Biomolecular potential energy functions

Soohaeng Yoo Willow
syoo7@iit.edu
Jan. 12, 2021

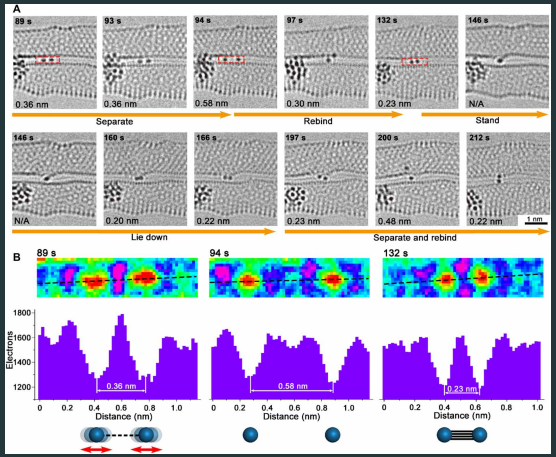
Experiment

- Transmission electron microscopy(TEM)

> Cao *et al* imaged the structure and dynamics of Re_2 molecules on the level of the single atom in real time
>
> —— Cao *et al*, *Sci. Adv.* 2020; 6: eaay5849

- Re : the atomic number 75, weight: 186.21 u
- movie

Experiment



A) Time-series AC-HRTEM images
B) (the first row) false-color images, (the second row) the intensity profiles, (the third row) the possible states of the two Re atoms

Limit in Experiments

- They could not provide a dynamics of proteins, organic molecules, and water molecules.

Contents

The Kinetic Theory of Ideal Gases

- There is no intermolecular interactions between the molecules of an ideal gas. In short, the potential energy E_{pot} of the system is zero: $E_{\mathrm{pot}} = 0$
- The total energy E_{tot} of the system becomes the kinetic energy E_{kin} of the system: $E_{\mathrm{tot}} = E_{\mathrm{kin}} + E_{\mathrm{pot}} = E_{\mathrm{kin}}$.
- $$E_{\mathrm{kin}} = \frac{3}{2} N k_B T = \sum_{A=1}^{N} \frac{1}{2} \frac{|\boldsymbol{p}_A|^2}{m_A}, \tag{1}$$

  where N is the number of atoms in the system, k_B is the Boltzmann constant. T is the thermal temperature of the system. $\boldsymbol{p}$ and m are the momentum and the mass.
- The momentum or the velocity is related to the thermal temperature T.

Dynamics

- The motion of atoms is described by **Newton's laws of motion**
- The second law states:

> The rate of change of the momentum of a body is directly proportional to the force applied, and this change in momentum takes place in the direction of the applied force
>
> —— Newton

- $\boldsymbol{F}_A = -\nabla_A V$, where $\boldsymbol{F}$ is a force acting on Ath atom. V is a potential energy.
- When we know the potential energy V of the system, the motion of atoms in the system is precisely predicted according to Newton's laws of motion:

$$\boldsymbol{R}_A(t+\delta t) = \boldsymbol{R}_A(t) + \boldsymbol{v}_A(t)\delta t + \frac{1}{2}\frac{\boldsymbol{F}_A(t)}{m_A}\delta t^2$$

Potential Energy: The Born-Oppenheimer Approximation

- time-independent Schrödinger equation:

$$
\begin{aligned}
E &= \langle \Psi | \hat{H} \Psi \rangle \\
\hat{H} &= -\sum_{i=1}^{N} \frac{1}{2} \nabla_i^2 - \sum_{A=1}^{M} \frac{1}{2M_A} \nabla_A^2 - \sum_{i=1}^{N} \sum_{A=1}^{M} \frac{Z_A}{|\boldsymbol{r}_i - \boldsymbol{R}_A|} \\
&\quad + \sum_{i=1}^{N} \sum_{j>i}^{N} \frac{1}{|\boldsymbol{r}_i - \boldsymbol{r}_j|} + \sum_{A=1}^{M} \sum_{B>A}^{M} \frac{1}{|\boldsymbol{R}_i - \boldsymbol{R}_j|}
\end{aligned}
$$

