Build a coarse-grained force field with OpenMM

Wei Lu

Apr 2021

Why OpenMM

- Has a high level wrapper that make the code platform independent.
  - One code, works both on CPU and GPU
- Has autograd (automatic gradient calculation), only the definition of Energy is needed.
  - The derivative with respect to coordniates are computed automatically.

What you need to run a simulation starting with a PDB file

1. Set up the system from a PDB file or Amber input.
2. Define the force fields.
3. Combine system, forcefield, and integrator.
4. Perform the simulation.
5. Analyze the results.

Setting up the System

Start from Amber or Gromac input file

```
prmtop = AmberPrmtopFile('input.prmtop')
inpcrd = AmberInpcrdFile('input.inpcrd')
system = prmtop.createSystem(nonbondedMethod=PME, nonbondedCutoff=1*nanometer,
        constraints=HBonds)
```

```
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)
```

Start from PDB

```
pdb = PDBFile("ca_only.pdb")
forcefield = ForceField("cg.xml")
system = forcefield.createSystem(pdb.topology)
connect = connect_term(system)
system.addForce(connect)
structure_based = structure_based_term(contact_list)
system.addForce(structure_based)
```

Example XML file (from PDB)

```xml
<ForceField>
 <AtomTypes>
  <Type name="tip3p-O" class="OW" element="O" mass="15.99943"/>
  <Type name="tip3p-H" class="HW" element="H" mass="1.007947"/>
 </AtomTypes>
 <Residues>
  <Residue name="HOH">
   <Atom name="O" type="tip3p-O"/>
   <Atom name="H1" type="tip3p-H"/>
   <Atom name="H2" type="tip3p-H"/>
   <Bond atomName1="O" atomName2="H1"/>
   <Bond atomName1="O" atomName2="H2"/>
  </Residue>
 </Residues>
 <HarmonicBondForce>
  <Bond class1="OW" class2="HW" length="0.09572" k="462750.4"/>
 </HarmonicBondForce>
 <HarmonicAngleForce>
  <Angle class1="HW" class2="OW" class3="HW" angle="1.82421813418" k="836.8"/>
 </HarmonicAngleForce>
 <NonbondedForce coulomb14scale="0.833333" lj14scale="0.5">
  <Atom type="tip3p-O" charge="-0.834" sigma="0.31507524065751241" epsilon="0.635968"/>
  <Atom type="tip3p-H" charge="0.417" sigma="1" epsilon="0"/>
 </NonbondedForce>
</ForceField>
```

Reference

https://github.com/npschafer/openawsem/blob/master/awsem.xml

https://github.com/cabb99/open3spn2/blob/master/open3SPN2/3SPN2.xml

https://github.com/openmm/openmm/blob/master/wrappers/python/openmm/app/data/tip3p.xml

Example XML file (from PDB)
Without bond definition

```
<ForceField>
 <AtomTypes>
  <Type name="tip3p-O" class="OW" element="O" mass="15.99943"/>
  <Type name="tip3p-H" class="HW" element="H" mass="1.007947"/>
 </AtomTypes>
 <Residues>
  <Residue name="HOH">
   <Atom name="O" type="tip3p-O"/>
   <Atom name="H1" type="tip3p-H"/>
   <Atom name="H2" type="tip3p-H"/>
  </Residue>
 </Residues>

</ForceField>
```

The definition of Atom

6.1.1. Atom Types and Atom Classes¶
Force field parameters are assigned to atoms based on their “atom types”. Atom types should be the most specific identification of an atom that will ever be needed. Two atoms should have the same type only if the force field will always treat them identically in every way.
Multiple atom types can be grouped together into “atom classes”. In general, two types should be in the same class if the force field usually (but not necessarily always) treats them identically. For example, the α-carbon of an alanine residue will probably have a different atom type than the α-carbon of a leucine residue, but both of them will probably have the same atom class.
All force field parameters can be specified either by atom type or atom class. Classes exist as a convenience to make force field definitions more compact. If necessary, you could define everything in terms of atom types, but when many types all share the same parameters, it is convenient to only have to specify them once.

