Molecular Dynamics

Dr. Daniel A. Nissley, Dr. Broncio Aguilar-Sanjuan

Doctoral Training Centre – University of Oxford

Molecular Dynamics Tutorials

Thursday the 2nd of December, 2021

Abbreviations

MD: Molecular dynamics

AA: All-atom

CG: Coarse-grain

GUI: graphical user interface

Overview

Molecular dynamics (MD) is a broad field with many applications in chemistry, physics, and biology. While this tutorial will be focused on the application of MD to biological systems, it is a general framework that can be applied in principle to any molecular system.

Due to computational limitations, this tutorial begins with a short all-atom (AA) example but then quickly moves into the realm of coarse-grain (CG) simulations. As discussed in lecture, CG systems provide greatly increased simulation speed while preserving essential features, allowing access to longer timescale processes. This is especially useful in the area of protein folding, which occurs on a timescale of seconds to minutes in real systems.

This tutorial consists of the following steps:

0. Installing required Python packages with Conda
1. Introduction to OpenMM – first simulation
2. Visualizing your simulation in PyMol
3. Creating a topology-based coarse-grain model
4. Temperature quenching simulations

Note: in the following tutorial, lines of Python code are colored **blue**. Lines that correspond to commands meant to be run on your terminal's command line are preceded by the '>' character and are in **bold**.

Note: the majority of this tutorial assumes that you are running OpenMM from a Linux console. You can, in principle, install OpenMM on a Windows machine as well (*e.g.*, install Anaconda3, open Spyder4, and use pip to install openmm package). In the interests of time, inform the instructor ASAP if you are not on Linux. We may be able to accommodate your use of Windows or pair you with a student on Linux.

0. Installing required Python packages with Conda

First things first, we need to setup OpenMM and a few other packages before running any simulations. The easiest way to do this is using Conda to create a new environment with all of the things we need:

>conda create -n md-sims python=3.8

>conda activate md-sims

>conda install numpy

>conda install -c conda-forge openmm

Note: if these commands don't work, you probably need to source your conda.sh file, which should be at "anaconda3/etc/profile.d/conda.sh" – you just need to determine where anaconda3 was installed.

1. Introduction to OpenMM – first simulation

Begin by navigating to

http://docs.openmm.org/latest/userguide/application.html#running-simulations

in your web browser of choice. This provides a quick guide on how to run an MD simulation in OpenMM.

Open a terminal and then copy and paste the code from the website's Example 3.1 into a blank text file named simulateVillin.py and then save the file. Feel free to use whichever text editor you prefer – if you are not familiar with any text editors, let a demonstrator know and they will help you get started.

This simulation script requires a single input file, input.pdb, which represents the 35-residue Villin headpiece protein domain in explicit water. The OpenMM User Guide directs you to find this file in the example/ directory – instead, use the file "input.pdb" in the "1_practical" directory of the files provided to you for this practical. To access the files in the practical directory you will need to extract it:

>tar -xvf filename.tar

Read through the User Guide section following Example 3.1 and add comments to your code describing what each line of the script does and what each of the parameters means in the context of our simulation.

In the lines

```
simulation.reporters.append(PDBReporter('output.pdb', 1000))
simulation.reporters.append(StateDataReporter(stdout, 1000, step=True,
                                              potentialEnergy=True, temperature=True))
```

change the number 1000 to 100 so that data will be saved to file and printed to screen more frequently.

We also want to update the script to give us some information about run time. To do this, add the line

```
from datetime import datetime
```

to the top of your program and then insert the line

```
startTime = datetime.now()
```

directly before the command that actually runs the AA MD simulation (which should be apparent from reading the User Guide). Finally, add the line

```
print('Execution time:', datetime.now()-startTime)
```

to the end of the program. This will print the approximate run time of the simulation. Now, run your simulation by executing the command.

>python simulateVillin.py > output.txt

This will take a few minutes. Once the 10,000th integration time step has been simulated the run will terminate automatically. Write down or otherwise record the execution time of the simulation.

- Comment on the temperature as a function of time during the simulation (stored in output.txt). Is it constant? Fluctuating? Is this what you expected? Generating plots in Python or another software of your choice is recommended.
- Comment on the potential energy of the system as a function of time during the simulation (stored in output.txt) in the same way as you did for temperature.
- Change the timestep to 0.010 ps, the frequency for saving data to 1 integration time step, and the name of the output file from 'output.pdb' to 'output2.pdb' and then rerun using the command "**>python simulateVillin.py > output2.txt".** What happens? Speculate on why.

Villin is one of the fastest folding proteins known, with an experimental folding time of ~5 μs. (So-called supervillin, a mutant, folds in ~700 ns and is considered the fastest known folder; more information and the relevant literature can be found at https://www.blopig.com/blog/2021/06/how-fast-can-a-protein-fold/#more-6979).