- Assumption: the electrons in a molecule are moving in the field of fixed nuclei.(from Szabo and Ostluand, **Modern Quantum Chemistry**)

$$
\begin{aligned}
\hat{H}_{\text{elec}} &= -\sum_{i=1}^{N} \frac{1}{2} \nabla_i^2 - \sum_{i=1}^{N} \sum_{A=1}^{M} \frac{Z_A}{|\boldsymbol{r}_i - \boldsymbol{R}_A|} + \sum_{i=1}^{N} \sum_{j>i}^{N} \frac{1}{|\boldsymbol{r}_i - \boldsymbol{r}_j|} \\
E_{\text{elec}} &= \langle \Psi_{\text{elec}} | \hat{H}_{\text{elec}} | \Psi_{\text{elec}} \rangle \\
E &= E_{\text{elec}} + \sum_{A=1}^{M} \sum_{B>A}^{M} \frac{1}{|\boldsymbol{R}_i - \boldsymbol{R}_j|}
\end{aligned}
$$

Movie

- Born-Oppenheimer Molecular Dynamics (BOMD)
- Path Integral Molecular Dynamics (PIMD)

Potential Energy

- Intra-molecular potential energy of the Ith monomer:

$$E_I - E_I^{\min} = E_{\text{bond}} + E_{\text{angle}} + E_{\text{torsion}},$$

  where $E_I = \langle \Psi_I | \hat{H}_I | \Psi_I \rangle$ represents the potential energy of Ith monomer.

- Inter-molecular potential energy:

$$E_{IJ} - E_I - E_J = E_{\text{LJ}} + E_{\text{Coul}},$$

  where $E_{IJ} = \langle \Psi_{IJ} | \hat{H}_{IJ} | \Psi_{IJ} \rangle$ represents the potential energy of the dimer consisting of Ith and Jth monomers.

-

$$E_{\text{LJ}} = \sum_{A>B} 4\epsilon_{AB} \left[\left(\frac{\sigma_{AB}}{R_{AB}} \right)^{12} - \left(\frac{\sigma_{AB}}{R_{AB}} \right)^{6} \right], \quad E_{\text{Coul}} = \sum_{A>B} \frac{q_A \, q_B}{R_{AB}}$$

Polarization Energy

$$
\begin{aligned}
E_I^{\mathrm{pol}} &= \langle \Psi_{I:Q_I} | \hat{H}_{I:Q_I} | \Psi_{I:Q_I} \rangle - \langle \Psi_I | \hat{H}_{I:Q_I} | \Psi_I \rangle \\
E^{\mathrm{pol}} &= 0.5 \sum_I E_I^{\mathrm{pol}}
\end{aligned}
$$

Quantum Chemistry Software

- Plane-wave:

$$\Psi(\boldsymbol{r}) = \frac{1}{\sqrt{\Omega}} \exp[i\,\boldsymbol{G}\cdot\boldsymbol{r}],$$

where Ω is the volume of the box, $\boldsymbol{G}$: Reciprocal lattice vector
- ABINIT, CP2K, CPMD, Quantum ESPRESSO6, VASP
- Density Functional Theory
- PBC (solid or liquid)

- With Gaussian-type orbitals:

$$\Psi_k(\boldsymbol{r}) = \exp[-\alpha_k|\boldsymbol{r}-\boldsymbol{R}|^2]$$

- Gaussian, GAMESS(US), MPQC, NWChem, PSI, PySCF, Q-Chem
- Hartree-Fock Theory, Kohn-Sham Density Functional Theory, Møllet-Plesset purturbation theory (MP), Coupled-Cluster method
- molecules

Prerequisites: Install the dependencies

- python 3.7
- (MM) Install OpenMM

```
conda install -c conda-forge openmm
or
conda install -c omnia openmm
```

- (QM) Install pyscf

```
pip install pyscf
```

- (GeomOpt) Install pyberny

```
pip install pyberny
```

Prerequisites: Exercise

- https://github.com/swillow/pdb2amber
- https://github.com/swillow/modelingworkshop

pyscf: Python-based Simulations of Chemistry Framework

- Free
- Python
- Easy to install:

```
$ pip install pyscf
```

drawback

- we have to know how to use a python.

