AMBER Simulation Guide

version 0.1

Last generated: January 26, 2020

Table of Contents

System Set-Up Using *LEaP*

Troubleshooting AMBER

Introduction

AMBER (Assisted Model Building with Energy Refinement) is a suite of programs that is used to study biomolecular systems. The name references both the force fields for simulation and the simulation software itself. This is some of the information from the Amber Manual , which is hundreds of pages of in-depth explanation. Additional information about Amber (and the error messages that may arise) is available through the Amber Mailing List .

[⬇ PDF Download]

Pre-Definitions

- **Microcanonical Ensemble**: A probability distribution for the state of a system which is isolated from energy changes and has an exact, specified total energy.

- **Canonical Ensemble**: A probability distribution for the state of a system where the system is in thermal equilibrium with a fixed-temperature heat bath.

- **Isothermal-isobaric**: A probability distribution for the state of a system where temperature and pressure are held constant.

- **Isoenthalpic-isobaric**: A probability distribution for the state of a system where enthalpy and pressure are held constant.

File Types

- `pdb` : the Protein Data Bank file. More information is available in the PDB files section (page 59).

- `prepi` : an AMBER PREP internal coordinate file. This is one file which allows non-standard residues to be added to the AMBER database for your simulation. More specific information is available in the AMBER documentation .

- `mol2` : the Tripos Mol2, printed in ASCII format, that was developed for molecular visualization. It contains information about the atom (including coordinates and charges), in addition to bonds and any relevant aspects of the structure.

- `frcmod` : modified force filed parameters. These are critically important for non-standard residues.

- `lib` : an OFF library file containing information about non-standard residues. The generation of this file is preferred for larger co-enzymes, instead of other treatments for non-standard residues. More specific information is available in the AMBER documentation .

- `mdin` : input files that control simulation conditions and settings for data collection.

- `prmtop` : the molecular topology file.

- `inpcrd` : coordinate files. These are preferred by *parm* and *LEaP*, but is almost equal to an `rst` .

- `rst` : restart files that contain information about the last frame's energy minimization or molecular dynamics information from *sander* or *gibbs*. They get their name from being used to restart a simulation from them, instead of starting from the beginning again.

- `mdcrd` : a trajectory file. These contain information on coordinates throughout the course of the simulation.

- `nc` : a NetCDF file. While the manuals/guides I've written tend talk about this as a file generated after *cpptraj* processing, it's really just anything written in NetCDF. Technically, then, everything generated with or after AMBER16 using an mdin (page 24) file without `ntxo=1` (for ASCII rst files) and `ioutfm=0` (for ASCII mdcrd files) should have the `nc` extension for both trajectory and restart information. That said, the default to NetCDF was a recent change, and people using the former `mdcrd` extension may not have changed their naming to the `nc` extension.

System Types

- **NVE**: The microcanonical ensemble, where the system is kept from changes in moles (N), volume (V), and energy (E). This set-up is an example of an adiabatic process.

- **NVT**: The canonical ensemble, where the system is kept from changes in moles (N), volume (V), and temperature (T). This set-up is also known as constant-temperature molecular dynamics, and requires a thermostat.

- **NPT**: The isothermal-isobaric ensemble, where the system is kept from changes in moles (N), pressure (P), and temperature (T). Both a thermostat and barostat are needed.

- **NPH**: The system is kept from changes in moles (N), pressure (P), and enthalpy (H). Enthalpy is held constant when the pressure is fixed without temperature control.

- **NST**: The system is kept under constant-temperature and constant-stress conditions. It is closely related to the NPT ensemble. Hydrostatic pressure is applied uniformly (isotropically), and the components of the stress tensor are controlled. It is good for studying the stress-strain relationship of polymers or metals.

Regulators

- **Langevin Dynamics (NVT or NPT)**: attempts to mimic solvent viscosity by introducing things that occasionally cause friction and perturb the system. When used to control temperature, a small damping constant, γ, should be used.

- **Berendsen Thermostat**: the system is weakly coupled to a heat bath at a set temperature. The thermostat doesn't mirror the canonical ensemble for small systems, but large systems are roughly ok. It uses a leap-frog algorithm to rescale velocities of particles, controlling temperature.

- **Andersen Thermostat**: reassigns a chosen atom or molecule's velocity given by the Maxwell-Boltzmann statistics for the given temperature.

Solvent Models

- **Implicit Solvent**: The solvent is implied and math occurs to make it seem like there's a solvent. Essentially, the system is held under a polarizable medium defined by the dielectric constant. Think of this like a magician waving a wand–there's obviously some magic happening, but you can't actually see it.

- **Explicit Solvent**: The solvent is explicitly set in the system and given physical coordinates. Instead of being an audience member seeing the magic show, you're the magician's apprentice, and you're seeing all the little things that go into tricking the audience (like how there's 2 people in the box being "sawed in half"). TIP3P water is an example of an explicit solvent model.

- **Hybrid Models**: These are somewhere between implicit and explicit, and typically found in QM/MM simulations.

- **Gas Phase**: This isn't actually a solvent model, but the lack of a solvent model. All calculations are done in a vacuum.

Overview

There's a general flow to setting up systems that is outlined in the figure below (page 9). The entire process starts with a crystal structure of the protein, which is explained more in the PDB section (page 59). Once you have a PDB file, you then clean it up to remove anything you don't need. One pretty easy way to do this is using a grep command to extract any lines you care about:

```
grep -e '^ATOM\|^HETATM\|^TER\|^END' file_from_PDB.pdb > file_f
rom_PDB_clean.pdb
```

Using this `grep` command extracts out only lines beginning with:

- `ATOM` : positional information for each atom

- `HETATM` : positional information for hetero atoms (these are typically metals)

- `TER` : termination lines (used to separate out protein, DNA, etc.)

- `END` : the final line of the file and saves them to a new file (`file_from_PDB_clean.pdb`).

This command does not copy any of the lines beginning with:

- `REMARK` : a bunch of comments crystallographers and experimentalists probably care about, but sometimes includes things about missing residues you need to know about

- `ANISOU` : which is specific information on how the crystal structure was obtained

- `CONECT` : connection info between atoms that some programs, like Chimera, add to saved files

- `MASTER` : a dummy "master path" for the file that's useless to us

Figure: Setting Up Systems
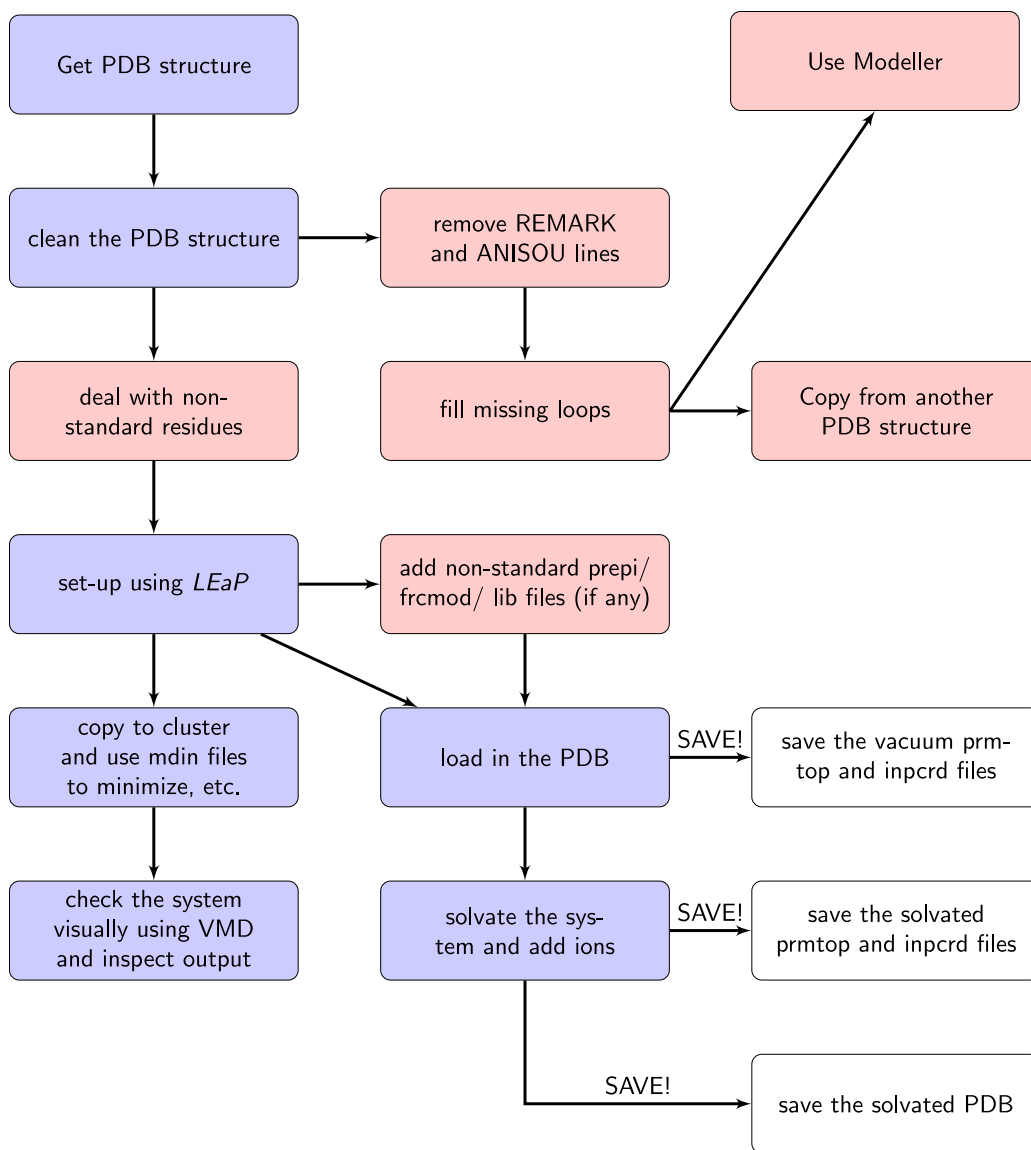
The flow of setting up a system using AMBER. Blocks colored in blue are critical, blocks colored in red may be necessary, and blocks colored in white are steps where you generate files.