More information at the section 6 of the openmm user guide.
http://docs.openmm.org/latest/userguide/application.html#creating-force-fields

The definition of Residue

6.1.2. Residue Templates¶

Types are assigned to atoms by matching residues to templates. A template specifies a list of atoms, the type of each one, and the bonds between them. For each residue in the PDB file, the force field searches its list of templates for one that has an identical set of atoms with identical bonds between them. When matching templates, neither the order of the atoms nor their names matter; it only cares about their elements and the set of bonds between them. (The PDB file reader does care about names, of course, since it needs to figure out which atom each line of the file corresponds to.)

Here we define the simplest case: a CA model

```xml
<!-- Definition for your Coarse-Grained atoms -->
<ForceField>
<!-- Specify atom types -->
<!-- name should be unique, and class could be more general, Example below -->
<!-- <Type name="tip3p-O" class="OW" element="O" mass="15.99943"/> -->
<AtomTypes>
    <Type name="CA" class="C" element="C" mass="12.01078"/>
</AtomTypes>

<!-- Setup residue templates -->
<!-- use type to match the atom type, and the element to match the residue, example below -->
<!-- <Atom name="O" type="tip3p-O"/> -->
<Residues>
    <Residue name="RES">
    <Atom name="Res" type="CA"/>
    </Residue>
</Residues>

</ForceField>
```

Read in the XML

```python
from simtk.openmm.app import ForceField

forcefield = ForceField("cg.xml")
```

Prepare the input file

```python
from pdbfixer import PDBFixer
from simtk.openmm.app import PDBFile

fixer = PDBFixer("1r69.pdb")
# more on pdbfixer, check:
# https://htmlpreview.github.io/?https://github.com/openmm/pdbfixer/blob/master/Manual.html
fixer.removeHeterogens(keepWater=False)
PDBFile.writeFile(fixer.topology, fixer.positions, open('1r69_cleaned.pdb', 'w'))
```

```python
import mdtraj
pdb = mdtraj.load("1r69_cleaned.pdb")
keep_list = []
for atom in pdb.topology.atoms:
    if atom.name == "CA":
        keep_list.append(atom.index)
chosen = pdb.atom_slice(keep_list)
chosen.save("ca_only.pdb")
```

ca_only.pdb

```
REMARK   1 CREATED WITH MDTraj 1.9.4, 2021-04-19
CRYST1   32.800   37.500   44.600  90.00  90.00  90.00 P 1           1
MODEL        0
ATOM      1  CA  SER A   1     -10.028  -9.538  22.229  1.00  0.00           C
ATOM      2  CA  ILE A   2     -10.398  -6.112  20.640  1.00  0.00           C
ATOM      3  CA  SER A   3     -13.366  -5.095  22.843  1.00  0.00           C
ATOM      4  CA  SER A   4     -11.428  -5.660  26.066  1.00  0.00           C
ATOM      5  CA  ARG A   5      -8.264  -4.075  24.581  1.00  0.00           C
ATOM      6  CA  VAL A   6     -10.186  -0.999  23.381  1.00  0.00           C
ATOM      7  CA  LYS A   7     -11.909  -0.521  26.747  1.00  0.00           C
ATOM      8  CA  SER A   8      -8.621  -0.998  28.604  1.00  0.00           C
ATOM      9  CA  LYS A   9      -6.756   1.611  26.546  1.00  0.00           C
ATOM     10  CA  ARG A  10      -9.472   4.249  26.488  1.00  0.00           C
ATOM     11  CA  ILE A  11      -9.636   4.024  30.314  1.00  0.00           C
ATOM     12  CA  GLN A  12      -5.870   4.408  30.637  1.00  0.00           C
ATOM     13  CA  LEU A  13      -6.141   7.537  28.469  1.00  0.00           C
ATOM     14  CA  GLY A  14      -9.099   8.870  30.471  1.00  0.00           C
ATOM     15  CA  LEU A  15     -11.617   8.633  27.594  1.00  0.00           C
ATOM     16  CA  ASN A  16     -15.306   7.725  27.837  1.00  0.00           C
```

