Run an MD simulation

A few of notes on the format.

- Any time I refer to a *local* terminal, this means a terminal navigating your personal computer. When I refer to a *cluster* terminal, this means a terminal navigating the cluster. When you open a new terminal, it is a *local* terminal. After you type `ssh USERNAME@login.talapas.uoregon.edu`, it becomes a *cluster* terminal. (Put functionally: if you type the command `ls` on a *local* terminal, it shows files on your laptop; if you type `ls` on a *cluster* terminal, it shows files on the cluster.)
- In this document, anything in "`this font`" is text that will appear on a terminal (whether typed or printed out as output).
- Anything in "`ALL CAPS`" is a placeholder name you will likely change. For example, the instructions refer to "`PROTEIN.pdb`". You will probably have a file with a different and more descriptive name, but we will refer to it `PROTEIN.pdb` in the instructions.
- In the terminal commands, any text after a "`#`" is a comment. It is not part of a command, but can be safely pasted into a terminal because the computer will ignore any text after `#`.

I. Prepare the computing environment

You should only have to do these steps once, before running your first MD simulation.

1. Copy the `setup_md` folder to the cluster. On a local terminal, navigate to the directory that has your run_md directory. Then type the following. This will copy the run_md directory to your home directory.

```
scp -r setup_md USERNAME@login.talapas.uoregon.edu:
```

2. SSH onto the cluster with the following command. After this runs, your terminal becomes a *cluster* terminal.

```
ssh USERNAME@login.talapas.uoregon.edu
```

3. Navigate to the `setup_md` directory:

```
cd setup_md
```

3. Edit the "run_md.srun" and "extract_run.srun" files to use the correct PIRG.

```
nano run_files/run_md.srun
```

```
nano run_files/extract_run.srun
```

The top of these files look something like the following. Alter `CHANGEME` to your PIRG (harmslab, etc.).

```
#!/bin/bash -l
#SBATCH --account=CHANGEME      ### change this to your actual account
for charging
#SBATCH --job-name=gmx_run      ### job name
#SBATCH --output=hostname.out   ### file in which to store job stdout
#SBATCH --error=hostname.err    ### file in which to store job stderr
#SBATCH --partition=computelong
#SBATCH --time=07-00:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=28
```

II. Prepare a structure for a simulation

Obtain a structural model for the system you want to simulate. This can come from the RCSB, AlphaFold, docking simulations, or other methods. The details of what pre-processing needs to be done differ depending on inputs. Rather than giving a specific protocol, here is a set of approaches to use.

1. Be very careful preparing your input structure. It is super annoying to discover you did something stupid after running a simulation for a month. Some important questions:
   - Is the amino sequence correct and from the most useful organism to compare to experiments?
   - Does the structure have all of the domains you need to capture the biology?
   - Is the starting conformation reasonable for revealing the biology you hope to study?

   - Does the structure have extra stuff that should be taken out (a His-tag or signal peptide, for example)?
   - After prepping your structure, show the structure to a lab mate and ask "Does it look okay to you?"

2. Work in PDB files, not mmCIF files. mmCIF is now the standard on the RCSB; however, simulation software still uses PDB files. These can still be downloaded from RCSB.

3. Familiarize yourself with the PDB format, as it is often way easier to edit a file in a text editor than to figure out how to do something in PyMOL.

4. Remove as many atoms as you can. This includes small molecules from crystallization like PEG, non-specific ions, waters, and phosphates (unless functionally/structurally critical). You can either delete these in PyMOL or manually edit the PDB file in a text editor.

5. Consider trimming disordered N- and C-terminal extensions. Sometimes these are important for function, but they also slow simulations down *a lot* because they require a larger box. Let the biology inform your choice here.

6. Make sure every residue number is unique by adding offsets to each chain. (I've set up the `setup_md.py` script to do this automatically). Chain labels are lost when running a simulation. The chains are not lost topologically (i.e., the physics engine knows there are different chains), but it can be annoyingly hard to select residues for analysis after a run. For example, imagine you ran a simulation with chains A and B, each having resides 1-100. After pulling down the results, there is no easy way to select only chain A or B. Any time you select "residue 5", the software selects residue 5 from both chains at once. 🤯 To prevent this headache, I indicate the chain by adding offsets of 1,000. In the example above, chain A would be residues 1-100, chain B would be residues 1,001-1,100. A hypothetical chain C would go from 2,001-2,100.

