Parametrization of coarse-grained protein models based on experimental data

Iryna Zaporozhets
Clementi group

CTBP - openMM workgroup
June 02, 2021

Observable-driven Design of Effective Molecular Models (ODEM)

$$V = V_{bond}(r, k_b) + V_{angle}(r, k_a) + V_{dihedral}(r, k_d) + V_{nonbonded}(r, \epsilon)$$

Input:

Parameter guess $\{\epsilon_i^{(0)}\}$

Experimental observables $\{f_k\}$

Estimate:

Equilibrium distribution π

$$\pi^{(1)} = \pi^{(0)} \exp[-\beta \Delta V(r, \epsilon)]$$

Observable values

$$e_k = \sum_i \pi_i(\epsilon) e_{ki}$$

Maximize:

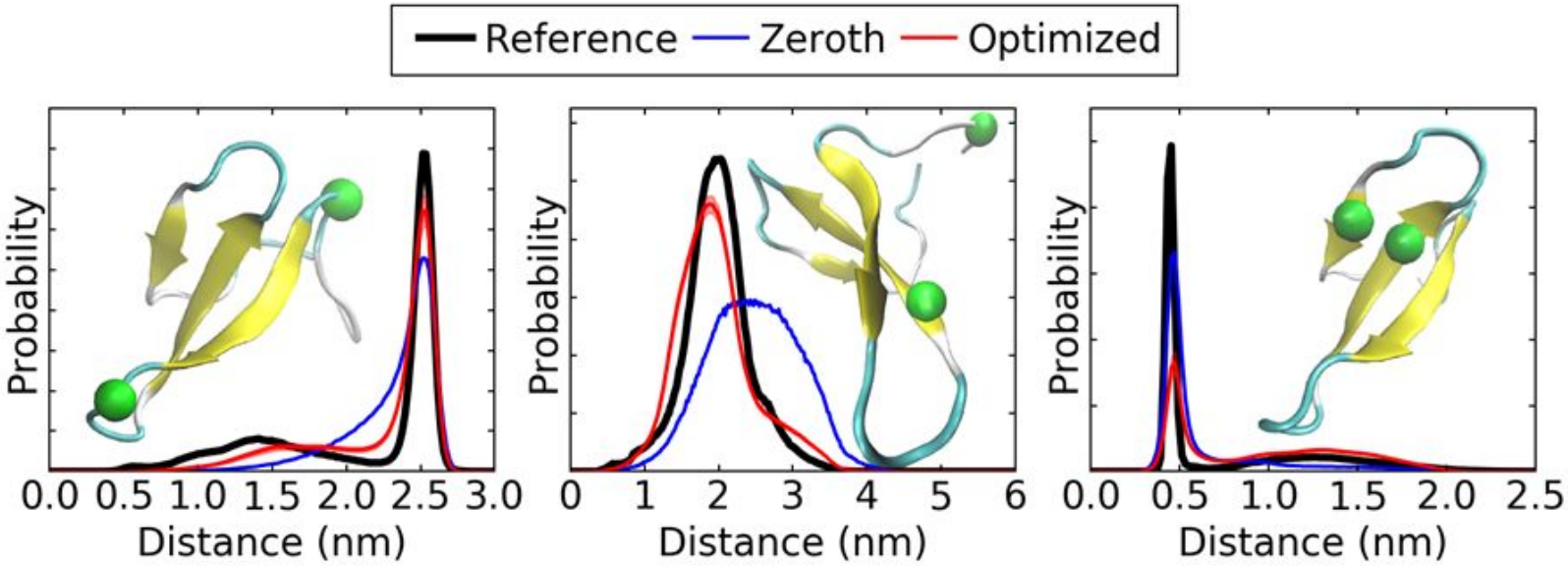
$$Q = \prod_{k=1}^{N} exp\left[-\frac{(e_k(\epsilon) - f_k)^2}{2\Delta f_k^2}\right]$$

Output:

