Tutorial by Wunna Kyaw. July 2018. Lee Lab, Single Molecule Science, EMBL Australia.

Edits by Stephanie Xu. August 2018. Lee Lab, Single Molecule Science, EMBL Australia.

**Intro:**

Here, we are going to use GROMACS to simulate the assembly of a pentameric protein complex called the Type 3 secretion system tip complex. Because this is a rather large structure, we are going to use a coarse grained variation (called martini) to simulate a slightly simplified version of the proteins.

This tutorial must be done on Linux with GROMACS 2016 or later installed.

All Command:s are to be typed in the Command: line terminal interface. Ideally, this is done in RAIJIN, accessed through the Windows putty client or ssh for Mac users.

Most of this tutorial (step 2-7) is automated in the file setup\_script.sh. To run it, navigate to the CGMD Template folder and type 'source setup\_script.sh' in Linux terminal.

**General steps for reference:**

1. Get our protein structure and fix crystallography errors.

2. Convert the .pdb structure into a structure readable by coarse grained simulation.

3. Add water molecules to simulate solvation.

4. Optional: Add restraints on the bottom of the protein which represent the effect of a scaffold.

5. Minimize - a short simulation of the protein structure aimed to relax side chain rotamers.

6. Equilibrate - a short simulation aimed to relax the protein in it's solvent.

7. Run MD: Also called "production" simulations: i.e. the business end.

8. Analyse - for coarse grain, this is quite involved because we need to convert coarse grained models back to atomic models.

**############Tutorial starts here.########################**

0a. All required files for simulation are contained in the 'CGMD Template' folder located in the same folder as this document. Create a backup before we touch it. "cp -r CGMDTemplate\_backup"

**1. Check our protein structure for incompatible residues.**

Protein structures are often published with GOL (glycerol) or other crystallization leftovers. GROMACS will not know what to do with these. Open up the protein.pdb in pyMOL, look at sequence view and delete anything that is not a standard amino acid or a biologically important ion such as Fe or Zn. Delete waters as well, then save it.

#############

**2. Convert our .pdb structure into a coarse grained structure using the MARTINI force field.**

**Command:**

module load python

chmod +x dsspcmbi

python martinize.py -f lcrv.pdb -o lcrv.top -x lcrv-cg.pdb -dssp ./dsspcmbi -p backbone -ff martini22

**Explanation:**

**module load python** = Only relevant on RAIJIN supercomputer. Tells RAIJIN to load the python program for us.

**chmod +x dsspcmbi** = This is a security/permissions thing. Required to give permission to the dsspcmbi program to run. If you're admin, it's probably not required.

**python martinize.py** = We are using python to run a pre-written script called martinize.py that will convert the structure into coarse grain for us.

**-f lcrv.pdb** = -f means input. We are inputting lcrv.pdb into the script.

**-o lcrv.top** = This declares the name of the output (-o) coarse grained .top file.

**-x lcrv-cg.pdb** = This declares the name of the output coarse grained .pdb file.

**-dssp ./dsspcmbi** = **-dssp** = Define Secondary Structure of Proteins. Martini requires knowledge about the secondary structure of proteins. '**./dsspcmbi**' is the location of the program that calculates protein secondary structures.

**-p** **backbone** = This tells the script to also output a file that enables backbone restraints on the protein.

**-ff** **martini22** = This explicitly tells the script which forcefield to use. (There are different versions of martini available, here we used martini22)

The outputs lcrv.top and lcrv-cg.pdb now contain coarse grained versions of our protein. If we open lcrv-cg.pdb in pyMOL and type 'as spheres', you will notice that each residue is now represented by 2-4 big spheres instead of the computationally expensive ~10-20 atoms.

**Background:**

A force field is a set of force equations that dictate the properties of a residue. MD groups atoms into distinct residues, and these residues are then described by their force field. There's a number of force field 'sets' available. Here we use MARTINI (which is specifically for coarse grain), but GROMOS, CHARMM or AMBER force fields are commonly used for all-atom simulations. Because every force field recognizes distinct atom groups, it often limits what structures we can simulate with MD. Anything involving the 20 amino acids is fine, but when we start looking at uncommon atom-groups such as small-molecule drugs or unnatural amino acids, there is often no force field available. Writing your own force field 'entry' (also sometimes confusingly called a topology) is very difficult.