After the first-pass for cleaning the PDB structure, it is important to check that the protein is intact. Sometimes, to get a crystal structure, huge chunks of protein are skipped over. That means, that instead of having one, smooth, connected necklace-like chain, your protein is broken into multiple necklaces of varying

sizes–and you're responsible for finding the correct clasps to link them into one (this is better explained in the image (page 10)). When you open a structure in Chimera, areas with missing residues are linked together with dashed lines.

Figure: Missing Residues

$$\text{G T R Y A G K V V}$$

$$\text{G T R \quad\quad K V V}$$
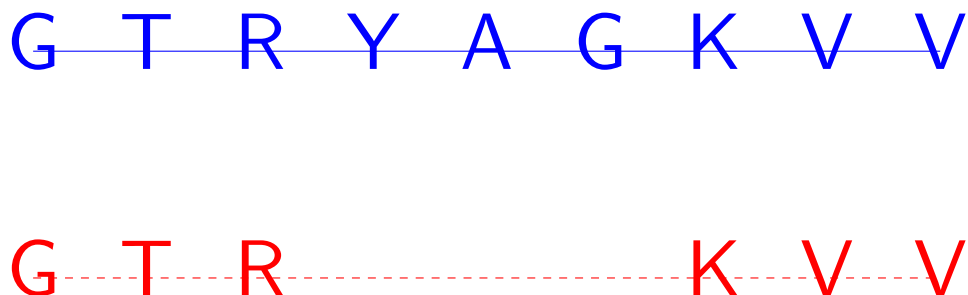
The top block (in blue) is the full, correct protein sequence. The bottom block (in red) is missing three residues in the center.

So how do you deal with these missing residues? First, it would behoove you to see what the experimentalists did to get the crystal structure… which means reading the paper. ? They'll probably talk about how those loops or residues were skipped, and if they had any experimental add-ins (like an X-residue linker) to tell them where the first part ended and the new part began.

Once you know what they did, you can make decisions to match the structure to that. Otherwise, there's 2 common paths moving forward. First, if there's another PDB structure for what you're looking at, you can try to copy the residues from there.

1. Find another PDB

2. Open both structures in Chimera

3. Overlay the structures

4. Resave these overlayed structures with respect to the one you wanted to use

5. Copy and paste the missing residues in a text editor and hope it worked

Unfortunately, the more likely solution is that you'll need to use Modeller to fix those missing sections, or match them to the experimental linker. Luckily, Modeller interfaces with Chimera, so you can build in the missing parts that way. Though, you can go the Python command-line driven route, if you prefer that.

Another important component to getting the PDB structure correct is having any non-standard residues included where they need to be included. Non-standard residues are anything that's not a prototypical amino acid, DNA, or RNA base. So, for example, 5-methylcytosine and thymidine triphosphate would be considered non-standard residues. Similarly, any ligands that are complexed with the protein are also treated as non-standard residues, because you need to generate parameter files for them. The way to deal with these situations is described in the non-standard residue section later on (page 83). Don't forget–you need to copy information from the non-standard residue's PDB file back into the master PDB file that you'll use with *LEaP*. And, you don't need to have hydrogens on that one, because *LEaP* will add them for you.

Finally, after all of that fun getting a master PDB, you're ready to use *LEaP*. Once again, this is described more thoroughly in the set-up using *LEaP* section (page 91). What you're doing in *LEaP* is actually building the files used for simulation.

Woohoo! You made it through making a `prmtop` and `inpcrd`. Now it's **_HIGHLY_** recommended that you check the structures by looking at them in VMD. Ask yourself: (a) *are there any crazy long bonds that are obviously incorrect?* (b) *does my non-standard residue look decent and have the appropriate connections?*. There are two small points to make here. First, what you see in VMD may not be an accurate representation, and the *xleap* editor's interpretation of bonds/connections is the only "true" source. Second, if things look close but not completely correct, they'll likely be fixed in minimization. Make a note of it and check after you've finished minimization.

So what now? Well, you'll copy your `prmtop` and `inpcrd` over to the cluster (or wherever you plan on running this). But that's not all you need–you'll need a runscript (i.e., a `.sh` file–keep reading for an example) and your `mdin` files (see the mdin section (page 24)). Without all of these pieces, your simulation will fail, and you'll spend several minutes asking "WHY????", when really you just didn't copy all the stuff you needed.

After running the minimization, heating, and equilibration on the CPUs, it is **_HIGHLY suggested_** that you check the system again in VMD before beginning production. It is a million times better to spend forever getting the system correct than finish 100 ns of simulation for it all to be for nothing. If everything looks ok, then submit the production steps to run on GPUs. If not, well, you're back to wherever it looks like things went wrong (probably either the non-standard's parametrization or set-up using *LEaP*).

Overlays in Chimera

As mentioned earlier, you can save structures relative to each other in Chimera. First, open both structures in Chimera. Then, follow `Tools → Structure Comparison → MatchMaker`.

This brings up the `MatchMaker` panel (see the image below). On the left, you'll select one structure as the reference, and you'll choose the ones to match on the right.
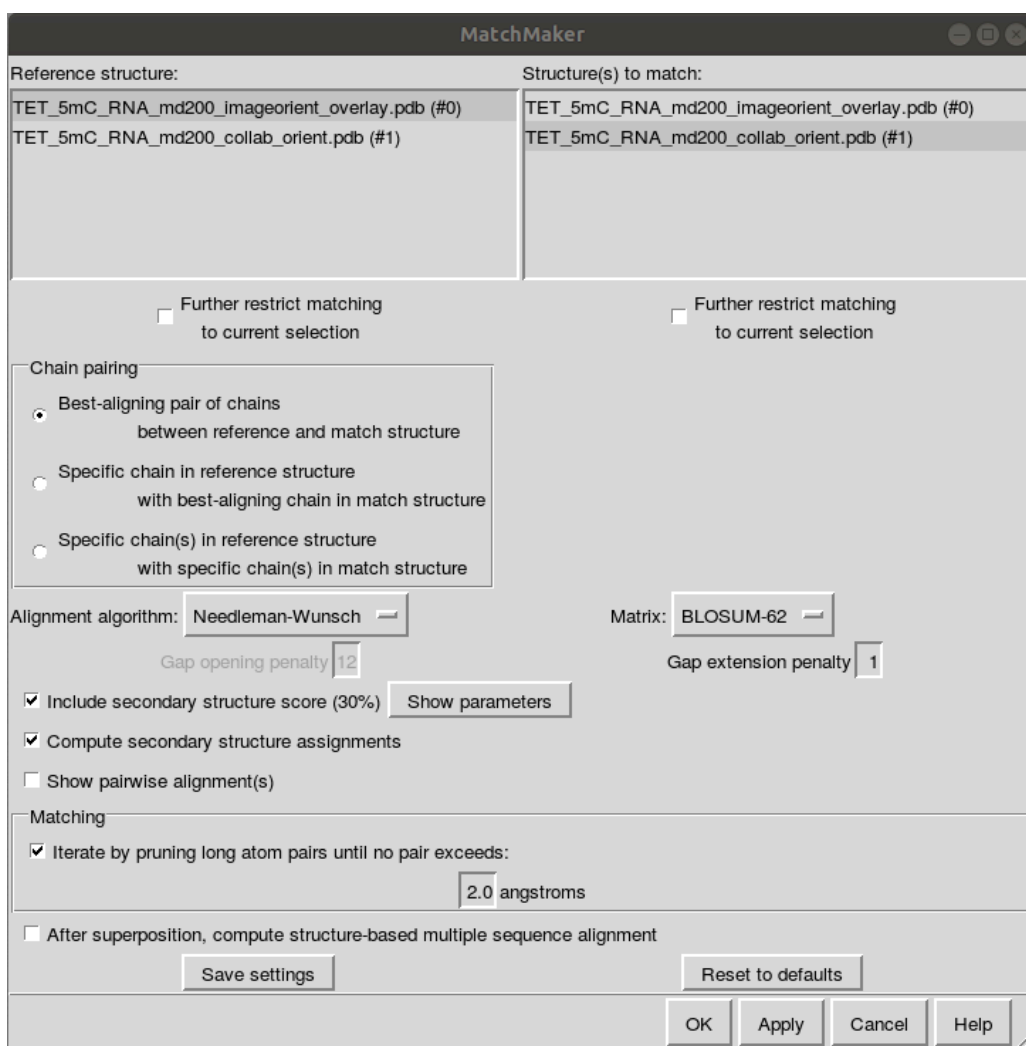
The MatchMaker screen, which allows you to align sequences and then save their coordinates relative to each other.

Once you've hit "apply," they'll be matched and oriented to the reference you selected. You can then save them through `File → SavePDB` . Highlight the one to save, and then select the correct option under "Save relative to model," depending on what you want.

Example CPU Script (Min-Eq)

Run scripts, or jobfiles, contain all the necessary information to run a job. An example is the `mineq.sh` script (thanks, Alice!). Doubly-commented parts (`##`) are included here for explanation purposes.