Three ways of defining the same force field

```python
from simtk.openmm import HarmonicBondForce
def connect_term(system):
    k_con= 10000
    con = HarmonicBondForce()
    n = system.getNumParticles()
    for i in range(n-1):
        con.addBond(i, i+1, 0.3816, k_con)
    return con
```

```python
from simtk.openmm import CustomBondForce
def connect_term_v2(system):
    k_con= 10000
    r0 = 0.3816
    con = CustomBondForce(f"0.5*{k_con}*(r-r0)^2")
    n = system.getNumParticles()
    con.addPerBondParameter("r0")
    for i in range(n-1):
        con.addBond(i, i+1, [r0])
    return con
```

```python
from simtk.openmm import CustomCompoundBondForce
def connect_term_v3(system):
    k_con= 10000
    r0 = 0.3816
    con = CustomCompoundBondForce(2, f"0.5*{k_con}*(distance(p1,p2)-r0)^2")
    n = system.getNumParticles()
    con.addPerBondParameter("r0")
    for i in range(n-1):
        con.addBond([i, i+1], [r0])
    return con
```

Default Units

Quantity	Units
distance	nm
time	ps
mass	atomic mass units
charge	proton charge
temperature	Kelvin
angle	radians
energy	kJ/mol

http://docs.openmm.org/latest/api-python/generated/simtk.openmm.openmm.CustomBondForce.html#simtk.openmm.openmm.CustomBondForce

Define another force field.

```python
# contact map
import numpy as np
from simtk.unit import *

pdb = PDBFile("ca_only.pdb")
pos = pdb.positions.value_in_unit(nanometer)
pos = np.array(pos)
dis = (((pos.reshape(1, -1, 3) - pos.reshape(-1, 1, 3))**2).sum(axis=-1))**0.5
```

```python
n = dis.shape[0]
contact_threshold = 0.8 # in unit of nm
contact_list = []
for i in range(n):
    for j in range(i+1, n):
        dis_ij = dis[i][j]
        if dis_ij < contact_threshold:
            sigma_ij = 0.1*(j-i)**0.15
            contact_list.append((i, j, (dis_ij, sigma_ij)))
```

```python
len(contact_list)
```