**Common troubleshooting:**

- Error message saying anything about an "UNRECOGNIZED RESIDUE", but the residue is a standard amino acid:

There are probably errors in the crystal structure, for example a Lysine that is missing a Nitrogen. These errors occur because of missing electron density when solving the X-ray structures. To fix it: Open the .pdb in pymol and go to the residue that is the source of the error (the error message should tell you the residue ID number). Go to the pymol wizard->mutagenesis and "mutate" the residue to itself, picking the default side chain rotamer. This will replace the damaged residue with it's fixed version. Save the .pdb and repeat.

#############

**3. Create the periodic box (no solvent yet).**

**Command:**

module load GROMACS/2018

gmx\_mpi editconf -f lcrv-cg.pdb -d 1.0 -bt cubic -o lcrv-cg.gro

**Explanation:**

**gmx\_mpi editconf** = We are calling a program called 'editconf' that is contained in GROMACS (gmx\_mpi). mpi stands for message parsing interface, a interface that lets us make use of the parallel computer architecture available on the supercomputers. gmx\_mpi is the MPI version of GROMACS (gmx).

**-f lcrv-cg.pdb** = Inputs the coarse grain file that was created in the previous step.

**-d 1.0** = Declares the distance between the box and the protein.

**-bt cubic** = Box-type cubic. Declares the box as a cube.

**-o lcrv-cg.gro** = The name of the output file that contains our protein, now contained in a box.

The output lcrv-cg.gro contains the coarse grained protein inside a 'box'. Later, we will fill this box with water molecules to simulate the protein in it's solution state.

**Background:**

MD simulates proteins as contained in a box, which defines the "world" that the protein is in. In this case, we have defined the "world" as a cubic box, with dimensions such that the boundaries of the box are at minimum 1.0 nm (10 angstroms) from the surface of the protein. For example, if our protein is shaped like a 5x5x5 cube, the box dimensions will be 6x6x6. Later, we will fill this box with water to simulate the solution state of the protein.

The number (1.0 nm) is important because if the distance is too small, the layer of water molecules between the boundaries of the box and protein will become very thin. This means longer-range electrostatic forces will no longer be reflective of real life.

"Periodic" refers to the fact that the box is a recursive boundary. If the protein strays beyond the boundaries of the box, it will reappear on the opposite end. Like that subway station scene in The Matrix.

#############

**4. Minimize the protein structure.**

**Command:**

gmx\_mpi grompp -p lcrv.top -f minimize.mdp -c lcrv-cg.gro -o vacuum-minimized.tpr

gmx\_mpi mdrun -deffnm vacuum-minimized -v

**Explanation:**

**gmx\_mpi grompp** = GROMACS pre-processor. This is a program that processes the previous files and combines them into a format (.tpr) readable by the MD engine.

**-f minimize.mdp** = Input the MD parameter file. This file contains the parameters necessary to simulate the minimization of the system. Usually you don't need to edit this file because minimization is pretty similar for most systems.

**-p lcrv.top** = input the topology. (bond information)

**-c lcrv-cg.gro** = input the structure (spatial xyz information of atoms)

**-o vacuum-minimized.tpr** = The output .tpr file. This output file is now in a format readable by the MD engine.

**gmx\_mpi mdrun** = mdrun is the workhorse program of GROMACS. Simulates atom movements by integrating forces over a defined timestep and outputting time-resolved atom locations as a trajectory file.

**-deffnm vacuum-minimized** = Define the name of all input and output files. Shorthand for saying "all input files are called vacuum-minimized, and I want all output files to be called the vacuum-minimized as well."

**-v** = verbose. Write a verbose (noisy and lots of information) log.

**Background:**

The rotamers of crystal structures are not always energetically optimal. Minimization is a very short simulation which aims to relax the configuration of side-chain rotamers in proteins. Because we are only concerned with side-chain bond angles, minimization keeps the protein backbone locked into place.

Why is it necessary? If there are side-chain clashes or unfavourable rotamers in the structure, they will cause movements in the protein backbone which may not be reflective of real life protein dynamics.

Because we're only looking at sidechain movement here and there is no solvent, minimization is a very short procedure - a few picoseconds of simulation time. It shouldn't take more than a few minutes.