To use this with as little modification as possible, copy your `whatever-random-name.inpcrd` as `whatever-random-name_init0.rst` , and then modify this script to change the `#PBS -N` line and change the `WT_protein_system_wat` prefix to match `whatever-random-name.prmtop` .

```bash
#!/bin/bash                          ## Tell the script to run i
n a bash shell
#PBS -q my_cpu_alloc                 ## Use CPUs to run the job
#PBS -l nodes=1:ppn=20,mem=20GB      ## Use 1 node, with 20 proce
ssors per node
                                     ## and 20GB memory
#PBS -j oe                           ## Combine standard output & stand
ard error files
#PBS -r n                            ## Says the job is not rerunnable
#PBS -o err.error                    ## Write printed errors to a file
titled err.error
#PBS -N WT_protein                   ## Name of the job to appear in qu
eue

## Access the directory you submitted from
cd $PBS_O_WORKDIR

## Make a nodefile for all of the CPUs to communicate from
cat $PBS_NODEFILE  > $PWD/PBS_NODEFILE
## Load the Amber module
module load amber/19-mvapich2

## Set counters for loop
e=0
f=1

## While file numbers less than 4, run this command
while [ $f -lt 4 ]; do

## Use 20 processors to use the parallel CPU pmemd amber code
## Use the respective mdin corresponds to the value of $f
## So loop 1 uses mdin.1 (etc, etc.)
## -o is your outfile, -p is your topology (prmtop)
## -c is your last save restart file, -r is the restart file wr
itten to
## -x is the velocity file written to, -ref is the reference (l
ast restart file)
mpirun -np 20 -hostfile $PWD/PBS_NODEFILE $AMBERHOME/bin/pmem
d.MPI -O -i mdin.$f \
-o WT_protein_system_wat_init$f.out \
-p WT_protein_system_wat.prmtop \
-c WT_protein_system_wat_init$e.rst \
-r WT_protein_system_wat_init$f.rst \
-x WT_protein_system_wat_init$f.mdcrd \
-ref WT_protein_system_wat_init$e.rst
```

```
## Update counters for loop
e=$[$e+1]
f=$[$f+1]
done
```

Example GPU Script (Production)

There are not many differences between a script to run on GPUs versus CPUs, other than specifying the actual location to run. The following is an example of the `dyanamicsgpu.sh` AMBER script to run on GPUs (thanks again, Alice!). Doubly-commented parts (`##`) are included here for explanation purposes.

```bash
#!/bin/bash                  ## Tell the script to run in a bash shell
#PBS -q my_gpu_alloc         ## Queue allocation
#PBS -l n11-12-13            ## Specify this GPU node to run on
#PBS -j oe                   ## Combine standard output & standard error files
#PBS -r n                    ## Says the job is not rerunnable
#PBS -o err.error            ## Write printed errors to a file titled err.error
#PBS -N WT_protein_GPU       ## Name of the job to appear in queue

## Specify which specific GPU card to run on
export CUDA_VISIBLE_DEVICES=3

## Access the directory you submitted from
cd $PBS_O_WORKDIR

## Load the Amber module
module load amber/16-cuda_serial

## Set counters for loop
e=0
f=1

## While file numbers less than 101, run this command
while [ $f -lt 101 ]; do

## Use pmemd.cuda (GPU) amber code
## Use mdin.4 for the entire loop
## -o is your outfile, -p is your topology (prmtop)
## -c is your last save restart file, -r is the restart file written to
## -x is the velocity file written to, -ref is the reference (last restart file)
$AMBERHOME/bin/pmemd.cuda -O -i mdin.4 \
-o WT_protein_system_wat_md$f.out \
-p WT_protein_system_wat.prmtop \
-c WT_protein_system_wat_md$e.rst \
-r WT_protein_system_wat_md$f.rst \
-x WT_protein_system_wat_md$f.mdcrd \
-ref WT_protein_system_wat_md$e.rst

## Update counters for loop
e=$[$e+1]
```

```
f=$[$f+1]
done
```

> ⚠ **Important:** It is also important to check that you're not accidentally overlapping with someone by connecting to the specific node with `ssh` and using the `nvidia-smi` command.

The following covers how to read the `nvidia-smi` output.

```
$ ssh n11-12-23
Last login: Sat Oct 27 18:52:32 2018 from mycomputer.local
Rocks Compute Node
Rocks 6.2 (SideWinder)
Profile built 20:47 28-Sep-2016

Kickstarted 15:48 28-Sep-2016
C4130(n11) Appliance
$ nvidia-smi
Tue Oct 30 15:34:35 2018
+-----------------------------------------------------------
----------------+
| NVIDIA-SMI 367.48              Driver Version: 367.4
8                        |
|-----------------------------+----------------------
+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volati
le Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Ut
il  Compute M. |
|=============================+======================+=======
===============|
|   0  Tesla K80          On   | 0000:06:00.0     Off
|                   0 |
| N/A   51C    P0   139W / 149W |    494MiB / 11439MiB |     9
4%      Default |
+-----------------------------+----------------------
+----------------------+
|   1  Tesla K80          On   | 0000:07:00.0     Off
|                   0 |
| N/A   73C    P0   133W / 149W |    494MiB / 11439MiB |     8
6%      Default |
+-----------------------------+----------------------
+----------------------+
|   2  Tesla K80          On   | 0000:0A:00.0     Off
|                   0 |
| N/A   22C    P8    26W / 149W |      0MiB / 11439MiB |
0%      Default |
+-----------------------------+----------------------
+----------------------+
|   3  Tesla K80          On   | 0000:0B:00.0     Off
|                   0 |
| N/A   22C    P8    29W / 149W |      0MiB / 11439MiB |
0%      Default |
+-----------------------------+----------------------
```

```
+---------------------+
|   4   Tesla K80           On   | 0000:0E:00.0      Off
|                     0 |
| N/A   21C    P8    28W / 149W |       0MiB / 11439MiB |
0%      Default |
+-----------------------------+----------------------
+----------------------+
|   5   Tesla K80           On   | 0000:0F:00.0      Off
|                     0 |
| N/A   56C    P0   150W / 149W |     494MiB / 11439MiB |     9
9%      Default |
+-----------------------------+----------------------
+----------------------+
|   6   Tesla K80           On   | 0000:12:00.0      Off
|                     0 |
| N/A   21C    P8    28W / 149W |       0MiB / 11439MiB |
0%      Default |
+-----------------------------+----------------------
+----------------------+
|   7   Tesla K80           On   | 0000:13:00.0      Off
|                     0 |
| N/A   56C    P0   151W / 149W |     494MiB / 11439MiB |     9
9%      Default |
+-----------------------------+----------------------
+---------------------+


+--------------------------------------------------------
---------------+
| Processe
s:                                                  GPU Me
mory |
|  GPU        PID  Type  Process nam
e                                 Usage      |
|=======================================================
===============|
|   0     94257   C   /share/apps/AMBER/amber16/bin/pmemd.cud
a      492MiB |
|   1     93626   C   /share/apps/AMBER/amber16/bin/pmemd.cud
a      492MiB |
|   5     93800   C   /share/apps/AMBER/amber16/bin/pmemd.cud
a      492MiB |
|   7     93644   C   /share/apps/AMBER/amber16/bin/pmemd.cud
a      492MiB |
+--------------------------------------------------------
---------------+
```

```
$ exit
logout
Connection to n11-12-13 closed.
```

This command shows us that someone is running something on GPUs 0, 1, 5, and 7. Each of those was specified in the individual GPU script for the particular system running on that specific GPU node (with something like `export CUDA_VISIBLE_DEVICES=7`).

> ⚠ **Important:** Not every system has `nvidia-smi` configured correctly. Make sure you check with someone locally that the numbers in the command and the numbers for exporting `CUDA_VISIBLE_DEVICES` match up.

Molecular Dynamics Workflow

By now you've probably seen some of the working parts to molecular dynamics, but the order that all these things go in can be a little confusing. Enter: the flow chart of doom.
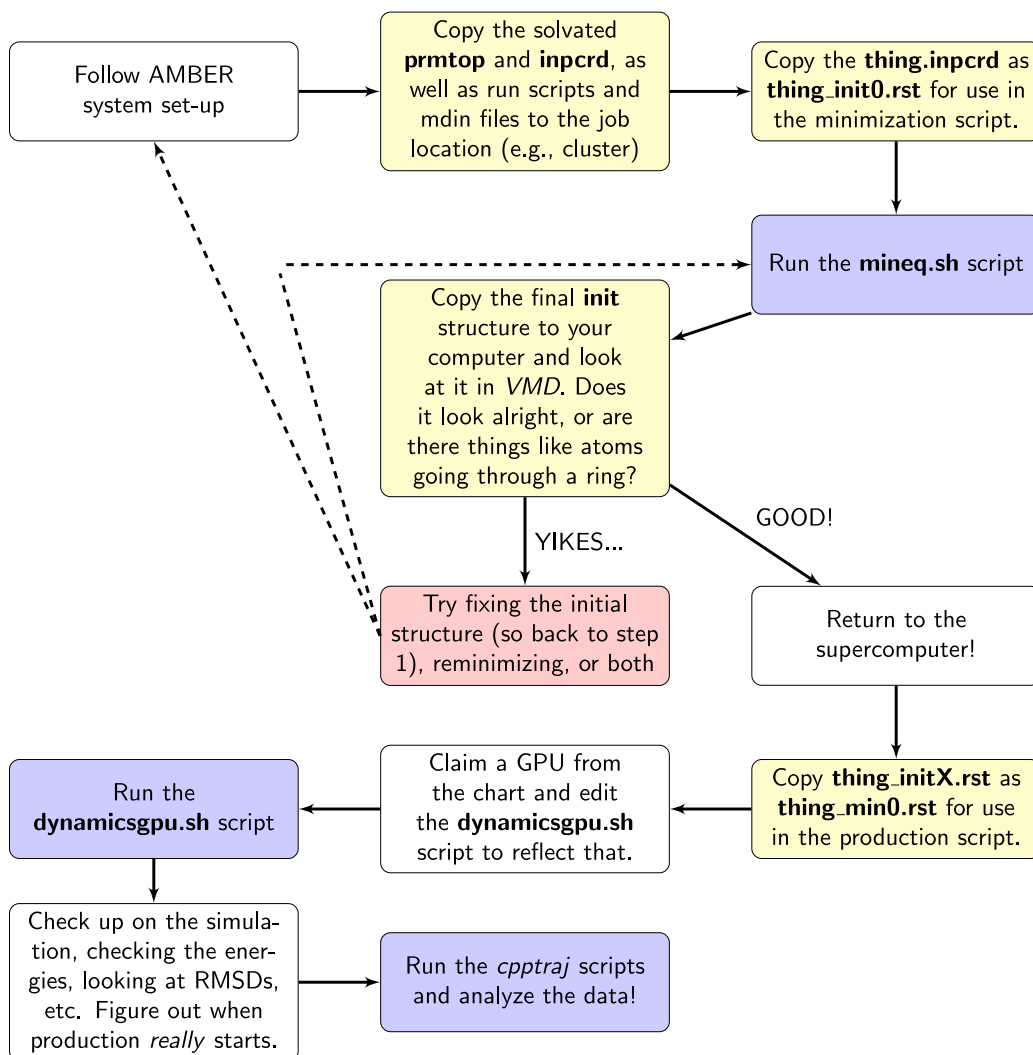
A workflow for running AMBER simulations. Blocks colored yellow are copy steps, blocks colored blue are run steps, blocks colored white are additional tasks, and the red block means something bad happened.