Code

```
from pyscf import gto, scf
mol = gto.M(atom='H 0 0 0; H 0 0 1.2', basis='ccpvdz')
mf = scf.RHF(mol)
mf.kernel()
```

```
python run_qmmm.py -i input.json
```

input.json

```
{
    "theory": "qm",
    "job": "ener", # gopt
    "qm": {
        "method": "scf",
        "basis": "6-31gs",
        "charge": 0,
        "fname_geom": "qm_step4.xyz",
        "esp": true,
        "esp_opts": {"resp": true,"resp_hfree": true}
    }

}
```

Molecular Mechanics Force Field

- The **force field** refers to the *functional form* and *parameter* sets used to calculate the potential energy of a system.
- The parameters for a chosen energy function are derived from experiments, calculations in quantum mechanics, or both.
-

$$
\begin{aligned}
E &= E_{\text{intra}} + E_{\text{inter}} \\
E_{\text{intra}} &= E_{\text{bond}} + E_{\text{angle}} + E_{\text{torsion}} \\
E_{\text{inter}} &= E_{\text{vdW}} + E_{\text{Coul}} \\
E_{\text{bond}} &= \frac{k}{2}(r - r_0)^2
\end{aligned}
$$

- The **force constant** k is determined from the experimental *Infrared spectrum*, *Raman spectrum*, or high-level *quantum mechanical calculations*. This value is related to the vibrational frequencies.

Molecular Mechanics Force Field Parameter Sets

- AMBER
- CHARMM
- MMFF
- OPLS
- AMOEBA

Topology

- lists of chemical bonds, angles, torsional angles, *et al*

OpenMM: A High Performance Molecular Dynamics Library

- http://openmm.org/
- Free
- Python
- Easy to install:

```
$ conda install -c conda-forge openmm
```

drawback

- we have to know how to use a python.

Code

```
from simtk.openmm.app import *
from simtk.openmm import *
from simtk.unit import *
from sys import stdout

prmtop = AmberPrmtopFile('input.prmtop')
inpcrd = AmberInpcrdFile('input.inpcrd')
system = prmtop.createSystem(nonbondedMethod=PME, nonbondedCutoff=1*nanometer, \
        constraints=HBonds)
integrator = LangevinIntegrator(300*kelvin, 1/picosecond, 0.002*picoseconds)
simulation = Simulation(prmtop.topology, system, integrator)
simulation.context.setPositions(inpcrd.positions)
if inpcrd.boxVectors is not None:
    simulation.context.setPeriodicBoxVectors(*inpcrd.boxVectors)
simulation.minimizeEnergy()
simulation.reporters.append(PDBReporter('output.pdb', 1000))
simulation.reporters.append(StateDataReporter(stdout, 1000, step=True, \
    potentialEnergy=True, temperature=True))
simulation.step(10000)
```

Main protein from SARS-CoV-2

- Jin, Z. *et al* Nature, Vol 582, page 289 (2020)
- PDB: 6LU7
  - 6LU7 contains coordinates of heavy atoms (C, N, O, S) of a monomer.

1. separate a substrate (or inhibitor) from a protein
2. add H atoms
   (option) build a dimer from a monomer
3. generate an AMBER parameter and topology file (prmtop) of a protein (to get the charge of the protein)
4. build a system consisting of a protein, water, and salt.
5. generate an AMBER parameter and topology file (prmtop) of the whole system.
6. Minimization, NVT MD simulations, and NPT MD simulations

Step 1. Separate a ligand from a protein

```
python separate_complex.py -i input_step1.json
```

input_step1.json

```
{
    "fname_xtal": "./MPRO/6lu7.pdb",
    "fname_protein": "./MPRO/protein.pdb",
    "fname_ligand": "./MPRO/subtrate.pdb"
}
```

Step 2. Add H atoms

- using AmberTools: conda install -c conda-forge ambertools

```
reduce abc.pdb > abc_H.pdb
```