7. If you are working from an experimental structure downloaded from the RCSB and find it is missing residues, has missing atoms, has undesired mutations, etc., use AlphaFold to generate a new version of the structure with the full sequence. This is the fastest (and probably most accurate) method to fill in any missing bits before the simulation.

8. If you need to load ligands or ions into an AlphaFold structure where a crystal structure with a docked ligand already exists, you can do so by aligning the crystal structure to the AlphaFold structure. You can then save out a PDB file with the protein structure from AlphaFold and the ligands from the crystal structure. The PyMOL "Mutagenesis wizard..." is a great tool to remove egregious clashes with the ligand because it allows you to search through protein side chain rotamers. (This clash removal does not have to be perfect, as the software will also remove clashes later).

III. Set up the simulation

9. Copy the PDB file of your model to the cluster. On a *local* terminal, type:

```
scp -r PROTEIN.pdb USERNAME@login.talapas.uoregon.edu:
```

2. SSH onto the cluster:

```
ssh USERNAME@login.talapas.uoregon.edu
```

3. On the cluster terminal, move the PDB file with your protein a directory within your "shared" folder. **All simulations should be run within the `~/shared` directory**. The `~/shared/` folder is actually a network drive with over a petabyte of space. Your home directory, by contrast, will rapidly run out of disk space.

   The following commands create a directory called `SIMS` on your shared drive and move your uploaded `PROTEIN.pdb` into that directory. Feel free to use more useful names than `SIMS` and `PROTEIN.pdb`.

```
cd shared/           # Move to the shared drive
mkdir SIMS           # Make a directory to hold simulations
cd SIMS              # Go into the sims directory
mv ~/PROTEIN.pdb .   # Move the PROTEIN.pdb from your home directory to
here
```

4. The next step is to add waters and ions using GROMACS. Because this will take some computational horsepower, we should start up an interactive session on a compute node on the cluster rather than running on the login node. To do so, run the following command, replacing MYPIRG with the PIRG for your account:

```
srun --account=MYPIRG --partition=compute --pty bash
```

5. Because we are going to use GROMACS, we need to load the gromacs module. This will make gromacs available as a command. Run the following:

```
module load gromacs/2023.4
```

6. Set up the calculation with the following command. This will create a directory called `md-input` that can be used to run an MD simulation. It uses the CHARMM36 forcefield and the .mdp run files from "run_md/ff". This command assumes you placed the `run_md` directory in your home directory when you set up the computing environment. If you put it somewhere else, alter the path accordingly.

```
~/setup_md/setup_md.py md-input --standard PROTEIN.pdb
```

- When this is run, a variety of information is written to the terminal output. There are two key things to inspect. The first is the assignment of histidine protonation state. GROMACS does this by analyzing hydrogen bond networks. Each histidine in the protein will either be assigned HIS (neutral, no proton), HISD (positive one charge, protonated at N^δ) or HISE (positive one charge, protonated at N^ε). I usually use the default assignment, but it is important to know that the software is making this decision.

```
Analysing hydrogen-bonding network for automated assignment of histidine
 protonation. 40 donors and 47 acceptors were found.
There are 60 hydrogen bonds
Will use HISE for residue 9
```

- The second output is the assignment of CYS REDOX state. In the following output, the software detected three disulfide bonds. Make sure these are reasonable!

```
8 out of 8 lines of specbond.dat converted successfully
Special Atom Distance matrix:
                   CYS3    CYS6    HIS9   CYS10   CYS19   CYS24
                   SG19    SG42   NE268    SG74   SG139   SG178
    CYS6    SG42   0.793
    HIS9   NE268   1.477   0.939
   CYS10    SG74   1.274   0.648   1.003
   CYS19   SG139   0.202   0.777   1.562   1.228
   CYS24   SG178   0.766   0.202   1.123   0.636   0.701
   CYS26   SG191   1.150   0.464   0.958   0.202   1.105   0.460
Linking CYS-3 SG-19 and CYS-19 SG-139...
Linking CYS-6 SG-42 and CYS-24 SG-178...
Linking CYS-10 SG-74 and CYS-26 SG-191...
Opening force field file ./charmm36-jul2022.ff/aminoacids.arn
Before cleaning: 1100 pairs
Before cleaning: 1115 dihedrals
```

- If you need to assign specific protonation states or disulfide bonds, you may need to run `gmx pdb2gmx` manually.
- **NOTE**: If you are running a calculation with custom ligand forcefield parameters, you will need to modify this command to something like the following, where `DIRECTORY` is the custom ligand parameter directory. For instructions on how to generate custom parameters, see the "Generate custom molecule parameters" protocol.

```
~/setup_md/setup_md.py md-input --standard PROTEIN.pdb --custom
DIRECTORY/model-to-pose.pdb --params DIRECTORY/gromacs/
```

