MD Simulation and Analysis in a Notebook

February 4, 2019

1 Molecular Dynamics Simulation and Analysis in a Notebook

Python is the perfect glue language, with big emphasis on readability and an awesome ecosystem of packages for all kind of areas, ranging from web development or machine learning to the most interesting science. In this short tutorial, we will focus on molecular dynamics as provided by the Omnia project.

Omnia is a compilation of diverse packages for chemistry simulations, but its main pillar is OpenMM: "a toolkit for molecular simulation using high performance GPU code". While written in C++/CUDA/OpenCL, it also offers rich APIs in Python. A good number of several other packages use OpenMM or provide additional functionalities, such as pdbfixer (preparation and cleaning), mdtraj or pytraj (analysis), nglview (visualization), parmed (converters between different MD programs) or openmoltools (a bit of everything, actually).

We will use all of them at some point of the tutorial, outlined here:

Part A

1. Prepare an environment for MDs with conda
2. Download, clean and prepare structures from PDB
3. Set up an OpenMM MD simulation from scratch

Part B

1. Preview the MD movie in the notebook with nglview
2. MD Analysis with pytraj or mdtraj
3. Custom clustering with the scipy stack

---

2 A.1 - Prepare the environment

First, create a new Python 3.7 environment and name it freely.

```
conda create -n openmm python=3.7
```

The following packages will be required along the tutorial:

- openmm
- mdtraj

- pdbfixer
- nglview
- parmed
- openmoltools

However, they are not provided in the standard (`defaults`) `conda` channels like `numpy` or `ipython`. They are hosted in the `omnia` channel, so we need to specify that with the `-c` switch:

```
!conda install -y -c omnia openmm mdtraj pdbfixer parmed openmoltools
```

`nglview` is not up-to-date in the omnia channel, but it is in `bioconda`:

```
!conda install -y -c bioconda nglview
!jupyter-nbextension enable nglview --py --user # only if it does not work
```

That's it! Now, try to import this packages to test if they are correctly installed.

3 A.2 - Download and prepare your structure

Disclaimer: This tutorial is not meant as a MD crash course, but as an exercise to play with Python capabilities in science. Hence, I am not trying to be very cautious about the MD simulation itself. We will use small structures that will allow us to progress rapidly with a modest computer. Remember, we don't care about the MD, only the setup process!

To run a proper MD, we need to download the structure from some place (PDB, for example), but that's only the first step. Sometimes, the structures are not complete, or include wrong residue names. Some include non-natural residues! Depending on the origin of the files, hydrogens must be added, and depending on which kind of simulation we want to run, we might need to solvate the structure (ie, surround the molecule with a box of waters). While that can be easily attained with Chimera or another GUI programs, we are going to use Python tools... Just because.

Another benefit of learning this procedure is that you could automatize the same pipeline for hundreds of files... :)

So... First! Download the structure. `parmed` comes handy here.

```
In [1]: import parmed as pmd
        mol = pmd.download_PDB('1rfo')
        mol.visualize()
```

```
NGLWidget(count=10)
```