**GMX GROMPP Troubleshooting:**

When errors come up, they will almost always come up when using gmx grompp because this program is the 'checkpoint' before starting any simulation.

Common errors:

Anything involving an unrecognized or missing atom:

Protein .pdb structure is probably missing an atom in a residue. Use the same pymol mutagenesis trick as described in step 2 troubleshooting section.

#############

**5. Add solvent box to minimized system**

**Command:**

gmx\_mpi solvate -cp vacuum-minimized.gro -cs water.gro -radius 0.21 -o solvated.gro -p lcrv.top

nWaters=$(grep -c W solvated.gro)

echo $'\nW '$nWaters >> $sysname'.top'

**Explanation:**

**gmx\_mpi solvate** = The name of the solvating program that adds water molecules.

**-cp vacuum-minimized.gro** = The input structure that we are adding water molecules to.

**-cs water.gro** = A structure file that describes the water molecules to be added. In all-atom GROMACS, this is not required because it already has a built-in water model. We're doing coarse grain so we need to supply a coarse grained water model.

**-radius 0.21** = van der waals radius of the water.

**-o solvated.gro** = The name of the output structure, now filled with water molecules.

**-p lcrv.top** = Input the topology.

**nWaters=$(grep -c W solvated.gro)** = Get the number of water molecules (W) added to the .gro file. Basically we're counting how many times "W" appears in the .gro file.

**echo $'\n W ' $nWaters >> lcrv.top** = Here, we add a line to lcrv.top to reflect the number of water molecules added. Why? The .top and .gro files are two files describing the same structure, so they must be consistent in terms of atom count. Otherwise GROMACS will be confused as to how many atoms are in the system. Usually, GROMACS can take care of atom count consistency for you. In our case, the water molecules added here are different from standard water, so GROMACS cannot do this automatically like it normally does.

**Background:**

Solvent effects are influential over protein binding or dynamics. MD has 2 main methods of simulating solvent effects: 'implicit solvation' and 'explicit solvation'. Here, we use explicit. In the explicit solvent technique, we add distinct water molecules randomly to the box to simulate a water bath. In contrast, implicit solvation includes a water term in the calculation of forces for the protein. This avoids the need to add thousands of water atoms to the system in favour of an 'implicitly declared' solvent.

Whether one uses implicit or explicit solvation is dependant on the simulation and the computational resources available. Generally, most published MD data uses explicit solvation due to it's accuracy, despite being many times more computationally expensive.

#############

**6. Add ions to the solvated system to neutralize the system's charge to zero.**

**Command:**

gmx\_mpi grompp -p lcrv.top -f minimize.mdp -c solvated.gro -o solvated.tpr

printf '13' | gmx\_mpi genion -s solvated.tpr -o ionized.gro -p lcrv.top -neutral

sed -i '2s/^/#include "martini-ions.itp"\n/' lcrv.top'

**Explanation:**

**gmx\_mpi grompp** = GROMACS pre-processor used again.

**gmx\_mpi genion** = generate ion. The program used to create ions in the system

**-s solvated.tpr** = the input structure to neutralize with ions.

**-p lcrv.top** = the input topology.

**-o ionized.gro** = the output structure, now neutralized with random Na or Cl ions to balance charge.

**-neutral** = A setting which is short hand for saying "please add Na or Cl ions to my system such that it has neutral overall charge".

**printf '13'** = genion only adds ions by replacing existing ions. Ideally, we would replace solvent atoms. This phrase sends the text '13' to genion, telling it to select 'atom group 13' (i.e. solvent waters) as the group to replace. This will only replace a few (<30) waters with ions. Normally, you'd physically type out '13' but this automates it.

**sed -i '2s/^/#include "martini-ions.itp"\n/' lcrv.top** = This gibberish sounding line adds the text "#include martini-ions.itp" to the second line of the lcrv.top file.

**Background:**

A non-zero overall system charge will not be reflective of real life. We need to add ions to the solvent which will balance the charge of the protein. Because these ions added are actually 'normal' (non-coarse grain) ions, we need to tell GROMACS to treat these as coarse grain. We do this by adding a .itp topology which describes ions in terms of the coarse grain MARTINI force field. The file required is called martini-ions.itp, and is already included in the current folder. However we need to change the .top file such that it reads the martini-ions.itp.