Set of optimal parameters $\{\epsilon_i^{(1)}\}$

J. Chen, J. Chen, G. Pinamonti, C. Clementi *J. Chem. Theory Comput.*, **2018**, *14* (7), pp 3849–3858

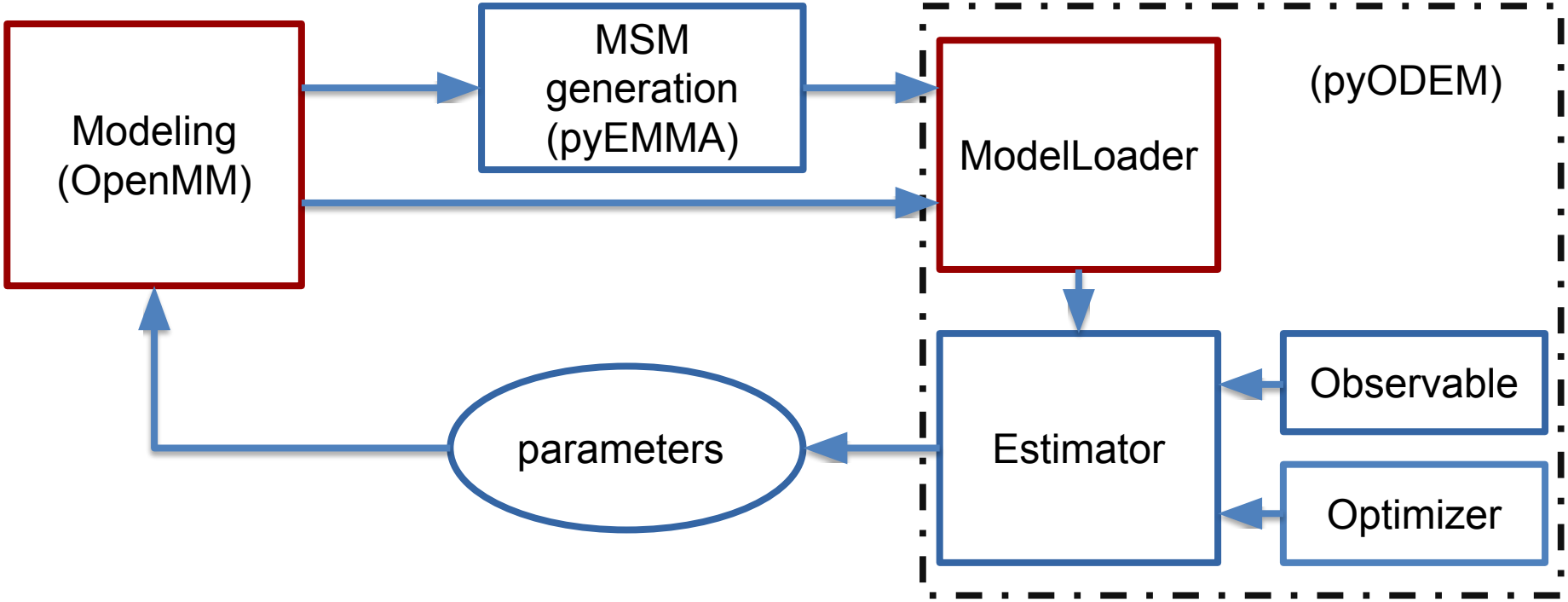
ODEM application: distance distribution as observable



- Optimized model better reproduces tryptophan fluorescence and structure of the transition state ensemble

J. Chen, J. Chen, G. Pinamonti, C. Clementi *J. Chem. Theory Comput.*, **2018**, *14* (7), pp 3849–3858

Observable-driven optimization of CG forcefield parameters: Software perspective



https://github.com/ClementiGroup/pyODEM

Customizing system using existing GROMACS files: setup I

```
import configparser
from datetime import datetime
import numpy as np
import simtk.openmm.app as app
import simtk.unit as u
import simtk.openmm as mm

print(datetime.now().ctime())
print("Simulation run with OPENMM version ",mm.version.version)

config_name = 'config.ini'
config = configparser.ConfigParser()
config.read(config_name)

gro_file = config['model']['gro_file']
top_file = config['model']['top_file']
model_params = config['model']['parameter_file']
pairwise_params = config['model']['pairwise_file']

...
```