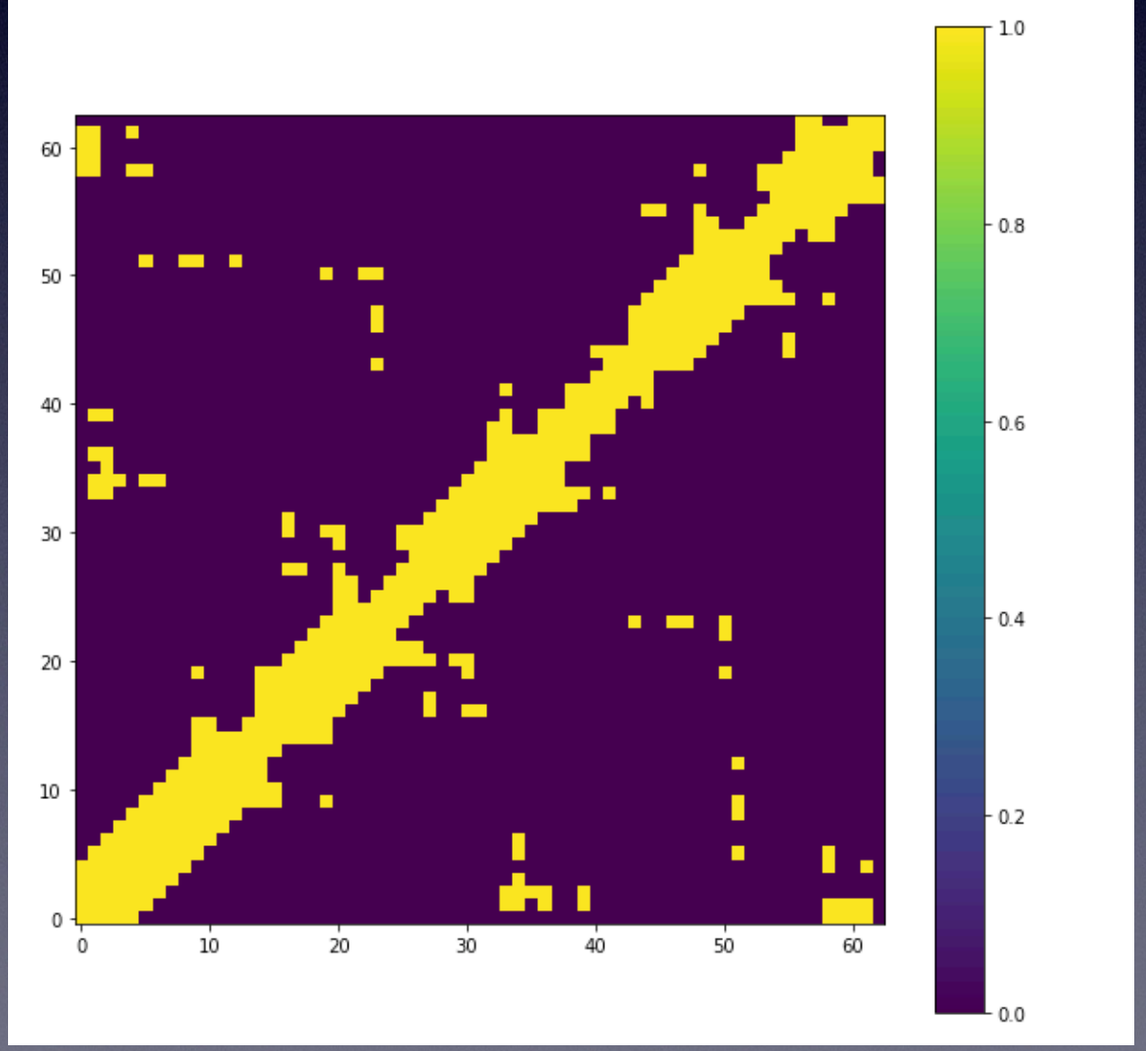
```
275
```

```python
from simtk.openmm import CustomBondForce
def structure_based_term(contact_list):
    k = 10
    structure_based = CustomBondForce(f"-{k}*exp(-(r-r_ijN)^2/(2*sigma_ij^2))")
#     structure_based = CustomBondForce(f"-{k}")
    structure_based.addPerBondParameter("r_ijN")
    structure_based.addPerBondParameter("sigma_ij")
    for contact in contact_list:
        structure_based.addBond(*contact)
    return structure_based
```

```python
import matplotlib.pylab as plt
%matplotlib inline
```

```python
plt.figure(figsize=[10,10])
plt.imshow(dis < 0.8, origin="lower")
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7f9bef4e95f8>
```



Combine system, forcefield, and integrator.

```
from simtk.openmm import LangevinIntegrator
from simtk.openmm import CustomIntegrator
from simtk.openmm.app import Simulation
from simtk.openmm.app import PDBReporter
from simtk.openmm.app import StateDataReporter
from simtk.openmm.app import DCDReporter
from sys import stdout


pdb = PDBFile("ca_only.pdb")
forcefield = ForceField("cg.xml")

print(pdb.topology)

system = forcefield.createSystem(pdb.topology)
system.removeForce(0)   # remove the default force "CMotionRemover"
# connect = connect_term(system)
# system.addForce(connect)

# connect = connect_term_v2(system)
# system.addForce(connect)

connect = connect_term_v3(system)
system.addForce(connect)

structure_based = structure_based_term(contact_list)
system.addForce(structure_based)

print("Number of particles: ", system.getNumParticles())
print("Number of forces: ", system.getNumForces())

integrator = LangevinIntegrator(300*kelvin, 1/picosecond, 0.004*picoseconds)
simulation = Simulation(pdb.topology, system, integrator)
simulation.context.setPositions(pdb.positions)
```

```
<Topology; 1 chains, 63 residues, 63 atoms, 0 bonds>
Number of particles:  63
Number of forces:  2
```