`1RFO` is a trimer. Let's suppose we only want to run the MD with one of the monomers... Let's download the PDB with `mdtraj` (alternative possibility), meant for trajectory analysis, but also features a handy selection language.

```
In [3]: import mdtraj as md
        trimer = md.load_pdb('http://www.rcsb.org/pdb/files/1RFO.pdb')
        trimer.topology # evaluate to obtain relevant info
```

```
/home/jrodriguez/.local/anaconda/envs/openmm/lib/python3.5/site-packages/mdtraj/formats/pdb/pdb
  warnings.warn('Unlikely unit cell vectors detected in PDB file likely '
```

```
Out[3]: <mdtraj.Topology with 3 chains, 81 residues, 1296 atoms, 1314 bonds at 0x7f3acd0237b8>
```

```
In [5]: # Select only the first chain
        indices = trimer.topology.select('chainid 0')
        monomer = trimer.atom_slice(indices)
        monomer
```

```
Out[5]: <mdtraj.Trajectory with 10 frames, 432 atoms, 27 residues, without unitcells at 0x7f3a
```

```
In [6]: # Check everything is OK
        import nglview as nv
        nv.show_mdtraj(monomer)
```

```
NGLWidget(count=10)
```

```
In [7]: # Do we have hydrogens already?
        monomer.top.select('element H')
```

```
Out[7]: array([  4,   5,   6,   7,   8,  21,  22,  23,  24,  25,  26,  27,  28,
                29,  38,  39,  40,  41,  42,  43,  44,  45,  46,  47,  48,  56,
                57,  58,  59,  60,  61,  62,  72,  73,  74,  75,  76,  77,  83,
                84,  85,  86,  87,  95,  96,  97,  98,  99, 100, 101, 113, 114,
               115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 134, 135,
               136, 137, 142, 143, 144, 154, 155, 156, 157, 158, 159, 160, 161,
               167, 168, 169, 170, 171, 184, 185, 186, 187, 188, 189, 190, 191,
               192, 200, 201, 202, 203, 204, 205, 206, 207, 208, 220, 221, 222,
               223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 242, 243, 244,
               245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 263, 264, 265,
               266, 271, 272, 273, 283, 284, 285, 286, 287, 288, 303, 304, 305,
               306, 307, 308, 309, 310, 311, 312, 320, 321, 322, 323, 324, 325,
               326, 327, 328, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346,
               347, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 373,
               374, 375, 376, 377, 385, 386, 387, 388, 389, 390, 391, 403, 404,
               405, 406, 407, 408, 409, 410, 411, 421, 422, 423, 424, 425, 426,
               427, 428, 429, 430, 431])
```

So far, so good. Looks like the structure is ready! If it weren't, `pdbfixer` could be applied.

4 A.3 - Set up an openMM calculation!

The `mdtraj` objects contain all the necessary information about the molecule we need: the topology and its coordinates. We only need to choose a forcefield and then some technical details of the simulation itself.

In [8]:
```
from simtk import unit
import simtk.openmm as mm
from simtk.openmm import app
from sys import stdout

# Molecule details - atoms, connectivity and positions
topology = monomer.top.to_openmm()
positions = monomer.xyz[0] # we only want the first 'frame' of the molecule; ie, one o

# Simulation details - the 'rules' of the game
# In openMM, a simulation is built with a topology, its corresponding system and the i
forcefield = app.ForceField('amber99sbildn.xml', 'amber99_obc.xml')
system = forcefield.createSystem(topology, nonbondedMethod=app.CutoffNonPeriodic)
integrator = mm.LangevinIntegrator(300*unit.kelvin, 1.0/unit.picoseconds, 1.0*unit.fem
simulation = app.Simulation(topology, system, integrator)

print('Setting initial positions')
simulation.context.setPositions(positions)

print('Minimizing...')
simulation.minimizeEnergy()

# What kind of info do we want back?
simulation.reporters.append(app.DCDReporter('trajectory_.dcd', 1000)) # the movie file
# some realtime data on the progress of the sim
simulation.reporters.append(app.StateDataReporter(stdout, 1000, step=True,
    potentialEnergy=True, temperature=True, progress=True, remainingTime=True,
    speed=True, totalSteps=1e5, separator='\t'))

simulation.context.setVelocitiesToTemperature(300*unit.kelvin) # Assign random but coh

# Run the thing alrady!
simulation.step(1e5)
```

```
Setting initial positions
Minimizing...
#"Progress (%)"        "Step"        "Potential Energy (kJ/mole)"        "Temperature (K)"
1.0%        1000        -3912.419189453125        255.1403017417056        0        --
2.0%        2000        -3799.124755859375        283.11515822247424        582        0:14
3.0%        3000        -3674.298095703125        308.12782733174953        601        0:13
4.0%        4000        -3684.833740234375        295.1534053639397        607        0:13
5.0%        5000        -3806.055908203125        306.961528638708        609        0:13
6.0%        6000        -3845.111328125        310.67067889418325        594        0:13
7.0%        7000        -3695.0908203125        306.81679170725835        596        0:13
8.0%        8000        -3758.9013671875        298.61258497957044        597        0:13
9.0%        9000        -3632.8154296875        291.52349107897453        597        0:13
10.0%        10000        -3664.41162109375        310.9290248895392        597        0:13
11.0%        11000        -3773.478515625        299.13939652405037        597        0:12
```