- For how long did you simulate the dynamics of Villin in your first run? This can be calculated directly from the simulation parameters.
- Based on the amount of computer processor time required to carry out this simulation, how long would it take to simulate the $U \rightarrow F$ transition of Villin on this computer (on average)?
- Villin is a very fast-folding protein. How long would it take to simulate the folding of a large globular protein with a mean folding time on the order of 1 s?

Note: Setting up an AA MD simulation typically requires many steps, including solvation, ionization, heating, minimization, and equilibration. This is an often time-consuming and tedious process that requires a great deal of practical knowledge not covered by this tutorial (see, for example, the steps listed at https://www.charmmtutorial.org/index.php/Full_example). The simple example used here begins with a structure that has already been run through these steps and is therefore ready for production dynamics.

2. Visualizing your simulation in PyMol

One of the worst assumptions you can make when performing MD simulations is that everything went according to plan just because the simulation terminated normally. MD packages are just complex programs. While they have many hundreds of error checks to try and make sure things run correctly or terminate aberrantly (*e.g.*, the "Exception: Particle coordinate is nan" message you likely received in the last section), sometimes non-physical interactions that invalidate the simulation results will slip through.

Perhaps the best way to verify a simulation is through visualization – this will allow you catch a vast array of problems that are not apparent from either the error messages during the run or from analysis of the output coordinate information.

In this tutorial we will visualize our simulations in PyMol.

The output.pdb file you generated in your initial run contains 100 snapshots of the system coordinates printed every 100 integration time steps. These snapshots are typically referred to as 'frames' in the MD field in common parlance. You can load all of these frames in time order into PyMol with the command

>pymol output.pdb

It will take a few seconds for all of the frames to be loaded. Once complete, you can play the simulation back as a movie by hitting the play button on the bottom right of the PyMol GUI.

- Reconcile your comments on the potential energy versus time during the simulation with the apparent state of the protein. Viewing the native state of Villin headpiece on the Protein Data Bank may be informative.

Now let's take a look at the failed simulation from the previous step of the practical. Close PyMol and then run the command

>pymol output2.pdb

- What do you see in this simulation? You should change the visual representation to one that allows you to see all bonds and atoms explicitly.

Note: other programs, including Visualize Molecular Dynamics (VMD), are available to visualize your MD simulations. VMD has a vast array of functionality ranging from simple visualization to data analysis and video rendering that make it an invaluable tool for MD. However, VMD has a steep learning curve, meaning that learning VMD is time well spent only if you need these enhanced functionalities. If you expect to use MD in future research projects I highly recommend learning VMD.

3. Creating a topology-based coarse-grain model

As you saw first hand in part 1 of this practical, AA simulations are extremely expensive and, in most cases, cannot be run on the timescales necessary to observe protein folding (assuming the AA forcefield is capable of producing the native state at all *a priori*). In this section you will consider a CG protein model, learning about its forcefield and running a simple simulation.

We will be using a model constructed based on the structure 1yrf_chain_a.pdb, which represents Villin headpiece domain.

The C-alpha CG model was already generated for you. The file gen.out contains details of the model generation.

You should also see the following set of 4 files:

1yrf.cor

1yrf.psf

1yrf_eta1.586.xml

1yrf_secondary_structure.stride

Examine the model generation output written to gen.out. A C-alpha CG model represents each amino acid with a single interaction site placed at the atomic coordinates of the C-alpha atom. In this case we have used the topology of the native state to parameterize a native-centric model that contains information about the contacts formed in the native tertiary structure. Load the 1yrf.cor and 1yrf.psf files into PyMol alongside 1yrf_chain_a.pdb to visualize the CG model.

- Using PyMol, check some of the distances reported in the gen.out file. For example, in the "Final native contact summary:' section, choose one of the reported distances and see if it matches the actual distance between the C-alpha interaction sites.

Now that you have generated the required input files, let's run a short single-temperature simulation to make sure everything is working correctly.

A simulation script, singleT.py, is provided. Open and examine this script before running the simulation. Where does the actual "singleT" function come from?

- What is the integration time step?
- What other differences between this simulation script and the one used for AA simulation of Villin can you identify?

To run this script, enter the command

>./singleT.py 1yrf.psf 1yrf.cor 1yrf_eta1.586.xml 10000 output3

This should run very quickly.

- For how much time did you just simulate dynamics for this CG protein model? Compare and contrast with the AA simulations.

- Is it reasonable to assume that 1 ns of dynamics for this CG model corresponds to 1 ns of dynamics in real time? If not, do you think 1 ns of dynamics for the CG model corresponds to more or less real time?
- Assuming the CG simulation has dynamics 4,000,000x faster than real systems, for how many time steps would you need to run your simulation to simulate the equivalent of the mean folding time (5 μs) of villin? How long do you estimate this would require on your laptop?