Mdin Files

Input files, known as `mdin` files, set up the conditions that the system is subjected to. Typically, molecular dynamics simulations go through the following steps:

1. **Minimization**. Since PDBs do not have hydrogens, and solvent molecules are just kind of thrown into a box by *LEaP*, the goal of minimization is to, for lack of a better work, minimize the number of clashes and bad contacts between things in the system. Let's say that you have two hydrogen atoms driving wild protein-based cars. If *LEaP* decided that the two hydrogen atom cars should be bumper-to-bumper, minimization would be the driving instructor pointedly reminding them to maintain a 3 second following distance. With each step of minimization, you typically lower any restraints that you have (it's a gradual learning process for our student-driver hydrogens), until your final step which would ideally happen without any restraints at all.

2. **Heating**. MD simulations are based on trajectories, and those trajectories need to have initial velocities in order to have kinetic energy. The heating step is where these initial velocities are assigned, and thermal energy (*cough* heat *cough*) is added to the system. The temperature starts from 0 K and is gradually brought up to the desired temperature (probably 300 K). Usually the restraints come back during heating because most of what is being heated is solvent.

3. **Equilibration**. After the system goes from from being motionless to reaching the desired temperature, the system should undergo equilibration. Equilibration is kind of like the system's time to relax. It goes from being restrained for most of its life to those restraints being lifted and given the opportunity to wiggle freely. Like with minimization, equilibration often has several steps of gradually reducing the restraint weight, until the final equilibration step that has nothing.

4. **Production**. The system is finally ready for unrestrained MD! Huzzah! [And there was much rejoicing.] Production is the part of MD simulations that takes the longest time, but it is also the part that people care about. Analyses are based on the data from the production steps, and production can last as long as you want it to. Typically anything less than 100 ns is useless, and a minimum of 1 μs is seen as ideal. How long you run for is entirely dependent on your resources, but the longer you run for, the better the data.

   - It is wise to break up production into smaller chunks, instead of generating one massive 100 ns data file, though, because writing more

frequent restarts means that if the computer crashes you don't have to start all over, and smaller file sizes means that data corruption is less likely. Additionally, many supercomputers have a wall-clock time, and you lose anything that goes over that time–so writing over that limit means you're wasting both your time and computational resources.

Replicate trajectories should go through each of these 4 steps independently. So, the initial `prmtop` and `inpcrd` should be copied into multiple folders and undergo all these steps alone. The reason for this is that your starting structure won't change, but what contacts are manipulated through minimization might.

Also, it is ** HIGHLY suggested ** that you check the system after you've finished equilibration before moving onto production. Sometimes there are bonds that are clearly not right, and checking will save you a lot of time waiting on a system doomed to fail.

Run Programs

There are a few programs that run AMBER simulations, and their input files/ functionality varies slightly. First, there's *sander* (**S**imulated **A**nnealing with **N**MR-**D**erived **E**nergy **R**estraints). Then there's *pmemd* (**P**article **M**esh **E**wald **M**olecular **D**ynamics), which builds upon *sander*. Finally, exists *pmemd.cuda*, which allows for GPU-acceleration of molecular dynamics (CUDA because it runs on NVIDIA graphics cards. Amazing). Both *sander* and *pmemd* are fairly input-compatible, but check the manual before switching between the two.

I've most often used *pmemd*, because the groups I've been in tend to run production steps using *pmemd.cuda*.

Creating Input Files

Every Amber input file starts with a comment line (that isn't commented out with a symbol like the `#` sign). This comment line typically includes a brief overview of what the input file does (such as which stage of dynamics the system is in). Then, there is a line with `&cntrl`, which signifies that the following lines will include information about basic controls. For a line starting `&pb`, the lines that followed would describe more detailed data manipulation. Any additional comments you want to insert can be started with an exclamation mark (`!`).

The has some information on what specific settings do, and where they can be located in the Amber18 manual .

Table: Highlighted AMBER18 Settings

Setting	Explanation	Amber18 Page
imin	Specifies the minization settings. `= 0` : Run MD without minimization [Default] `= 1` : Energy minimization `= 5` : Read trajectory for analysis	321
maxcyc	Specifies the maximum number of cycles for minimization. [Default = 1].	324
ncyc	Minimization method will switch from steepest descent to conjugate gradient after this many cycles (when default `ntmin = 1`). [Default = 10]	324
ntc	Specifies whether to perform SHAKE (helps ensure bonds meet proper length constraints, and should be used for MD simulations). When using TIP3P water, use `ntf = ntc = 2` . `= 1` : No SHAKE [Default] `= 2` : Bonds with hydrogen are constrained `= 3` : All bonds are constrained (not an option for QM/MM or parallel runs using *sander*).	329

Setting	Explanation	Amber18 Page
iwrap	Specifies wrapping coordinates of restart and trajectory files to within the primary box. `= 0` : No wrapping. The use of *cpptraj* will be necessary to translate back to primary box. [Default] `= 1` : Wrapping is performed. May be necessary for very long runs if files written to ASCII and not NetCDF.	322
ntb	Specifies whether the system is subjected to periodic boundaries. `= 0` : No periodicity and PME is off. [Default when `igb > 0`] `= 1` : Constant volume. [Default when `ntp = igb = 0`, which are both their respective defaults] `= 2` : Constant pressure. [Default when `ntp > 0`]	338
ntr	Specifies whether atoms specified through the restraintmask string should be restrained in Cartesian space with a harmonic potential when `ntr = 1`. [Default = 0]	323
cut	Specifies the nonbonded cutoff (in Å). Any value can be specified. `= 9999.0` : Effectively an infinite cutoff. [Default when `igb > 0`] `= 8.0` : A "good value." [Default when `igb == 0`]	338
ntx	Specifies what will be read from the inpcrd file. This can be used with either ASCII or NetCDF formatted data. `= 1` : Coordinates will be read [Default] `= 5` : Coordinates and velocities will be read. When `ntb > 0`, box information read. Note: need `irest = 1` to use velocity information.	321

Setting	Explanation	Amber18 Page
irest	Allows a simulation to be restarted. `= 0` : Run a new simulation, not a restart. Velocities ignored, and time step set to 0. [Default] `= 1` : Restart simulation, using coordinates and velocities from previously saved restart file. If using this flag, need `ntx > 4`	321
ntpr	Specifies after how many steps the energy information will be printed to both mdout and mdinfo. [Default = 50]	322
ntwx	Specifies how frequently coordinates are written to the mdcrd (trajectory) file. If `ntwx = 0` , no trajectory file is written. [Default = 0]	322
ntwv	Specifies how frequently velocities are written to the mdvel file. `= −1` : velocities written to mdcrd, in a combined coordinate/velocity trajectory file. Must be used with NetCDF binary outpput (`ioutfm = 1`). `= 0` : no velocity trajectory file is written. [Default]	322
nstlim	Specifies the number of MD steps to run. [Default = 1]	324
ioutfm	Specifies the format for the coordinate and velocity trajectory files. `= 0` : Formatted ASCII `= 1` : Binary NetCDF [Default]	323
t	Specifies the time at the start (for personal reference). Automatically taken from an input file if `IREST = 1` . `= 0.0` : [Default]	324
dt	Specifies the time step in picoseconds. `= 0.001` : Recommended maximum for runs without SHAKE (equivalent to 1 fs). [Default] `= 0.002` : Recommended maximum for runs with SHAKE (equivalent to 2 fs).	325

Setting	Explanation	Amber18 Page
ntf	Specifies how forces are evaluated. `= 1` : calculates complete interactions [Default] `= 2` : calculation ignores bond interactions with H-atoms (use with `ntc = 2`) `= 3` : calculation ignores all bond interactions (use with `ntc = 3`) `= 4` : calculation ignores all bonds and angles with H-atoms `= 5` : calculation ignores all bonds and angle interactions `= 6` : calculation ignores dihedrals involving H-atoms and all bonds and angle interactions `= 7` : calculation ignores all bond, angle, and dihedral interactions `= 8` : calculation ignores all bond, angle, dihedral, and non-bonded interactions	338
ntp	Specifies constant pressure dynamics. `= 0` : no pressure scaling performed. [Default] `= 1` : MD with isotropic position scaling. `= 2` : MD with anisotropic pressure scaling (should only be used with orthogonal boxes, aka "solvateoct"). `= 3` : MD with semiisotropic pressure scaling (should only be used with "solvateoct" and `csurften > 0`).	327
ig	The random seed. MD starting velocities are dependent on the random number. Setting a specific random seed can be done to check reproducibility. `= -1` : Seed is set based off current date and time. [Default]	326
barostat	Specifies the barostat for pressure control. `= 1` : Berendsen [Default] `= 2` : Monte Carlo	328