12.0%	12000	-3664.970703125	302.67442193913104	578	0:13
13.0%	13000	-3647.97314453125	312.99146268667556	566	0:13
14.0%	14000	-3760.64306640625	301.47680143539657	553	0:13
15.0%	15000	-3659.640625	298.40482426590074	543	0:13
16.0%	16000	-3750.9033203125	277.9339111008682	546	0:13
17.0%	17000	-3806.254150390625	284.0794876891146	549	0:13
18.0%	18000	-3604.528564453125	262.35879989021754	553	0:12
19.0%	19000	-3780.028564453125	286.2416137589961	555	0:12
20.0%	20000	-3801.136474609375	278.3202014015979	558	0:12
21.0%	21000	-3708.656982421875	280.03189411956623	560	0:12
22.0%	22000	-3708.0107421875	293.7886462700308	563	0:11
23.0%	23000	-3730.06884765625	298.99313393513603	564	0:11
24.0%	24000	-3742.56982421875	297.9574054846186	559	0:11
25.0%	25000	-3722.419921875	323.73305548131947	551	0:11
26.0%	26000	-3617.74560546875	292.7293363326272	553	0:11
27.0%	27000	-3720.60595703125	301.28902700372026	554	0:11
28.0%	28000	-3748.8408203125	303.5723739547896	556	0:11
29.0%	29000	-3812.727294921875	305.0808673594874	558	0:10
30.0%	30000	-3764.34228515625	302.3649755923602	560	0:10
31.0%	31000	-3810.065185546875	307.59988129365286	562	0:10
32.0%	32000	-3784.51025390625	292.8560812028919	559	0:10
33.0%	33000	-3735.146240234375	299.8589668476414	554	0:10
34.0%	34000	-3635.647216796875	301.9056408189937	555	0:10
35.0%	35000	-3709.749755859375	305.7724598008482	557	0:10
36.0%	36000	-3749.156494140625	286.90512788032356	558	0:09
37.0%	37000	-3663.1640625	288.4916065505724	558	0:09
38.0%	38000	-3713.674072265625	287.4947694374799	560	0:09
39.0%	39000	-3584.599365234375	287.7820896037873	561	0:09
40.0%	40000	-3719.69091796875	323.62614172969216	562	0:09
41.0%	41000	-3732.34033203125	295.8140972196611	563	0:09
42.0%	42000	-3640.459716796875	303.4353928001559	563	0:08
43.0%	43000	-3711.54296875	291.8394150426518	564	0:08
44.0%	44000	-3589.1650390625	321.9818072301102	565	0:08
45.0%	45000	-3743.32861328125	298.22388433135427	567	0:08
46.0%	46000	-3579.72021484375	300.98306477226555	564	0:08
47.0%	47000	-3599.2138671875	316.8408960366888	560	0:08
48.0%	48000	-3631.72265625	288.4392452718049	561	0:08
49.0%	49000	-3744.54345703125	301.7210317571156	561	0:07
50.0%	50000	-3602.35546875	286.6430120842868	558	0:07
51.0%	51000	-3638.01611328125	308.24828160853536	556	0:07
52.0%	52000	-3708.8935546875	301.1719370932537	556	0:07
53.0%	53000	-3670.3193359375	306.5778730223913	556	0:07
54.0%	54000	-3594.435302734375	296.1965466072362	557	0:07
55.0%	55000	-3605.58447265625	317.4166602610022	557	0:06
56.0%	56000	-3571.251953125	300.05646006130894	555	0:06
57.0%	57000	-3693.7705078125	326.96076663080675	552	0:06
58.0%	58000	-3616.3310546875	306.1399818627573	553	0:06
59.0%	59000	-3607.896728515625	324.30227994775635	554	0:06

