Week 4: Many Molecules

2MMN40: Introduction to Molecular Modeling and Simulation
Last update: November 24, 2020

Contents

1 The Assignment

In the last three weeks you have developed a simulator for a single molecule with up to three atoms. This weeks assignment is to extend that simulator to allow for many molecules. The assignment consists of only one part

1. Adapt your code so it can simulate many molecules. This entails:
   - Creating some sort of topology to keep track of all the bonds, angles and force constants.
   - Rewriting the force calculations (depending on you implementation)
   - Adapting the integrator (depending on you implementation; if you followed the implementation tip from last week, this should be easy, maybe not even necessary)

This is by far the most challenging programming part of this course. If you do not succeed this week with the implementation do not despair. Next week we will add interactions between molecules and periodic boundary conditions, but there will also be time to continue working on this.

2 MD Topologies

We saw in week 1 that the .xyz files don't contain information about bonds and angles etc. In the last few weeks you probably specified the bonds and angles by hand in your code. This is very unpractical for larger systems. Professional MD codes define a *topology* to deal with this problem. The word *topology* might lead to confusion. It does not mean the formal mathematical topology. It is used for its informal definition, i.e. a topology describes the spatial relation between elements of a set. The set in our case is the set of atoms, the spatial relations are the bonds and angles (and, in a few weeks, dihedrals and pairs of molecules).

To simulate multiple molecules, you will need to create your own topology. There are two parts to this. First of all you need to create an input file that contains your systems topology. The second part is representing this topology in your code so you can use it to do actual calculations.

2.1 Topology Files

Their are multiple possibilities for defining a topology file (you can google this if you like). Here I will show you a simple topology file format that is based on the GROMACS .itp topology file format (but not the same). It is advanced enough to handle almost all molecules, but simple enough such that it is as easy as a .xyz file to read and write with python.

A .xyz file for two water molecules. You can visualize it with VMD if you want.

```
6
Some comment
O 2.25 2.25 2.25
H 1.25 2.25 2.25
H 2.25 1.9245 1.30445
O 2.25 2.25 6.75
H 1.25 2.25 6.75
H 2.25 1.9245 5.80445
```

A corresponding topology file for the two molecules

```
bonds 4
0 1 5024.16 0.9572
0 2 5024.16 0.9572
3 4 5024.16 0.9572
3 5 5024.16 0.9572
angles 2
1 0 2 628.02 1.8242181
4 3 5 628.02 1.8242181
```

This file format consists of *keywords*, the `bonds` and `angles`, a number specifying how many of the keyword type things there are, the `4` and `2`, and the actual bonds and angles. The latter are described by the indices that take part in the bond or angle followed by the constants needed in the corresponding potential. As an example the first line in bonds should be read as: there is a bond between atom 0 and atom 1 with a force constant of 5024.16 and equilibrium length of 0.9572.

Two remarks:

- Realise that this is just one of the many options for defining a topology file, you can choose your own way of doing this. For example you might want to add information about which atoms belong to which molecule.
- While working on this topology file and the implementation keep in mind that you eventually want to simulate a couple of hundred molecules.

2.2 Topology in your code

To use the topology you also need to make a representation of the topology in your code. A representation means something that makes bonds, angles and parameters accessible to the code. The most simple way of doing this (and actually also a very decent way) is simply loading the data from the topology file into NDArrays. For the bonds you could do something like this

```
# you should not type the data in, but load it from the file, this is just for
↪  illustrating the possible representation of the bonds in your code.
bonds = np.array([[0, 1],
                  [0, 2],
                  [3, 4],
                  [3, 5]])
constants = np.array([[5024.16, 0.9572],
                      [5024.16, 0.9572],
                      [5024.16, 0.9572],
                      [5024.16, 0.9572]])
```

Given this representation of the topology we can use NumPy tricks to compute, for example, the forces very quickly.

2.2.1 NumPy Tricks

This is probably a good time to look at our NumPy tutorial again (or look at it for the first time), especially the *Advanced Indexing*, *Advanced NumPy Trickery* and *Buffered vs Unbuffered Addition in NumPy* sections are relevant.

In the advanced indexing section you have seen that you can access NumPy arrays with indexing arrays. Note that both columns of the `bonds` variable are indices. To get the first column we would type `bonds[:,0]` and to get all the positions corresponding to these indices we could simply type `coords[bonds[:,0]]` (assuming your atom positions are stored in `coords`). Take a bit of time to realise what the implications are and how you can use this feature to do, for example, all bond length calculations or angle calculations at the same time. To make the last statement a bit more explicit think of what this piece of code would do

```
dr = coords[bonds[:,0]] - coords[bonds[:,1]].
```

2.2.2 Alternative Approach (Optional)

The previous approach for the representation probably leads to procedural programming. It is the most natural way of programming and relatively easy to understand. If you have experience with Object Oriented Programming (OOP) then it might have crossed your mind that the topology is something that is well suited to be contained in a class, with some nice functions that give you the bonds, angles etc. If you like OO and know it already, you could try to write your own topology class and use that instead. Note, however, that there probably won't be any benefit from a performance or "code niceness" perspective. But if you like OO give it a try.

3 Performance Tip

Try to avoid for-loops while scaling to many atoms and molecules. Use NumPy tricks instead, try to write simple computations in one line of code. For example you could apply the `np.linalg.norm()` function to the `dr` array with the right `axis` argument to compute all bond distances in one line of code.

It should in principle be possible to do all force calculations for bonds and angles without for-loops. Where you do need for-loops is probably while reading and writing files and for the integrator (maybe a while-loop). In most other places you can do without.

4 Initial Conditions

Just as in week 3 you need initial conditions to run the simulation. You will need a .xyz file with all the positions of the atoms. You can generate one yourself. You already have a single water molecule from the last few weeks, you can copy it as many times as you want, be sure to shift the molecules each time, so that they don't overlap and you have a starting point for a simulation with many molecules. In the same way you can generate the topology file, create it for one molecule and then shift everything accordingly. These two tasks can easily be automated by a python script.

5 Sanity Check

After everything is implemented be sure to visualize your multi-molecule system in VMD to check if what you simulate makes sense. Also, are your units still correct?