Open up lcrv.top and note that #include martini-ions.itp must come after #include martini.itp. Line order matters!

#############

**7. Equilibrate the solvated system. Prepare/Combine files for Equilibrate**

**Command:**

gmx grompp -p lcrv.top -c ionized-minimized.gro -f equilibration.mdp -o equilibrate.tpr

gmx\_mpi mdrun -ntomp $ompthreads -deffnm equilibrate

**Explanation:**

**gmx grompp** = Combine all the files required to setup equilibration.

**Background:**

In the last few steps we have made some major environmental changes (adding waters, neutralizing charge) but the protein structure has not had a chance to adapt in response to the changing energy landscape. These changes should be mirrored by structural changes as well. Equilibration is a short simulation that relaxes sidechain rotamers in the presence of water molecules. It is similar to Minimization, but with many more solvent molecules it requires a longer simulation time.

An automatic script for steps 1-7 is in setup\_script.sh

To do steps 1-7 automatically, open up setup\_script.sh and edit the line:

sysname=lcrv5-cluster3-link.pdb

to

sysname=name

where name is the name of your starting structure (without the .pdb extension.).

#############

8. Optional: Add restraints. Here, we simulate the effect of a DNA scaffold that spatially restrains the protein to a pentameric conformation. We do this by fixing the bottom 2 residues of the linker in place.

**Command:** No Command:, edit the text file manually:

Open up the .itp file corresponding to the protein subunits (Usually called Protein\_A.itp. You may have multiple: Protein\_A.itp, Protein\_B.itp..etc. You need to repeat this step for all of them because each .itp corresponds to a subunit.)

**Command:** nano Protein\_A.itp

This file contains a long list of residues and bonds. Find the residues that you want to restrain. In our case, we are restraining the bottom two residues of the linker (S305 and G306).

Find the atoms that belong to S305 and G306 (the last two residues.) and note down the atom numbers. In our case it is 674-676.

The code to restrain these 3 'atoms' is below. (remember it's coarse grain, a residue is represented by a small number of 'atoms')

Navigate to the very end of the file and type in the following lines (after #endif):

[ position\_restraints ]

674 1 1000 1000 1000

675 1 1000 1000 1000

676 1 1000 1000 1000

Each line corresponds to a position restraint applied to 1 atom.

The first column denotes the atom index that you wish to restrain.

The last 3 columns correspond to energy penalties in the 3 coordinate axes (x,y,z).

The '1' corresponds to what kind of position restraint is applied. We just want a simple point-restraint, which is denoted by '1'.

#############

**9. Prepare/Combine files for MD:**

Now, with restraints code added, we need to do one more grompp to prepare for simulation.

**Command:**

gmx grompp -p $sysname'.top' -c equilibrate.gro -r equilibrate.gro -f MD.mdp -o MD.tpr -maxwarn 1

If this works with no errors, we are ready to run the main MD simulation.

**Explanation:**

The only difference here is the -r argument. Because we applied restraints in the previous step, GROMACS requires an explicit declaration of which structure in which to find the restrained atoms. It is exactly the same file as the -c argument introduced in step 4.

If you skipped step 7, remove '-r equilibrate.gro' from the Command: above.

#############

**10. MD Simulation on RAIJIN Supercomputer**

In this step, we create a script that the supercomputer later reads and executes. This is called submitting a 'job'.

An example of this script is in j-MDjobscript.sh

This step uses a bit of 'PBS' code. PBS is a type of software that handles queuing of jobs on supercomputers. We tell PBS what we want the supercomputer to do, and PBS will pass our message along to the supercomputer. It also makes sure computational resources are spread fairly amongst the users.

We want to use GPUs to simulate the system because GPUs (graphics processing units/graphics cards) are exponentially faster than regular CPUs for numerical simulations. Luckily GROMACS can be built for use on GPUs.

Create an empty job submission script.