- using pdb2pqr: pip install pdb2pqr

```
pdb2pqr30 --ff AMBER --with-ph=7 --nodebump
--ffout=AMBER abc.pdb abc.pqr
```

- using OpenMM:

Code

```
...
pdb = PDBFile('input.pdb')
modeller = Modeller(pdb.topology, pdb.positions)
modeller.addHydrogens(forcefield)
...
```

Step 3 and 5. Build an AMBER prmtop file

- download a zip file from https://github.com/swillow/pdb2amber

```
python pdb2amber.py -i input.json
```

input.json

```
{
    "fname_pdb": "dimer.pdb",
    "fname_prmtop": "dimer.prmtop",
    "fname_ff": [
        "./data/protein.ff14SB.xml",
        "./data/wat_opc3.xml"
    ]
}
```

Build an AMBER prmtop file: advanced

input.json

```
{
    "fname_pdb": "dimer.pdb",
    "fname_prmtop": "dimer.prmtop",
    "fname_ff": [
        "./data/protein.ff14SB.xml",
        "./data/wat_opc3.xml"
    ],
    "linked_residues": [
        ["FE1 FES","SG CYF"],
        ["FE2 FES","SG CYF"],
        ["P   FMN","OG1 THO"],
        ["FE FE","SG  CYG"]
    ]
}
```

Step 4. Build a system containing a protein, (membrane) water, and salt

```
python build_system.py -i input_build.json
```

input_build.json

```
{
    "fname_protein": "dimer.pdb",
    "protein_charge": -8,
    "fname_system": "dimer_wat.pdb",
    "box": [90,120,120],
    "membrane": {
        "fname_unit": "dppe_box15.pdb",
        "bool": false
    },
    "solvent": {
        "fname_unit": "water_box15.xyz",
        "bool": true,
        "NaCl(Molarity)": 0.1
    }
}
```

OpenMM Script Builder

- http://builder.openmm.org/

Minimization

```
python AmberOpenMM.py -i input_min.json
```

input_min.json

```
{
    "job": "min",
    "min": {
        "pdb": "input.pdb",
        "prmtop": "input.prmtop",
        "save_pdb": "minimized.pdb",
        "fc_pos": 500.0,
        "Platform": "OpenCL",
        "maxIteraction": 5000,
        "tolerance": 100.0
    }
}
```

NVT MD simulation

python AmberOpenMM.py -i input_nvt.json

input_step1_nvt.json

```
{
    "job": "nvt",
    "nvt": {
        "pdb": "minimized.pdb",
        "prmtop": "input.prmtop",
        "save_pdb": "step1.pdb",
        "save_state": "step1.rst",
        "stdout": "bomd_nvt_step1.dat",
        "dcd": "traj_step1.dcd",
        "fc_pos": 300.0,
        "dt_fs": 2.0,
        "Temperature": 300.0,
        "Platform": "OpenCL",
        "totalSteps": 125000,
        "nstdout": 1000,
        "ndcd": 5000
    }
}
```

input_step2_nvt.json

```
{
    "job": "nvt",
    "nvt": {
        "pdb": "step1.pdb",
        "prmtop": "input.prmtop",
        "save_pdb": "step2.pdb",
        "save_state": "step2.rst",
        "stdout": "bomd_nvt_step2.dat",
        "dcd": "traj_step2.dcd",
        "dt_fs": 2.0,
        "Temperature": 300.0,
        "Platform": "OpenCL",
        "totalSteps": 125000,
        "nstdout": 1000,
        "ndcd": 5000
    }
}
```