7. After this command runs, the `md-input/` directory will be populated with several subdirectories. The `final` directory has everything we need to run a simulation. I like to copy this to a directory called `template` that I can just copy again if I want to do replicate runs. I then create a copy of `template` called `run000` with my first replicate. If I wanted to do another run, I could copy `template` to `run001`, etc. (Note: each simulation randomly assigns initial velocities to each atom, so the outputs will be different even starting from the exact same starting structure).

```
cp md-input/final template
cp -r template run000
```

8. You can now leave the interactive session on the compute node.

```
exit
```

9. Next, download the file `template/system.gro` to your local computer to inspect and make sure it looks reasonable. Type the following on a *local* terminal. This will copy system.gro from the cluster to the current directory on your computer.

```
scp USERNAME@login.talapas.uoregon.edu:shared/SIMS/template/system.gro .
```

10. Open `system.gro` in VMD or PyMOL and make sure it looks right. It should have your protein centered in a cube of waters with ions sodium and chloride atoms randomly scattered about. If you added a custom ligand, that ligand should be in place. This is the final, final check to make sure things look good before running your calculation.

IV. Run an MD simulation

1. Log into the cluster and navigate to your simulation directory in the shared folder.

```
ssh USERNAME@login.talapas.uoregon.edu
cd ~/shared/SIMS
```

2. Navigate into the directory where you are going to do the run:

```
cd run000
```

3. Make sure that your "run_md.srun" file is configured correctly.

```
nano run_md.srun
```

The top of the file will look like.

```
#!/bin/bash -l
#SBATCH --account=CHANGEME      ### change this to your actual account
for charging
#SBATCH --job-name=gmx_run      ### job name
#SBATCH --output=hostname.out   ### file in which to store job stdout
#SBATCH --error=hostname.err    ### file in which to store job stderr
#SBATCH --partition=computelong
#SBATCH --time=07-00:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=28
```

- Make sure that `CHANGEME` is the correct PIRG (harmslab, etc.)
- Make sure `--time` matches how long you think this should run. The less time you request, the faster the job scheduler will give you time: only request what you think you'll need. If you are going to run less than 24 hr, set `--partition=compute` rather than `--partition=computelong`.
- Depending on what you are simulating, you may want to update `--nodes`, `--ntasks-per-node` and/or `--cpus-per-task`. These defaults should be fine for relatively small systems. Tuning them is a can of worms that goes outside of this basic protocol.

3. Once you are satisfied with the .srun file, start the run with `sbatch`:

```
sbatch run_md.srun
```

4. Make sure it starts using `qstat`. Once the job launches the `Use` column in the output will switch from `Q` (queued) to `R` (running). Depending on how much compute time you requested, how many other jobs you already have running, and the current server load, it can take anywhere from seconds to days for the job to start.

```
qstat -u USERNAME
```

5. Once the job is running, you can now follow its output. The `run_md.srun` file does three preliminary calculations, often taking a couple of hours, then launches the main MD simulation. You can check the current status of the directory by typing:

```
ls -ltr
```

This command shows the files in the directory sorted by time modified. The bottom-most file in the output was the last modified. As the calculation proceeds, the files "em.log", "nvt.log", "npt.log", and "md_0_1.log" will sequentially appear as the last file. These are log files continuously updated by GROMACS as it runs each of these calculations. You can follow calculation progress using `tail`. The following command will follow the MD log file as it grows, allowing you to track the accumulation of simulation time.

```
tail -f md_0_1.log
```

Lines like the following are repeated over and over. The key information you probably care about is the "Time" column (30 ps in this case).

```
           Step           Time
          15000       30.00000

   Energies (kJ/mol)
           Bond            U-B    Proper Dih.  Improper Dih.      CMAP
Dih.
    2.77785e+02    1.07439e+03    1.26644e+03    7.46330e+01
-6.91381e+01
          LJ-14     Coulomb-14        LJ (SR)   Coulomb (SR)   Coul.
recip.
    2.30127e+02    7.84404e+03    1.75495e+04   -2.02206e+05
6.43167e+02
 Position Rest.      Potential    Kinetic En.   Total Energy  Conserved
En.
    2.16364e+02   -1.73098e+05    3.11099e+04   -1.41989e+05
-1.57647e+05
    Temperature Pressure (bar)   Constr. rmsd
    2.97052e+02   -4.16466e+02    2.58260e-06
```

Note that `tail -f` will go on forever. To break out of it and free up your terminal, type `CTRL-C`. (Note for macOS users: `CTRL` is really `control`, NOT the Mac `command` key).