Setting	Explanation	Amber18 Page
ntt	Specifies temperature scaling. `= 0` : Constant total energy, assuming `ntb < 2`. Microcanonical NVE. `= 1` : Constant temperature, with weak-coupling algorithm. Dangerous for generalized Born simulations. `= 2` : Anderson-like temperature, with Newtonian dynamics. Canonical (constant T) ensemble. `= 3` : Langevin dynamics with collision frequency specified by `gamma_ln`. (If `gamma_ln = 0`, same as `ntt = 0`). Restarts of these systems should have an explicitly set `ig` value. `= 9` : Optimized Isokinetic Nose-Hoover (OIN) chain ensemble, aka a fancy constant temperature simulation. `= 10` : Stochastic Isokinetic Nose-Hoover RESPA Integrator. Can help demonstrate a Boltzman distribution.	325
temp0	The temperature that systems with `ntt > 0` should be held constant at. `= 300` : [Default] If greater than 300, reduce step size, because of potential for errors with SHAKE and other things.	326
gamma_ln	Specifies the collision frequency, γ, in ps^{-1} when using `ntt = 3`. Also specifies constants when `ntt = 9` and `ntt = 10` `= 0` : [Default]	326
tempi	Specifies the initial temperature, assigning velocites from a Maxwellian distribution. This has no impact if `ntx > 3` `= 0.0` : velocities calculated from the forces, instead of assigned. [Default]	322
taup	Specifies the pressure relaxation time when `nto > 0`. Recommended to be between 1-5 ps. Unstable trajectories may need a higher value. `= 1.0` : [Default]	328

Setting	Explanation	Amber18 Page
tautp	Specifies the time constant (in ps) when `ntt = 1`. Values should range from 0.5-5.0 ps; small values result in faster heating and less natural movement. `= 1.0` : [Default]	326
vlimit	Velocities greater than specified `vlimit` will be reduced to it, keeping the sign, which helps avoid run instability. [Default = 20]	327
ibelly	Specifies belly type dynamics. `= 1` : Some atoms in the system are allowed to move, and the rest will be frozen. The moving atoms are specified through `bellymask`. Note: `ibelly ≠ 0` is not supported by GPUs.	323
nsnb	Specifies the frequency of nonbonded list updates when `igb = nbflag = 0`. [Default 25]	339
nmropt	This specifies whether NMR data will be performed. `= 0` : No NMR analysis will be done [Default] `= 1` : Gives NMR restraints and weight changes `= 5` : Gives NMR restraints, weight changes, NOESY volumes, chemical shifts, and residual dipolar restraints Note: `nmropt > 1` is unsupported for runs on GPUs.	321
pres0	The reference pressure (in bar, where 1 bar = 0.987 atm). [Default = 1]	328
restraint_wt	The weight of positional restraints (in kcal mol^{-1}-Å^2).	324
restraintmask	The string that explains what atoms should be restrained when `ntr= 1`.	323

Spacing and Other Formatting

Amber input files also have their own internal spacing requirements. Comment lines start at the first line position. Both types of lines that start with the `&` have a space after the start of the line and before the ampersand. The lines after that, which actually describe the system settings, start with two spaces after the beginning of the command line. The final line, which ends that slew of command lines, consists of three spaces between the start of the command line and the `/` symbol (which means end). Thus, it looks something like this (where the underscores denote spaces):

```
Comment
_&cntrl
__imin  = 1
___/
```

When restraints are involved, they follow the `/` line, and start yet again with a comment line (something like "restraints" or "hold protein fixed"). The weight of the restraint is on a line to itself, followed by a line specifying the residue numbers. After each specific restraint has been included, the file ends with two lines that simply say `END`. The following example sets the restraint weight at 10 kcal mol^{-1}-Å^2 for residues 1 through 25.

```
System Restraints
10.0
RES 1 25
END
END
```

You can insert comments by beginning them with an exclamation mark (`!`).

```
General comment describing settings of this file
  &cntrl
  imin=1, ! Do minimization
  cut=8.0, ! Use 8 angstrom cutoff
   /
```

AMBER Math (aka Simulation Length)

Since `nstlim` refers to the total number of steps, the simulation time of a single file can be found through an equation of [`nstlim` / 1000000 fs ns⁻¹ x `dt` = time of simulation (in ns)]. Thus, with an `nstlim` of 250000 at a 2 fs time-step, 200 files of unrestrained MD data would yield a total simulation time of 100 nanoseconds. One shows times with a 2 fs time-step, and another shows times with a 1 fs time-step. The 2 fs time-step is recommended when using SHAKE (set by `ntc > 1`). Without SHAKE, the 1 fs time-step is recommended.

Table: Simulation times with 2.0 fs timestep

nstlim	Single File Length	Number of Files for 100 ns	Number of Files for 1 µs
125000	0.25 ns	400	4000
250000	0.5 ns	200	2000
500000	1 ns	100	1000
1000000	2 ns	50	500
2500000	5 ns	20	200
5000000	10 ns	10	100
10000000	20 ns	5 & 50	
12500000	25 ns	4 & 40	

Table: Simulation times with 1.0 fs timestep

nstlim	Single File Length	Number of Files for 100 ns	Number of Files for 1 µs
250000	0.25 ns	400	4000

nstlim	Single File Length	Number of Files for 100 ns	Number of Files for 1 µs
500000	0.5 ns	200	2000
1000000	1 ns	100	1000
2000000	2 ns	50	500
5000000	5 ns	20	200
10000000	10 ns	10	100
20000000	20 ns	5	50
25000000	25 ns	4	40

GPU Support

Simulations in the NVE, NVT, and NPT ensembles, as well as implicit solvent Generalized Born simulations, can benefit from GPU acceleration using *pmemd.cuda*. GPU runs do not give an Ewald error estimate, so it is recommended to run tests on CPUs to ensure reasonable error estimates. There are some specific options that are unsupported with *pmemd.cuda*, and those are listed on page 369 of the Amber17 manual. For completeness, the specific settings that aren't supported are in the GPU table (page 35).

Table: Unsupported GPU settings in AMBER

ibelly ≠ 0	icfe ≠ 0	igb ≠ 0 && cut < systemsize
nmropt > 1	nrespa ≠ 1	vlimit ≠ -1
MPI runs with imin = 1	es_cutoff ≠ vdw_cutoff	order > 4
emil_do_calc ≠ 0	lj1264 ≠ 0	iemap > 0
	isgld > 0	

Example Files

NVT with Berendsen Thermostat

Yet again, thanks to Alice for these inputs (who in turn thanks Sajeewa). Using the Berendsen thermostat makes this halfway NVE and halfway NVT, but it should be reported as NVT. That said, you really shouldn't use the Berendsen thermostat because of the possibility of experiencing the "Flying Ice Cube Effect." You can read more about that in the original 1998 paper and a 2018 paper revisiting the concept . If you need any more convincing, see the tweet below.

A compelling argument about not using Berendsen. Don't do it (for the meme).

Minimization lasts from inputs 1-5. Heating occurs at input 6. Equilibration lasts from inputs 7-10. Production is performed with input 11.

`mdin.1`

Energy minimization for 5000 cycles, switching from steepest descent to conjugate gradient after 500 cycles. Bonds with hydrogen are constrained and ignored with SHAKE. Wrapping is performed for files, the system is held at constant volume, atoms with the specified restraint mask are restrained at those weights (in kcal mol^{-1}), and the nonbonded cutoff is 9 Å.

```
Protein
 &cntrl
   imin  = 1,
   maxcyc = 5000,
   ncyc  = 500,
   ntc = 2, ntf = 2,
   iwrap = 1,
   ntb   = 1,
   ntr   = 1,
   cut   = 9
   /
Hold protein fixed
200.0
RES 1 430
200.0
RES 431 450
200.0
RES 451 455
END
END
```

`mdin.2`

Coordinates will be read and a new simulation will be run. The velocities are
ignored and the time step is set to zero. After 5 steps, energy information will be
printed to the mdout and mdinfo files. After 100 steps, coordinates will be written
to the trajectory file, and no velocity trajectory file is written. 10,000 MD steps will
be run from a start time of 0, and the time step is every 1 fs. Bonds with hydrogen
are constrained and ignored with SHAKE. The simulation is at constant pressure
with isotropic position scaling. Wrapping is performed for files, and the random
seed is set based off the current date and time. The simulation is run at constant
temperature, with weak coupling. The initial temperature is 10 K, and the
externally coupled temperature is held constant at 10 K at a speed of 0.5 ps.
Atoms with the specified restraint mask are restrained at those weights (in kcal
mol^{-1}), and the nonbonded cutoff is 9 Å.

```
Protein
 &cntrl
  ntx=1,          irest=0,
  ntpr=5,         ntwx=100,      ntwv=00,
  nstlim=10000,   t=0.00,         dt=0.00100,
  ntc = 2, ntf = 2,
  ntb    = 2, ntp=1,
  iwrap = 1,
  ig=-1
  ntt=1, temp0=010.0, tempi=010.0, tautp=0.5,
  ntr    = 1,
  cut    = 09.0,
 /
Hold protein fixed
200.0
RES 1 430
200.0
RES 431 450
200.0
RES 451 455
END
END
```

mdin.3

Coordinates and box information are read from a NetCDF or ASCII coordinate file. The simulation is restarted, based on coordinates and velocities from the previously saved restart file. The restraint weight is halved. Everything else is the same as [See the note below.]

```
Protein
 &cntrl
  ntx=7,          irest=1,
  ntpr=5,         ntwx=100,      ntwv=00,
  nstlim=100000,  t=0.00,        dt=0.00100,
  ntc = 2, ntf = 2,
  ntb   = 2, ntp=1,
  iwrap = 1,
  ig=-1
  ntt=1, temp0=010.0, tempi=010.0, tautp=0.5,
  ntr   = 1,
  cut   = 09.0,
 /
Hold protein fixed
100.0
RES 1 430
100.0
RES 431 450
100.0
RES 451 455
END
END
```

