修正ERmodでreweighting計算（**solnの方法**）

refs/parameters\_erのwgtsys = 1にして、**修正したERmod**でermodを行う

以上でERmod\_XX/soln/,refs/にslvfe計算のためのファイルが出力される。後は通常のERmod計算同様、ERmod\_XX/でslvfeコマンドを実行することでincremental chemical potentialが得られる。

Tips

「溶質」のSltInfoやMDinfo等インプットファイルの作り方

incremental chemical potential計算では、ポリマーを構成するセグメントの一部分のみを溶質分子とみなしてエネルギーヒストグラム計算を行う。この際、parameters\_erのみならず、SltInfoやMdinfo, MolPrmといったファイルを一部書き換える必要があるので説明する。

グラフ が含まれている画像

自動的に生成された説明



図 14:　溶液系(左)と参照溶媒系（右）の違いを色で表している。incremental chemical potential計算ではピンク色で囲んだ“Ghost部分”のみが溶質であり、ERmodにおける溶質の情報を表すインプットファイルSltInfoも特殊な形で用意する必要がある。

POLYMER\_AとBの二種類のポリマーがそれぞれ、個存在する場合のincremental chemical potential計算を行うことを考える。図14のように、POLYMER\_A内のピンク色の部分が溶質である場合を考える。soln/, refs/ディレクトリの作成は通常のERmodと同じくgen\_structureコマンドを用いてtopologyファイルを指定し、溶質をPOLYMER\_Aに指定して作成する。その後, 各ディレクトリ内のSltInfoとMDinfo, MolPrmファイルを直接修正する。

SltInfo

gen\_structureで作成したSltInfoには、POLYMER\_Aの非結合相互作用パラメータ情報が全ての原子について書き出されている。今回知りたいのはその一部（図14ピンク色の部分）の溶媒和自由エネルギーであるので、ピンク色部分の原子以外の非結合相互作用パラメータのうち、エネルギー情報に関わる量である*q*とepsilonを全てゼロにする必要がある。これはSltInfoの3、４列目（それぞれ*q*とepsilonを意味する）を0にすればよい。図15にSltInfoファイルの例を掲載する。

…

405 H 0.0000 0.0000 0.0000

406 H 0.0000 0.0000 0.0000

407 H 0.0000 0.0000 0.0000

408 H 0.0000 0.0000 0.0000

409 H 0.0000 0.0000 0.0000

410 C -0.52924447 0.4577296 0.33996695

411 H 0.13129053 0.0656888 0.2471353

412 C 0.52155353 0.359824 0.33996695

413 C 0.074881529 0.4577296 0.33996695

414 C -0.12010147 0.4577296 0.33996695

415 C 0.022426529 0.4577296 0.33996695

416 N -0.27216247 0.71128 0.32499985

417 O -0.56323447 0.87864 0.29599219

418 C 0.48709753 0.4577296 0.33996695

419 H -0.0014494706 0.0656888 0.26495328

420 H 0.025565529 0.0656888 0.26495328

421 H 0.067364529 0.0656888 0.26495328

422 H 0.045554529 0.0656888 0.26495328

423 H 0.0046855294 0.0656888 0.26495328

424 H 0.022894529 0.0656888 0.26495328

425 H -0.068662471 0.0656888 0.2471353

426 H 0.15154053 0.0656888 0.2471353

427 C 0.0000 0.0000 0.0000

428 H 0.0000 0.0000 0.0000

429 C 0.0000 0.0000 0.0000

430 C 0.0000 0.0000 0.0000

…

図 15:　溶質部分のみを非零にしたSltInfoの例（一部抜粋）

MDinfo

POLYMER\_AとBの二種類が混在するポリマーブレンドのtopologyでgen\_structureを行なうため、MDinfoは例えば以下のようになる。

FRAMES 2

852 352

上述した通り、溶質はPOLYMER\_Aのセグメントの一部のみであり、SltInfoはPOLYMER\_Aとは大きく異なる。そのため系には三種類のポリマーが存在することになり、MDinfoも

FRAMES 3

1

852 852 352

と変更する必要がある。

MolPrm1,2

MolPrmXはX番目の溶媒分子の非結合相互作用パラメータ情報が書き込まれたファイルである。MDinfoのところで述べた通り、ポリマーの種類が2から3に増えるため、

元のMolPrm1をMolPrm2に、修正前のSltInfoをMolPrm1にrenameする必要がある。

MolPrm1(POLYMER\_B) 🡺 MolPrm2(POLYMER\_B)

SltInfo(POLYMER\_A) 🡺 MolPrm1(POLYMER\_A)

SltInfo, MDinfo, MolPrmXファイルはsoln/, refs/で共通したものを使う。

複数構造を平均化して平均溶媒和自由エネルギーを計算する方法

ポリマーブレンド系は数十ns程度のシミュレーション中に大規模な構造変化をほとんど起こさない。このため、溶媒和自由エネルギー等の物理量がシミュレーションの初期配置に強く依存することが知られている。ポリマーシミュレーションの分野では、サンプリングの偏りを防ぐため、数十〜数百構造の互いに独立な構造サンプリング計算を行い、各構造を何らかの意味で平均化して自由エネルギーを計算するという手順をとることが多い。ERmodを用いた自由エネルギー計算の場合、一体および二体のエネルギーヒストグラムを平均化することで平均自由エネルギーを求める。