NPT MD simulation

input_step3_npt.json

```
{
    "job": "npt",
    "npt": {
        "pdb": "step2.pdb",
        "prmtop": "input.prmtop",
        "save_pdb": "step3.pdb",
        "load_state": "step2.rst",
        "save_state": "step3.rst",
        "stdout": "bomd_npt_step3.dat",
        "dcd": "traj_step3.dcd",
        "dt_fs": 2.0,
        "Temperature": 300.0,
        "Barostat" : "MonteCarloMembraneBarostat",
        "Pressure" : 1.0,
        "Platform": "OpenCL",
        "nstep_cycle": 125000,
        "ncycle": 20,
        "nstdout": 1000,
        "ndcd": 5000
    }
}
```

input_step4_npt.json

```
{
    "job": "npt",
    "npt": {
        "pdb": "step3_20.pdb",
        "prmtop": "input.prmtop",
        "save_pdb": "step4.pdb",
        "load_state": "step3_20.rst",
        "save_state": "step4.rst",
        "stdout": "bomd_npt_step4.dat",
        "dcd": "traj_step4.dcd",
        "dt_fs": 2.0,
        "Temperature": 300.0,
        "Barostat" : "MonteCarloBarostat",
        "Pressure" : 1.0,
        "Platform": "OpenCL",
        "nstep_cycle": 125000,
        "ncycle": 20,
        "nstdout": 1000,
        "ndcd": 5000
    }
}
```

Generate Ligand Force Fields

1. add hydrogen atoms

```
reduce ligand.pdb > ligand_H.pdb
```

2. check the molecular structure and identify the charge of the ligand
3. generate mol2 files (estimate the atomic point charges)

```
antechamber -fi pdb -fo mol2 -i ligand_H.pdb -o ligand_H.mol2 -c bcc -pf y -nc chg
```

4. generate frcmod

```
parmchk2 -i abc_H.mol2 -o abc_H.frcmod -f mol2
```

5. generate the parameter and topology file using a General Amber Force Field

```
tleap -s -f ligand.in
```

ligand.in

```
source leaprc.gaff2
mol = loadmol2 ligand_H.mol2
loadamberparams ligand_H.frcmod
saveamberparm mol ligand.prmtop ligand.inpcrd
quit
```

A minor problem of 'antechamber'

The sum of the atomic point charges does not become the total charge of the ligand.

Build a force field parameter file for OpenMM from an AMBER prmtop file

```
python write_xml_pretty  -i  input_ff.json
```

```
{
    "fname_prmtop": "n3i.prmtop",
    "fname_xml": "n3i.ff.xml",
    "ff_prefix": "n3i"
}
```

Potential Energy of the system consisting of QM molecules and MM molecules

$$E = E_{\mathrm{QM}} + E_{\mathrm{MM}} + E_{\mathrm{QM/MM}},$$

where $E_{\mathrm{QM/MM}}$ is the potential energy between QM and MM molecules.

- classical mechanics:

$$E_{\mathrm{QM/MM}} = \sum_{A \in \mathrm{QM}} \sum_{B \in \mathrm{MM}} \left(4\epsilon_{AB} \left[\left(\frac{\sigma_{AB}}{R_{AB}} \right)^{12} - \left(\frac{\sigma_{AB}}{R_{AB}} \right)^{6} \right] + \frac{q_A q_B}{R_{AB}} \right)$$

- embedded fragment method:

$$E_{\mathrm{QM}} + E_{\mathrm{QM/MM}} = \langle \Psi_{I:Q_I} | \hat{H}_{I:Q_I} | \Psi_{I:Q_I} \rangle + \sum_{A \in \mathrm{QM}} \sum_{B \in \mathrm{MM}} 4\epsilon_{AB} \left[\left(\frac{\sigma_{AB}}{R_{AB}} \right)^{12} - \left(\frac{\sigma_{AB}}{R_{AB}} \right)^{6} \right]$$

```
python run_qmmm.py -i input.json
```

input.json

```
{
    "theory": "qmmm", # qm-pol
    "job": "opt", # ener
    "qm": {
        "method": "scf",
        "basis": "6-31gs",
        "charge": 0,
        "fname_geom": "qm_step4.xyz",
        "fixed_atom_list": [1,16],
        "constraints_list": [[23,38,500,1.85]],
        "esp": true,
        "esp_opts": {"resp": true,"resp_hfree": true}
    },
    "mm": {
        "fname_prmtop": "mm_step4.prmtop",
        "fname_geom": "mm_step4.pdb",
        "fixed_atom_list": [2312,2314,2714,2716]
    }
}
```