> **ⓘ Note:** `ntx = 7` is an old (OLD) form , and is equivalent to `ntx = 5`. Thus, `ntx = 5` should be used. This is corrected in the remaining input files shown here.

mdin.4

The frequency of the nonbonded list updates every step, and there are no belly type dynamics. The restraint weight is halved. Everything else is the same as mdin.3 (page 38).

```
Protein
 &cntrl
  ntx=5,          irest=1,
  nsnb=1,
  ntpr=5,         ntwx=100,      ntwv=00,
  nstlim=100000,   t=0.00,         dt=0.00100,
  ntc = 2, ntf = 2,
  ntb    = 2, ntp=1,
  iwrap = 1,
  ntt=1, temp0=010.0, tempi=010.0, tautp=0.5,
  ntr    = 1,
  cut    = 09.0,
  ibelly=0,
 /
Hold protein fixed
50.0
RES 1 430
50.0
RES 431 450
50.0
RES 451 455
END
END
```

`mdin.5`

The restraint weight is halved from mdin.4 (page 39).

```
Protein
 &cntrl
  ntx=5,          irest=1,
  nsnb=1,
  ntpr=5,         ntwx=100,      ntwv=00,
  nstlim=10000,   t=0.00,          dt=0.00100,
  ntc = 2, ntf = 2,
  ntb    = 2, ntp=1,
  iwrap = 1,
  ntt=1, temp0=010.0, tempi=010.0, tautp=0.5,
  ntr    = 1,
  cut    = 09.0,
  ibelly=0,
 /
Hold protein fixed
25.0
RES 1 430
25.0
RES 431 450
25.0
RES 451 455
END
END
```

`mdin.6`

Coordinates will be read and a new simulation will be run. The velocities are ignored and the time step is set to zero. The frequency of the nonbonded list updates every step. After 100 steps, energy information will be printed to the mdout and mdinfo files. After 1000 steps, coordinates will be written to the trajectory file, and no velocity trajectory file is written. 100,000 MD steps will be run from a start time of 0, and the time step is every 1 fs. Bonds with hydrogen are constrained and ignored with SHAKE. The simulation is at constant volume. Wrapping is performed for files. The simulation is run at constant temperature, with weak coupling. The initial temperature is 10.0 K, and the externally coupled temperature is held constant at 10.0 K at a speed of 0.001 ps. Velocities greater than 20.0 will be reduced to the magnitude 20.0. Atoms with the specified restraint mask are restrained at those weights (in kcal mol^{-1}), NMR restraints and weight changes will be given, and the nonbonded cutoff is 9 Å. The target temperature (`TEMP0`) will be adjusted at the specified time intervals.

```
Protein Constant volume constraints on protein + active site
 &cntrl
  ntx=5,          irest=1,
  nsnb=1,
  ntpr=100,         ntwx=1000,      ntwv=00,
  nstlim=100000,   t=0.00,          dt=0.00100,
  ntc = 2, ntf = 2,
  ntb    = 1,
  iwrap = 1,
  ntt=1, temp0=010.0, tempi=010.0, tautp=0.001, vlimit=20.0,
  ntr    = 1,
  cut    = 09.0,
  nmropt=1
 /
 &wt type='TEMP0', istep1=00000, istep2=05000, value1=000., val
ue2=010.,  &end
 &wt type='TEMP0', istep1=05001, istep2=10000, value1=010., val
ue2=020.,  &end
 &wt type='TEMP0', istep1=10001, istep2=20000, value1=020., val
ue2=050.,  &end
 &wt type='TEMP0', istep1=20001, istep2=30000, value1=050., val
ue2=100.,  &end
 &wt type='TEMP0', istep1=30001, istep2=40000, value1=100., val
ue2=150.,  &end
 &wt type='TEMP0', istep1=40001, istep2=50000, value1=150., val
ue2=200.,  &end
 &wt type='TEMP0', istep1=50001, istep2=60000, value1=200., val
ue2=250.,  &end
 &wt type='TEMP0', istep1=60001, istep2=70000, value1=250., val
ue2=300.,  &end
 &wt type='TEMP0', istep1=70001, istep2=80000, value1=300., val
ue2=325.,  &end
 &wt type='TEMP0', istep1=80001,istep2=100000, value1=325., val
ue2=300.,  &end
 &wt type='END'  &end
Hold protein fixed
500.0
RES 1 430
500.0
RES 431 450
500.0
RES 451 455
END
END
```

mdin.7

After 10,000 steps, energy information will be printed to the mdout and mdinfo files. After 1,000 steps, coordinates will be written to the trajectory file, and no velocity trajectory file is written. 20000 MD steps will be run from a start time of 0, and the time step is every 1 fs. Bonds with hydrogen are constrained and ignored with SHAKE. The initial temperature is 300.0 K, and the externally coupled temperature is held constant at 300.0 K at a speed of 1.0 ps. No NMR analysis will be done, and the initial temperature will not be varied. Otherwise, this is the same as mdin.6 (page 41).

```
Protein Constant volume constraints on protein + active site
 &cntrl
  ntx=5, irest=1,
  nsnb=1,
  ntpr=10000, ntwx=1000, ntwv=0,
  nstlim=20000, t=0.00, dt=0.00100,
  ntc = 2, ntf = 2,
  iwrap = 1,
  ntb    = 1,
  ntt=1, temp0=300.0, tempi=300.0, tautp=1.0, vlimit=20.0,
  ntr    = 1,
  cut    = 9.0,
 /
Hold protein fixed
500.0
RES 1 430
500.0
RES 431 450
500.0
RES 451 455
END
END
```

mdin.8

After 100 steps, energy information will be printed to the mdout and mdinfo files. After 1,000 steps, coordinates will be written to the trajectory file, and no velocity trajectory file is written. 10,000 MD steps will be run from a start time of 0, and the time step is every 1 fs. The restraint weight is reduced. The rest is the same as mdin.7 (page 43).

```
Protein Constant volume constraints on protein + active site
 &cntrl
  ntx=5,         irest=1,
  nsnb=1,
  ntpr=100,       ntwx=1000,      ntwv=00,
  nstlim=10000,   t=0.00,         dt=0.00100,
  ntc = 2, ntf = 2,
  ntb    = 1,
  iwrap = 1,
  ntt=1, temp0=300.0, tempi=300.0, tautp=1.0, vlimit=20.0,
  ntr    = 1,
  cut    = 09.0
 /
Hold protein fixed
200.0
RES 1 430
200.0
RES 431 450
200.0
RES 451 455
END
END
```

mdin.9

The restraint weight is reduced from mdin.8 (page 43).

```
Protein Constant volume constraints on protein + active site
 &cntrl
  ntx=5,          irest=1,
  nsnb=1,
  ntpr=100,       ntwx=1000,      ntwv=00,
  nstlim=10000,   t=0.00,         dt=0.00100,
  ntc = 2, ntf = 2,
  ntb    = 1,
  iwrap = 1,
  ntt=1, temp0=300.0, tempi=300.0, tautp=1.0, vlimit=20.0,
  ntr    = 1,
  cut    = 09.0
 /
Hold protein fixed
0.0
RES 1 430
25.0
RES 431 450
0.0
RES 451 455
END
END
```

mdin.10

The restraint weight is reduced from .

```
Protein Constant volume constraints on protein + active site
 &cntrl
  ntx=5,          irest=1,
  nsnb=1,
  ntpr=100,        ntwx=1000,      ntwv=00,
  nstlim=10000,    t=0.00,         dt=0.00100,
  ntc = 2, ntf = 2,
  ntb    = 1,
  iwrap = 1,
  ntt=1, temp0=300.0, tempi=300.0, tautp=1.0, vlimit=20.0,
  ntr    = 1,
  cut    = 09.0
 /
Hold protein fixed
0.0
RES 1 430
10.0
RES 431 450
0.0
RES 451 455
END
END
```

mdin.11

Coordinates will be read and a new simulation will be run. The velocities are
ignored and the time step is set to zero. The frequency of the nonbonded list
updates every step. After 100 steps, energy information will be printed to the
mdout and mdinfo files. After 1,000 steps, coordinates will be written to the
trajectory file, and no velocity trajectory file is written. 250,000 MD steps will be
run from a start time of 0, and the time step is every 2 fs. Bonds with hydrogen are
constrained and ignored with SHAKE. The simulation is at constant volume. No
wapping is performed, requiring *cpptraj* to translate back to the original box. The
simulation is run at constant temperature, with weak coupling. The initial
temperature is 300.0 K, and the externally coupled temperature is held constant at
300.0 K at a speed of 1 ps. Velocities greater than 20.0 will be reduced to the
magnitude 20.0; this is later changed to -1.0. NMR restraints and weight changes
will be given, and the nonbonded cutoff is 9 Å. The relative weights of all the NMR
restraint energy terms (REST) will be adjusted at step 5,000.

```
Protein Constant volume no contraints
 &cntrl
  ntx=5,          irest=1,
  nsnb=1,
  ntpr=100,       ntwx=1000,      ntwv=00,
  nstlim=250000,  t=0.00,          dt=0.00200,
  ntc = 2, ntf = 2,
  ntb    = 1,
  ntt=1, temp0=300.0, tempi=300.0, tautp=1.0, vlimit=20.0,
  iwrap = 0
  cut    = 09.0,
  nmropt=1
  vlimit=-1
 /
 &wt type='REST', istep1=000000,istep2=5000, &end
 &wt type='END'  &end /
```

NVT Through Heating and NPT Production Langevin Dynamics

Thanks to Hedi for these inputs.