If all has gone well, you have now set up a simulation that will run for the specified amount of simulation time or until you run out of allocated computer time, whichever comes first. You can log into the cluster at any time and check the current status by typing `tail -f md_0_1.log`.

Extending the simulation

If you run out of compute time before the simulation finishes, or if you want to extend the simulation longer than you specified in your initial .MDP file, you can edit the `run_md.sbatch` file. When you launched the job, it runs four calculations (EM, NVT, NPT, then MD).

```
gmx grompp -f em.mdp -c system.gro -p topol.top -o em.tpr
gmx mdrun -v -deffnm em

gmx grompp -f nvt.mdp -c em.gro -r em.gro -p topol.top -o nvt.tpr
gmx mdrun -deffnm nvt

gmx grompp -f npt.mdp -c nvt.gro -r nvt.gro -t nvt.cpt -p topol.top -o
npt.tpr
gmx mdrun -deffnm npt

gmx grompp -f md.mdp -c npt.gro -t npt.cpt -p topol.top -o md_0_1.tpr
gmx mdrun -deffnm md_0_1

# To extend by 1 us, comment out stuff above and uncomment below
#gmx convert-tpr -s md_0_1.tpr -extend 1000000 -o next.tpr
#gmx mdrun -s next.tpr -cpi md_0_1.cpt  -deffnm md_0_1 -append
```

To extend the simulation, comment out the lines running the initial calculations and un-comment the extension line:

```
#gmx grompp -f em.mdp -c system.gro -p topol.top -o em.tpr
#gmx mdrun -v -deffnm em

#gmx grompp -f nvt.mdp -c em.gro -r em.gro -p topol.top -o nvt.tpr
#gmx mdrun -deffnm nvt

#gmx grompp -f npt.mdp -c nvt.gro -r nvt.gro -t nvt.cpt -p topol.top -o
npt.tpr
#gmx mdrun -deffnm npt

#gmx grompp -f md.mdp -c npt.gro -t npt.cpt -p topol.top -o md_0_1.tpr
#gmx mdrun -deffnm md_0_1

# To extend by 1 us, comment out stuff above and uncomment below
gmx convert-tpr -s md_0_1.tpr -extend 1000000 -o next.tpr
gmx mdrun -s next.tpr -cpi md_0_1.cpt  -deffnm md_0_1 -append
```

Once you have done this, you can run `sbatch run_md.srun` to run the simulation for as long as desired. If you want to keep extending the job, you can simply re-run the file again and again without further modification.

V. Extract simulation trajectories

The next step is to extract reduced versions of the simulations for analysis and visualization. There are two main tasks:

- Place our molecule(s) of interest in the center of the solvent box. Over the simulation, our protein has likely wandered, at least a small amount, from where it started. We are rarely interested in this diffusion. Worse, if it reached the edge of the box, it may be "sticking through" the side of the solvent box, so it is apparently cut in half. Recentering the box on the molecule(s) of interest ensures that they are intact for downstream analyses.

- Split the simulation into manageable chunks. For simulations of proteins with several hundred amino acids, I've found that 20 ns chunks are a good size. A 20 ns simulation is about a gigabyte of space and can thus moved across networks and loaded for analysis without too much trouble.

  1. Log into the cluster and navigate to your simulation directory in the shared folder.

```
ssh USERNAME@login.talapas.uoregon.edu
cd ~/shared/SIMS
```

  2. Navigate into the directory where you have an MD simulation either completed or currently running.

```
cd run000
```

  3. If you'd like, you can edit the file "extract_md.srun". The main setting to check out is `split_interval=20000`. If you'd like to split your trajectories into shorter or longer blocks, this is where to change it.
  4. Once you are satisfied with the settings in "extract_md.srun" Execute the following command. This will create a directory called "output" that has the files "traj.gro", "traj.tpr", and a collection of "traj*.xtc" files. After the job starts running, this can take anywhere from a few minutes to a few hours to run.

```
sbatch extract_md.srun
```

  4. Once the job completes, check to see how big the folder containing these files with the following command.

```
du -h output/
```

  5. On a *local* terminal, navigate into a convenient directory with enough space to hold the simulation, and type the following. This will copy down the cluster "output" directory into a local "run000" directory.

```
scp -r USERNAME@login.talapas.uoregon.edu:shared/SIMS/run000/output
run000
```