60.0%	60000	-3580.207275390625	323.2232175404115	555	0:06
61.0%	61000	-3733.04296875	315.6978294230763	556	0:06
62.0%	62000	-3597.271484375	286.337971251071	557	0:05
63.0%	63000	-3600.22998046875	320.4957958455029	558	0:05
64.0%	64000	-3745.9501953125	313.3981271958507	558	0:05
65.0%	65000	-3749.006591796875	293.731460360096	556	0:05
66.0%	66000	-3765.334228515625	304.55847225352295	557	0:05
67.0%	67000	-3741.2890625	318.0459479759179	558	0:05
68.0%	68000	-3640.86083984375	295.0653850342389	558	0:04
69.0%	69000	-3794.205322265625	319.1932311386324	558	0:04
70.0%	70000	-3812.51318359375	303.9289241386815	555	0:04
71.0%	71000	-3709.5771484375	302.74293024556636	555	0:04
72.0%	72000	-3694.13720703125	298.9777397176467	553	0:04
73.0%	73000	-3723.738037109375	305.2532880261148	551	0:04
74.0%	74000	-3718.926513671875	289.4929347645235	551	0:04
75.0%	75000	-3621.745849609375	275.26951273857145	552	0:03
76.0%	76000	-3721.079345703125	300.0220166488373	552	0:03
77.0%	77000	-3686.07421875	322.401279183664	553	0:03
78.0%	78000	-3676.48095703125	303.5737968912384	553	0:03
79.0%	79000	-3772.57373046875	288.22530072694883	554	0:03
80.0%	80000	-3672.38623046875	287.6955539439126	555	0:03
81.0%	81000	-3779.01123046875	312.84405692644555	555	0:02
82.0%	82000	-3691.008544921875	297.29186298282525	556	0:02
83.0%	83000	-3744.86328125	311.2703921177584	557	0:02
84.0%	84000	-3881.615234375	302.53777058695607	557	0:02
85.0%	85000	-3761.157470703125	304.693427401345	556	0:02
86.0%	86000	-3812.009033203125	286.83626821717576	554	0:02
87.0%	87000	-3852.99169921875	298.76031468816933	554	0:02
88.0%	88000	-3816.6220703125	326.86271404809065	553	0:01
89.0%	89000	-3714.221435546875	311.8507415408242	551	0:01
90.0%	90000	-3670.736083984375	311.7926654921615	552	0:01
91.0%	91000	-3675.901611328125	296.88794503423776	552	0:01
92.0%	92000	-3704.190185546875	326.6872316309446	553	0:01
93.0%	93000	-3805.680908203125	286.27176420213243	553	0:01
94.0%	94000	-3803.348876953125	297.9689803738306	554	0:00
95.0%	95000	-3710.503173828125	283.4268993607863	554	0:00
96.0%	96000	-3760.884033203125	282.8875948260259	553	0:00
97.0%	97000	-3700.34228515625	310.73530654127245	551	0:00
98.0%	98000	-3706.4619140625	302.8961849651263	552	0:00
99.0%	99000	-3640.35205078125	299.320620823075	553	0:00
100.0%	100000	-3736.70947265625	276.3179006480767	553	0:00

5 Play the trajectory

MDTraj is really useful for this kind of stuff!

```
In [20]: # Did we got it right?
         traj = md.load_dcd('trajectory_.dcd',  top=monomer.top)
         nv.show_mdtraj(traj)
```

```
NGLWidget(count=100)
```

```
In [23]: traj = traj.superpose(traj)
```

```
In [27]: rmsds = md.rmsd?
```