`mdin.1` (aka `min1.in`)

Energy minimization for 1,000 cycles, switching from steepest descent to conjugate gradient after 50 cycles. No SHAKE, and complete interactions are calculated. Wrapping is performed for files, the system is held at constant volume, atoms with the specified restraint mask are restrained at those weights (in kcal mol^{-1}), and the nonbonded cutoff is 9 Å.

```
minimization of water and ions
 &cntrl
  imin   = 1,
  maxcyc = 1000,
  ncyc   = 50,
  ntc = 1, ntf = 1,
  iwrap = 1,
  ntb    = 1,
  ntr    = 1,
  cut    = 9.0
 /
Hold the Enzyme Fixed
500.0
RES 1 455
END
END
```

mdin.2 (aka min2.in)

Energy minimization for 2,500 cycles, switching from steepest descent to conjugate gradient after 50 cycles. Bonds with hydrogen are constrained and ignored with SHAKE. Wrapping is performed for files, the system is held at constant volume, no restraints are held on atoms, and the nonbonded cutoff is 9 Å.

```
minimization of enzyme
 &cntrl
  imin   = 1,
  maxcyc = 2500,
  ncyc   = 50,
  ntc = 2, ntf = 2,
  iwrap = 1,
  ntb    = 1,
  ntr    = 0,
  cut    = 9.0
 /
```

mdin.3 (aka heat.in)

Coordinates will be read, and a new simulation will be run at constant volume. Atoms with the specified restraint mask are restrained at those weights (in kcal mol^{-1}), and the nonbonded cutoff is 9 Å. Bonds with hydrogen are constrained

and ignored with SHAKE. The initial temperature is 0.0 K, and the externally coupled temperature is held constant at 300.0 K using Langevin dynamics with collisional frequency of 1 ps. The random seed is set based on date and time. 100000 MD steps will be run with a time step of 1 fs. After 100 steps, energy information will be printed to the mdout and mdinfo files. After 10,000 steps, coordinates will be written to the trajectory file, and the restart file will be written to every 100 steps. Atoms with the specified restraint mask are restrained at those weights (in kcal mol^{-1}).

```
heat up
 &cntrl
  imin   = 0,
  irest  = 0,
  ntx    = 1,
  ntb    = 1,
  cut    = 9.0,
  ntr    = 1,
  ntc    = 2,
  ntf    = 2,
  tempi  = 0.0,
  temp0  = 300.0,
  ntt    = 3,
  gamma_ln = 1.0, ig = -1,
  nstlim = 100000, dt = 0.001
  ntpr = 100, ntwx = 10000, ntwr = 100
 /
Keep Enzyme fixed with weak restraints
10.0
RES 1 455
END
END
```

mdin.4 (aka prod.in)

The simulation will be restarted from the previously saved restart file. Coordinates and velocities will be read. The system will be held at constant pressure using isotropic position scaling, with a reference pressure of 1.0 bar, and a pressure relaxation time of 2.0 ps. The nonbonded cutoff is 9 Å, and no atoms will be restrained. Bonds with hydrogen are constrained and ignored with SHAKE. The initial temperature is 300.0 K, and the externally coupled temperature is held constant at 300.0 K using Langevin dynamics with collisional frequency of 1 ps^{-1}. The random seed is set based on date and time. 1,000,000 MD steps will be run

with a time step of 1 fs. After 100 steps, energy information will be printed to the mdout and mdinfo files. After 10,000 steps, coordinates will be written to the trajectory file, and the restart file will be written to every 100 steps.

```
production
 &cntrl
  imin = 0, irest = 1, ntx = 7,
  ntb = 2, pres0 = 1.0, ntp = 1,
  taup = 2.0,
  cut = 9.0, ntr = 0,
  ntc = 2, ntf = 2,
  tempi = 300.0, temp0 = 300.0,
  ntt = 3, gamma_ln = 1.0, ig = −1,
  nstlim = 1000000, dt = 0.001,
  ntpr = 100, ntwx = 10000, ntwr = 100
 /
```

NPT Langevin Dynamics

These inputs were provided by Dr. Bill Miller III.

mdin.1 (aka min1.mdin)

Energy minimization for 5,000 cycles, switching from steepest descent to conjugate gradient after 1,000 cycles. Information is printed to mdout and mdinfo every 50 steps. The nonbonded cutoff is 8 Å, wrapping is performed for files, and atoms with the specified restraint mask (i.e. all non-hydrogen atoms) are restrained at 10.0 kcal mol^{-1}.

```
Minimization to relax initial bad contacts, explicit solvent
 &cntrl
  imin=1,
  ncyc=1000,
  maxcyc=5000,
  ntpr=50,
  cut=8,
  iwrap=1,
  ntr=1,
  restraint_wt=10.0,
  restraintmask='!@H=',
 /
```

mdin.2 (aka min2.mdin)

The is the same as mdin.1 (page 50), except the restraints are reduced to 5.0 kcal mol^{-1}.

```
Minimization to relax initial bad contacts, explicit solvent
 &cntrl
   imin=1,
   ncyc=1000,
   maxcyc=5000,
   ntpr=50,
   cut=8,
   iwrap=1,
   ntr=1,
   restraint_wt=5.0,
   restraintmask='!@H=',
 /
```

mdin.3 (aka min3.mdin)

The is the same as mdin.2 (page 51), except the restraints are reduced to 2.0 kcal mol^{-1}.

```
Minimization to relax initial bad contacts, explicit solvent
 &cntrl
   imin=1,
   ncyc=1000,
   maxcyc=5000,
   ntpr=50,
   cut=8,
   iwrap=1,
   ntr=1,
   restraint_wt=2.0,
   restraintmask='!@H=',
 /
```

mdin.4 (aka min4.mdin)

This is the same as mdin.3 (page 51), except the restraints are reduced to 1.0 kcal mol^{-1}.

```
Minimization to relax initial bad contacts, explicit solvent
 &cntrl
  imin=1,
  ncyc=1000,
  maxcyc=5000,
  ntpr=50,
  cut=8,
  iwrap=1,
  ntr=1,
  restraint_wt=1.0,
  restraintmask='!@H=',
 /
```

mdin.5 (aka min5.mdin)

This is the same as mdin.4 (page 51), except the restraints are reduced to 0.5 kcal mol^{-1}.

```
Minimization to relax initial bad contacts, explicit solvent
 &cntrl
  imin=1,
  ncyc=1000,
  maxcyc=5000,
  ntpr=50,
  cut=8,
  iwrap=1,
  ntr=1,
  restraint_wt=0.5,
  restraintmask='!@H=',
 /
```

mdin.6 (aka min6.mdin)

This is the same as mdin.5 (page 52), except the restraints are reduced to 0.1 kcal mol^{-1}.

```
Minimization to relax initial bad contacts, explicit solvent
 &cntrl
  imin=1,
  ncyc=1000,
  maxcyc=5000,
  ntpr=50,
  cut=8,
  iwrap=1,
  ntr=1,
  restraint_wt=0.1,
  restraintmask='!@H=',
 /
```

mdin.7 (aka min7.mdin)

The is the same as mdin.6 (page 52), except the restraints have been completely removed.

```
Minimization to relax initial bad contacts, explicit solvent
 &cntrl
  imin=1,
  ncyc=1000,
  maxcyc=5000,
  ntpr=50,
  cut=8,
  iwrap=1,
 /
```

mdin.8 (aka heat.mdin)

Coordinates will be read, and a new simulation will be run. 1,000,000 MD steps will be run with a time step of 2 fs. After 50,000 steps, energy information will be printed to the mdout and mdinfo files. After 50,000 steps, coordinates will be written to the trajectory file, and the restart file will be written to every 50,000 steps. Langevin temperature scaling with collisional frequency of 5.0 ps^{-1} is used, and the starting velocities are picked from a random seed based on the date and time. For SHAKE, bonds with hydrogen are constrained and hydrogen atoms are ignored. The nonbonded cutoff is 8 Å. The system is subjected to the constant pressure periodic boundary, and constant pressure MD is used with isotropic position scaling. Wrapping is performed for files, the outputs are written as binary NetCDF files, and NMR restraints and weight changes are given. Atoms with the specified restraint mask are restrained at the weight of 10 kcal mol^{-1}. The NMR-

style restraints describe the temperature increases. The temperature at time step 0 is 10.0 K, and linearly increases until reaching 100.0 K at time step 100,000. From there, the temperature linearly increases until reaching 300.0 K at time step 500,000.

```
Explicit solvent initial heating mdin
 &cntrl
  imin=0, irest=0, ntx=1,
  ntpr=50000, ntwx=50000, ntwr=50000, nstlim=1000000,
  dt=0.002, ntt=3, gamma_ln=5.0, ig=−1,
  ntc=2, ntf=2, cut=8, ntb=2, ntp=1,
  iwrap=1, ioutfm=1, nmropt=1,
  ntr=1, restraint_wt=10, restraintmask=':1−603'
 /
 &wt
  TYPE='TEMP0', ISTEP1=0, ISTEP2=100000,
  VALUE1=10.0, VALUE2=100.0,
 /
 &wt
  TYPE='TEMP0', ISTEP1=100001, ISTEP2=500000,
  VALUE1=100.0, VALUE2=300.0,
 /
 &wt TYPE='END' /
```

`mdin.9` (aka `eq1.mdin`)