**Command:**

nano jobScript.sh

Type in the PBS code which specifies which supercomputer resources we want (e.g. number of cpus, amount of RAM, how long to run)

Text to type into the empty job script:

#!/bin/bash

#PBS -P cx04

#PBS -q gpu

#PBS -l ngpus=2

#PBS -l ncpus=6

#PBS -l mem=16gb

#PBS -l wd

#PBS -l walltime=48:00:00

#PBS -M wunna.a.kyaw@gmail.com

#PBS -m abe

ompthreads=8

export OMP\_NUM\_THREADS=$ompthreads

module load GROMACS/2018-gpu

gmx\_mpi mdrun -ntomp $ompthreads -deffnm MD

Once it's all there, exit and save the file (Ctrl-x)

Now submit the job to the queueing system

**Command:**

qsub jobScript.sh

You can view the status of the job at any time:

**Command:**

qstat -u <your username here>

On Raijin, your username has the format ab1234.

**Explanation:**

**#!/bin/bash** = We are telling the computer that we want the software called 'bash' to read and execute this as a script. This is also an indication that the file is a script and not a regular text file.

**#PBS -P cx04** = -P means project. Enter the project group that you are allocated on the NCI. lawrence lee's group is cx04.

**#PBS -q gpu** = -q means queue. Enter the queue you wish to be in. We use the GPU queue.

**#PBS -l ngpus=2** = We want 2 GPUs.

**#PBS -l ncpus=6** = We want 6 CPUs. (Reason for this is below)

**#PBS -l mem=16gb** = We want 16gb of RAM.

**#PBS -l wd** = We want the current directory as the working directory (wd)

**#PBS -l walltime=48:00:00** = We want this to simulate for 48 hours. (this is currently the max allowed time for gpu jobs)

**#PBS -M wunna.a.kyaw@gmail.com** = Replace this with your email.

#**PBS -m abe** = You will be notified when the job (a)borts, (b)egings, or (e)nds, via the email above. Please enter the right email so you get notices.

**ompthreads=8** = We want 8 OpenMPI threads. This should be equal to the number of CPUs+GPUs.

**export OMP\_NUMTHREADS=$ompthreads** = Apply the ompthreads setting.

**module load GROMACS/2018-gpu** = Load the gpu version of GROMACS. You can't do this on normal Command: line because that system uses regular cpus.

**gmx\_mpi mdrun -ntomp $ompthreads -deffnm MD** = -ntomp $omptherads = Tell GROMACS that we want the specified number of openMPI threads.

**Background:**

Why are we using CPUs?

The architecture of GPU systems on RAIJIN is that each GPU unit comes with 3 CPUs - they have to coexist. Despite cpu's not contributing very much to the simulation, the architecture is such that they have to be present.

#############

**11. Monitoring/troubleshooting your simulation**

**Command:**

qstat -u ab1234, where ab1234 is your nci ID.

Also check your email for updates.

qdel <jobID>

Deletes your job. (Doesn't affect your files, only the queueing of your job).

Your job ID is the 7 digit number that pops up when you type qstat -u ab1234

If there is an error during the simulation, a log file will be created in your working directory called name.e0000000, where 'name' is the name of your job script, and 0000000 is a sequence of numbers corresponding to your job ID. Read this log to find out what is wrong with your simulation.

#############

**12. Analysis.**

Here, the main goal is to visualize our simulation. We will do this in a program called VMD (visual molecular dynamics). First, we have to make some changes to our run output file.

**Command:**

source convert-to-CA.sh

**Explanation:**

convert-to-CA.sh is a script I've written that makes a few quality-of-life changes to our MD.trr (run output) file to make it viewable in VMD.

- Creates a alphacarbon-ONLY version of the trajectory

- Removes the solvent waters

- Aligns the molecule in the center of the periodic box. Without this step, we will see the molecule drifting around the screen due to brownian motion.

**13. Viewing your simulation**

**Command:**

Open up VMD.

Open up '<sysname>-CG-noSolv-backboneCAs.pdb'. Make sure this molecule is selected in the VMD Main window.

Go to file>load data into molecule, and select the alphacarbon-only, aligned trajectory 'MD-CG-noSolv-backboneCAs-ALIGNED.trr' (.dcd works too)

Before you click load:

Make sure 'Load all at once is checked'. This just speeds up the loading process.

Set 'stride' to 20. This setting means that only every 20 frames are loaded into the program, for performance.

Now click load.

If your VMD crashes, set stride to higher values to reduce the memoryload on your computer.