Import

Logging

Loading configuration parameters

Customizing system using existing GROMACS files: setup II

```
def get_force_by_name(system, force_name):
    """
    Return a list of force references in the system that
have a name specified by the force_name
    The force name must be specified exactly as in OpenMM
documentation.
    ...
    """
    forces = system.getForces()
    forces_to_return = [ ]
    for force in forces:
        if type(force).__name__ == force_name:
            forces_to_return.append(force)
    return forces_to_return
```

Customizing system using existing GROMACS files: System I

```
gro = app.GromacsGroFile(gro_file)
top = app.GromacsTopFile(top_file)
system = top.createSystem()

attractive_forces = mm.openmm.CustomBondForce("((epsilon +
(((excl / r)^12))) * (1 - exp(-0.5 * (((r-rnot) /
sigma_g)^2)))) - epsilon")
attractive_forces.addPerBondParameter("excl")
... # add parameters also for "rnot", "sigma_g", "epsilon"


# Defining repulsive interactions energy function
repulsive_forces = mm.openmm.CustomBondForce("((excl / r)^12)
- (epsilon * (tanh((rnot - r + sigma_t) / sigma_t) + 1)) / 2
")
repulsive_forces.addPerBondParameter("excl")
... # add parameters also for "rnot", "sigma_t", "epsilon"
```

Customizing system using existing GROMACS files: System II

```
forces = {'LJ12GAUSSIANTANH': repulsive_forces,
          'LJ12GAUSSIAN': attractive_forces}

# Get nonbonded interactions to be modified:
nonbonded_force = get_force_by_name(system,"NonbondedForce")[0]
custom_nonbonded_force = get_force_by_name(system, "CustomNonbondedForce")[0]

parameters = np.loadtxt(model_params)
pairs = np.genfromtxt(pairwise_params, dtype=None, encoding='utf-8')
for pair in pairs:
    force_type = pair[3]
    atom_0 = int(pair[0]-1)
    atom_1 = int(pair[1]-1)
    forces[force_type].addBond(atom_0,atom_1,
                               [pair[4],pair[5],pair[6],parameters[pair[2]]])
    nonbonded_force.addException(atom_0, atom_1, *exception_params)
    custom_nonbonded_force.addExclusion(atom_0, atom_1)
```

Customizing system using existing GROMACS files: System III

```
system.addForce(attractive_forces)
system.addForce(repulsive_forces)

with open('system.xml', 'w') as f:
    system_xml = mm.XmlSerializer.serialize(system)
    f.write(system_xml)
```

Customizing system using existing GROMACS files: Simulation

```
integrator = mm.openmm.LangevinIntegrator(temp, gamma, dt)
platform_properties = {"DeviceIndex" : str(device_id)}
simulation = Simulation(top.topology, system, integrator, platform,
platform_properties)
simulation.context.setPositions(gro.positions)
simulation.minimizeEnergy()
e_pot_after = simulation.context.getState(getEnergy=True).getPotentialEnergy()
current_reporter = PDBReporter("structure_after_minimization.pdb",1)
current_reporter.report(simulation,
simulation.context.getState(getPositions=True))
simulation.context.setVelocitiesToTemperature(temp)
simulation.reporters.append(...)
simulation.step(nsteps)
```

Extending pyODEM with a new model: ModelLoader

```
class CustomProteinDihedral(ModelLoader):
    """
    A child class of ModelLoader. Loads data from trajectory, calculate energy
and derivatives
    """
...
    def load_data(self, traj):
        ...

    def get_epsilons(self):
        ...
```

Extending pyODEM with a new model: ModelLoader

```
def get_potentials_epsilon(self, data):
    """
    Generate two functions, that can calculate Hamiltonian and Hamiltonian
    derivatives.
    ...
    Returns
    -------
    hepsilon : function,  takes model parameters and calculates -beta*H for
each frame.
     dhepsilon : function, takes model parameters and calculates  derivative
of H with respect to them.                    """
   # TO DO: Implement integration with openMM models.
```

```
print(“Thank you for your attention!”)
```

Acknowledgments

Clementi group

Prof. Cecilia Clementi

Nick Charron
Andrea Guljas
Tim Hempel
Felix Musil

Fabio Trovato
WangFei Yang
David Rosenberger
Clark Templeton