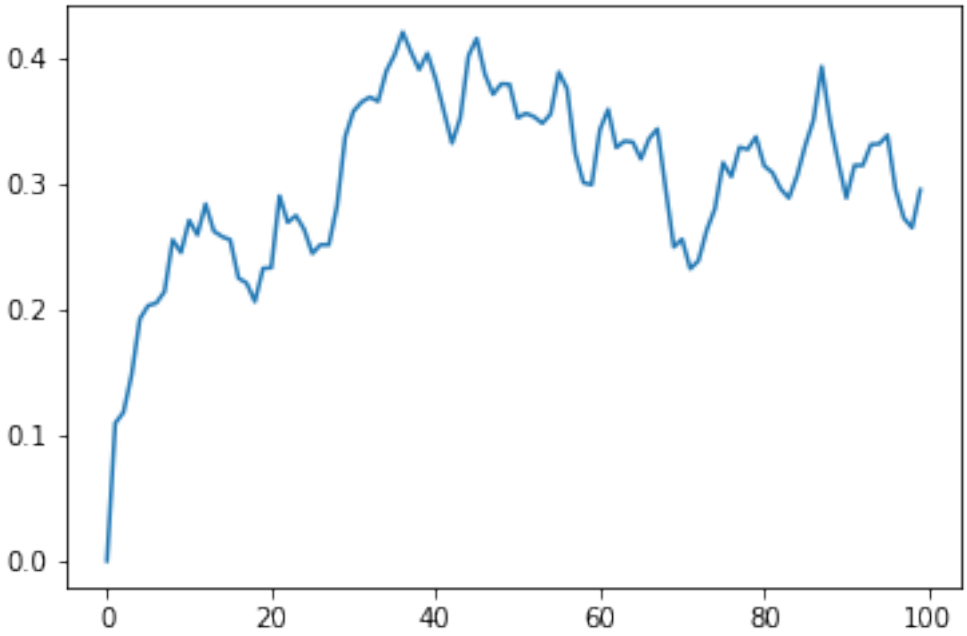
```
In [ ]: rmsds = md.rmsd
```

```
In [28]: rmsds = md.rmsd(traj, traj)
         rmsds
```

```
Out[28]: array([0.        , 0.10928389, 0.11783434, 0.1477965 , 0.19208157,
                0.20257197, 0.20463501, 0.21371627, 0.25483242, 0.24471691,
                0.27025995, 0.25876102, 0.2832585 , 0.2617139 , 0.25740445,
                0.25476924, 0.2244741 , 0.22038634, 0.20579784, 0.23220263,
                0.23265488, 0.28972477, 0.2686142 , 0.27404445, 0.26306516,
                0.24396637, 0.25100985, 0.25079134, 0.28160116, 0.33680338,
                0.35687238, 0.364234  , 0.36811307, 0.3647071 , 0.3888069 ,
                0.40152544, 0.41966155, 0.40384197, 0.3899522 , 0.40278998,
                0.38256994, 0.35712883, 0.33154985, 0.35154703, 0.40081313,
                0.4147309 , 0.3863013 , 0.37033686, 0.37862107, 0.37823653,
                0.35164106, 0.35520852, 0.3524708 , 0.3472838 , 0.35470998,
                0.3879469 , 0.37487093, 0.32339868, 0.30009985, 0.29856536,
                0.34217384, 0.35833737, 0.32796285, 0.3331824 , 0.3324482 ,
                0.3189772 , 0.3352538 , 0.3428761 , 0.29553968, 0.24914207,
                0.25530547, 0.23184885, 0.23824449, 0.2624519 , 0.279393  ,
                0.3160544 , 0.30498043, 0.32833785, 0.32663763, 0.33662003,
                0.31355458, 0.30816355, 0.2953405 , 0.28799576, 0.30564123,
                0.3297034 , 0.34991273, 0.39237761, 0.34933305, 0.3171404 ,
                0.28803524, 0.3139718 , 0.3138669 , 0.3305046 , 0.33096147,
                0.33784673, 0.2945271 , 0.27212813, 0.26439694, 0.29454005],
               dtype=float32)
```

```
In [29]: from matplotlib import pyplot as plt
         %matplotlib notebook
```

```
In [31]: p = plt.plot(rmsds)
```


0.4
0.3
0.2
0.1
0.0
0
20
40
60
80
100