You'll see a bunch of dots representing atoms. For better visuals, go to graphics->representations.

Set draw to 'tube' - for a tube representation. Unfortunately, most of the graphical representations here only work well for all-atom simulations.

Set color to 'chain' - to color by chain.

Go to 'trajectory' tab, and play around with the 'smoothing' - this is a setting that smooths out the jitteriness of your simulation by averaging every x frames, where x is the value that you put in the field.

Click the play button on VMD main to play the simulation.

**GROMACS file types for reference:**

1. **.pdb.** Protein data bank. Standard protein structure files. This is our starting point. Basically a list of atoms and the xyz coordinates for each.

2. **.gro.** A GROMACS-compatible protein structure. Basically the same thing as .pdb but in a different format and doesn't have chain identifiers.

3. **.top.** Topology files. A topology is a type of protein structure file. Instead of having spatial xyz information, it tells GROMACS which atoms are bonded to which. .top and .gro are 2 files that describe the same system, and so each corresponding atom in the .gro must have a corresponding entry in the .top.

4. **.itp.** "Include topology". Same thing as .top but in a modular format. Useful when you have multiple copies of the same structure in 1 simulation. e.g. If you have 5 identical subunits, you can type "#include protein.itp" in a .top file rather than having 5x copies of an entire protein file.

5. **.mdp.** MD Parameter file. These files contain options that govern the parameters of the simulation you are going to run. These files are edited to change things like length of simulation, simulation temperature, pressure, restraints, steering etc. Nearly every simulation will require you to edit the .mdp files.

6. **.tpr.** MD Run input. .gro, .top, .itp, .mdp filetypes are 'combined' into a single file called the run input (.tpr) file. This .tpr file is necessary because it is now in a format that is readable by the main MD engine of GROMACS, 'mdrun'

7. **.trr.** Trajectory. Full precision. The 'result' of a simulation. Usually >10gb in size.

8. **.xtc.** Trajectory, compressed format.

9. **.log.** Log file.

10. **.edr.** Energy file. Resultant energies of the system throughout the simulation.

**Cheat sheet of common commands and their meanings:**

chmod = provide permissions to a program to run

+x = makes the program/command executable

gmx\_mpi = calling card for programs contained in GROMACS

editconf = program; converts genertic structure format to **.gro** or **.pdb**

-f = defines input (**.pdb**)

-d = defines distance between box and protein (in nanometres)

-bt = defines box type

-o = defines output (.**gro**)

genion = program; generates ions in the system

printf = defines ion generation by replacing existing ions

‘13’ = defines selection of ‘atom group 13’ (solvent waters)

-s = defines structure input for ion neutralisation (**.tpr**)

-p = defines topology input (**.top**)

-o = defines neutralised structure output (**.gro**)

-neutral= defines sustem to have neutral overall charge

grompp = program; processes files in order to expand topology from molecular description to atomic description

-p = defines topology input (bond information; **.top**)

-f = defines MD parameter input (**.mdp)**

-c = defines structure input (spatial xyz information of atoms; **.gro**)

-o = defines output file (**.tpr**)

-maxwarn= defines maximum number of warnings allowed before failure

mdrun = program; simulates atom movements by integrating forces over a defined timestep and outputting time-resolved atom locations as a trajectory file

-deffnm = defines name of all input and output files

-ntomp = defines number of threads to start on each node (should be equal to total number of

-v = write a *verbose* log

solvate = program; adds water

-cp = defines structure input that waters are being added to (**.gro**)

-cs = defines structure input that describes water molecules to be added (only required for coarse-grained; **.gro**)

-radius = van der Waals radius of water

-o = defines output file (**.gro**)

-p = defines topology input (**.top**)

module load = specific to RAIJIN; tells RAIJIN to load particular program

python = python program

-dssp = specific for martinize.py; defines protein secondary structure

-f = *find*; defines input file

-ff = specific for martinize.py; tells script which forcefield to use

-p = specific for martinize.py; defines output file that enables backbone restraints on protein

-o = *output*; defines output coarse-grained **.top** file

-x = defines output coarse-grained **.pdb** file

qdel = deletes job

qstat = check status of job

-u = defines username

qsub = submits scripted job to RAIJIN supercomputer

source = runs script locally