Perform the simulation.

```
integrator = LangevinIntegrator(300*kelvin, 1/picosecond, 0.004*picoseconds)
simulation = Simulation(pdb.topology, system, integrator)
simulation.context.setPositions(pdb.positions)
simulation.reporters.append(DCDReporter('output.dcd', 1000, append=True))
simulation.reporters.append(StateDataReporter(stdout, 1000, step=True,
        potentialEnergy=True, temperature=True))
simulation.step(10000)
```

```
#"Step","Potential Energy (kJ/mole)","Temperature (K)"
1000,-2464.26416015625,278.7213749109017
2000,-2458.77099609375,362.43019795568466
3000,-2444.4677734375,291.95222452623176
4000,-2497.534423828125,249.1159317567878
5000,-2494.3193359375,301.37449363057584
6000,-2445.806884765625,237.61744366634323
7000,-2480.2412109375,286.44012580228656
8000,-2493.31640625,320.2336396749569
9000,-2498.3935546875,293.1181628227125
10000,-2471.962158203125,311.41793281739996
```

Compute the energy for the initial structure

```
integrator = CustomIntegrator(0.001)
simulation = Simulation(pdb.topology, system, integrator)
simulation.context.setPositions(pdb.positions)
simulation.reporters.append(DCDReporter('output.dcd', 1))
simulation.reporters.append(StateDataReporter(stdout, 1, step=True,
        potentialEnergy=True, temperature=True))
simulation.step(int(1))
simulation.minimizeEnergy()
simulation.step(int(1))

integrator = LangevinIntegrator(300*kelvin, 1/picosecond, 0.004*picoseconds)
simulation = Simulation(pdb.topology, system, integrator)
simulation.context.setPositions(pdb.positions)
simulation.reporters.append(DCDReporter('output.dcd', 1000, append=True))
simulation.reporters.append(StateDataReporter(stdout, 1000, step=True,
        potentialEnergy=True, temperature=True))
simulation.step(10000)
```

```
#"Step","Potential Energy (kJ/mole)","Temperature (K)"
1,-2749.25732421875,0.0
2,-2749.87255859375,0.0
#"Step","Potential Energy (kJ/mole)","Temperature (K)"
1000,-2464.26416015625,278.7213749109017
2000,-2458.77099609375,362.43019795568466
3000,-2444.4677734375,291.95222452623176
4000,-2497.534423828125,249.1159317567878
5000,-2494.3193359375,301.37449363057584
6000,-2445.806884765625,237.61744366634323
7000,-2480.2412109375,286.44012580228656
8000,-2493.31640625,320.2336396749569
9000,-2498.3935546875,293.1181628227125
10000,-2471.962158203125,311.41793281739996
```

Analyze the results.

```
1 view = nglview.show_structure_file("ca_only.pdb")
2 view
```

```
1 traj = mdtraj.load_dcd("output.dcd", top="ca_only.pdb")
```

```
1 view = nglview.show_mdtraj(traj)
2 view
```

Analysis, Energy evaluation

```python
# energy evaluation.
pdb = PDBFile('ca_only.pdb')
traj = mdtraj.load_dcd("output.dcd", top='ca_only.pdb')

integrator = CustomIntegrator(0.001)
simulation = Simulation(pdb.topology, system, integrator)
for frame in range(traj.n_frames):
    simulation.context.setPositions(traj.openmm_positions(frame))
    state = simulation.context.getState(getEnergy=True)
    termEnergy = state.getPotentialEnergy().value_in_unit(kilojoule_per_mole)
#     termEnergy = state.getPotentialEnergy()
    print(frame, f"{termEnergy:.3f} kJ/mol")
```

```
0 -2749.257 kJ/mol
1 -2749.873 kJ/mol
2 -2507.495 kJ/mol
3 -2500.490 kJ/mol
4 -2472.177 kJ/mol
5 -2527.056 kJ/mol
6 -2477.699 kJ/mol
7 -2480.507 kJ/mol
8 -2532.085 kJ/mol
9 -2523.294 kJ/mol
10 -2519.302 kJ/mol
11 -2456.396 kJ/mol
```


Thank You!