Coordinates will be read, and a new simulation will be run. 250,000 MD steps will be run with a time step of 2 fs. After 10,000 steps, energy information will be printed to the mdout and mdinfo files. After 10,000 steps, coordinates will be written to the trajectory file, and the restart file will be written to every 10,000 steps. Langevin temperature scaling with collisional frequency of 1.0 ps^{-1} is used, and the starting velocities are picked from a random seed based on the date and time. For SHAKE, bonds with hydrogen are constrained and hydrogen atoms are ignored. The nonbonded cutoff is 9 Å. The system is subjected to the constant pressure periodic boundary, and constant pressure MD is used with isotropic position scaling. Wrapping is performed for files, and the outputs are written as binary NetCDF files. Atoms with the specified restraint mask are restrained at the weight of 10 kcal mol^{-1}.

```
Explicit solvent molecular dynamics constant pressure MD
 &cntrl
   imin=0, irest=0, ntx=1,
   ntpr=10000, ntwx=10000, ntwr=10000, nstlim=250000,
   dt=0.002, ntt=3, tempi=300,
   temp0=300, gamma_ln=1.0, ig=-1,
   ntp=1, ntc=2, ntf=2, cut=9,
   ntb=2, iwrap=1, ioutfm=1,
   ntr=1, restraint_wt=10, restraintmask=':1-603'
 /
```

`mdin.10` (aka `eq2.mdin`)

The is the same as mdin.9 (page 54), except the restraints are reduced to 5.0 kcal mol^{-1}.

```
Explicit solvent molecular dynamics constant pressure MD
 &cntrl
   imin=0, irest=0, ntx=1,
   ntpr=10000, ntwx=10000, ntwr=10000, nstlim=250000,
   dt=0.002, ntt=3, tempi=300,
   temp0=300, gamma_ln=1.0, ig=-1,
   ntp=1, ntc=2, ntf=2, cut=9,
   ntb=2, iwrap=1, ioutfm=1,
   ntr=1, restraint_wt=5.0, restraintmask=':1-603'
 /
```

`mdin.11` (aka `eq3.mdin`)

The is the same as mdin.10 (page 55), except the restraints are reduced to 2.0 kcal mol^{-1}.

```
Explicit solvent molecular dynamics constant pressure MD
 &cntrl
   imin=0, irest=0, ntx=1,
   ntpr=10000, ntwx=10000, ntwr=10000, nstlim=250000,
   dt=0.002, ntt=3, tempi=300,
   temp0=300, gamma_ln=1.0, ig=-1,
   ntp=1, ntc=2, ntf=2, cut=9,
   ntb=2, iwrap=1, ioutfm=1,
   ntr=1, restraint_wt=2.0, restraintmask=':1-603'
  /
```

`mdin.12` (aka `eq4.mdin`)

The is the same as mdin.11 (page 55), except the restraints are reduced to 1.0 kcal mol^{-1}.

```
Explicit solvent molecular dynamics constant pressure MD
 &cntrl
   imin=0, irest=0, ntx=1,
   ntpr=10000, ntwx=10000, ntwr=10000, nstlim=250000,
   dt=0.002, ntt=3, tempi=300,
   temp0=300, gamma_ln=1.0, ig=-1,
   ntp=1, ntc=2, ntf=2, cut=9,
   ntb=2, iwrap=1, ioutfm=1,
   ntr=1, restraint_wt=1.0, restraintmask=':1-603'
  /
```

`mdin.13` (aka `eq5.mdin`)

The is the same as mdin.12 (page 56), except the restraints are reduced to 0.5 kcal mol^{-1}.

```
Explicit solvent molecular dynamics constant pressure MD
 &cntrl
  imin=0, irest=0, ntx=1,
  ntpr=10000, ntwx=10000, ntwr=10000, nstlim=250000,
  dt=0.002, ntt=3, tempi=300,
  temp0=300, gamma_ln=1.0, ig=-1,
  ntp=1, ntc=2, ntf=2, cut=9,
  ntb=2, iwrap=1, ioutfm=1,
  ntr=1, restraint_wt=0.5, restraintmask=':1-603'
 /
```

mdin.14 (aka eq6.mdin)

The is the same as mdin.13 (page 56), except the restraints are reduced to 0.1 kcal mol^{-1}.

```
Explicit solvent molecular dynamics constant pressure MD
 &cntrl
  imin=0, irest=0, ntx=1,
  ntpr=10000, ntwx=10000, ntwr=10000, nstlim=250000,
  dt=0.002, ntt=3, tempi=300,
  temp0=300, gamma_ln=1.0, ig=-1,
  ntp=1, ntc=2, ntf=2, cut=9,
  ntb=2, iwrap=1, ioutfm=1,
  ntr=1, restraint_wt=0.1, restraintmask=':1-603'
 /
```

mdin.15 (aka eq7.mdin)

The restraints are removed from mdin.14 (page 57) for the "final" stage of equilibration.

```
Explicit solvent molecular dynamics constant pressure MD
 &cntrl
  imin=0, irest=0, ntx=1,
  ntpr=10000, ntwx=10000, ntwr=10000, nstlim=250000,
  dt=0.002, ntt=3, tempi=300,
  temp0=300, gamma_ln=1.0, ig=-1,
  ntp=1, ntc=2, ntf=2, cut=9,
  ntb=2, iwrap=1, ioutfm=1,
 /
```

mdin.16 (aka md.mdin)

Coordinates and velocities will be read, and a the simulation will be restarted using the coordinates and velocities from the restart file. 12,500,000 MD steps will be run with a time step of 2 fs. After 10,000 steps, energy information will be printed to the mdout and mdinfo files. After 10,000 steps, coordinates will be written to the trajectory file, and the restart file will be written to every 10,000 steps. Langevin temperature scaling with collisional frequency of 1.0 ps^{-1} is used, and the starting velocities are picked from a random seed based on the date and time. For SHAKE, bonds with hydrogen are constrained and hydrogen atoms are ignored. The nonbonded cutoff is 9 Å. The system is subjected to the constant pressure periodic boundary, and constant pressure MD is used with isotropic position scaling. Wrapping is performed for files, and the outputs are written as binary NetCDF files. Atoms with the specified restraint mask are restrained at the weight of 10 kcal mol^{-1}.

```
Explicit solvent molecular dynamics constant pressure 25 ns MD
 &cntrl
  imin=0, irest=1, ntx=5,
  ntpr=50000, ntwx=50000, ntwr=50000, nstlim=12500000,
  dt=0.002, ntt=3, tempi=300,
  temp0=300, gamma_ln=1.0, ig=-1,
  ntp=1, ntc=2, ntf=2, cut=9,
  ntb=2, iwrap=1, ioutfm=1,
 /
```

> ❶ **Note:** You can use a a negative value for `ntwr`, and this will give each restart file will have a unique name and make it so they won't be overwritten. Each restart file would be named like `WT_protein_md001.rst7_50000`.

PDBs and SNPs

PDB stands for Protein Data Bank and is a file extension type that is used for crystal structures by the RCSB PDB . Once resolved, crystal structures are published on the website (see below), which provides information about the specifics of the crystals structure and relevant citations for the protein of interest. Published structures are assigned a PDBID, which is an identifying set of numbers and letters for that specific structure.

A page for a recently resolved crystal structure (PDB ID: 5O6O).

The PDB file (here being the data file with a `.pdb` extension) can be downloaded from a box on the right-hand side of the structure's page, shown below. This file will contain a lot of extraneous information, which is why you will likely end up using a command like:

```
$ grep -e '^ATOM\|^HETATM\|^TER\|^END' PDBID.pdb > PDBID_clea
n.pdb
```

to extract out any lines that start with ATOM, HETATM, TER, or END, and save them to a new file.

The data in the PDB format is arranged into the following columns:

- Record Type

- Atom Number

- Residue Name / resname

- Chain Identifier (It is likely everything is the same chain)

- Residue Number / RESID

- X orthogonal coordinate

- Y orthonogonal coordinate

- Z orthogonal coordinate

- Occupancy

- Temperature Factor

- Segment ID (Essentially obsolete by everyone/thing except Chimera)

- Element Symbol
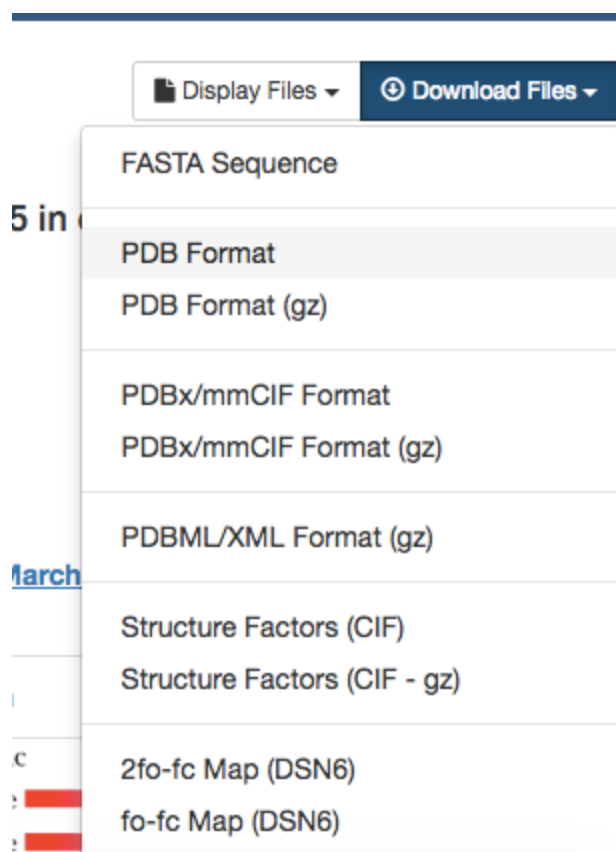
Download options from the RCSB website.

A very important part of the PDB page, especially when dealing with SNPs (page 68) is the sequence tab for a given structure. The information on that tab connects the computational structure with the actual numbering of the protein. For instance, in the figure below, the downloadable PDB file sets has residue `1` as `G`, but what

experimentalists consider residue 1 is actually an M . Knowing this conversion is not only important so that the correct residues are mutated for SNP studies, but also for working with collaborators and publishing results.

Sequence Chain View for 5O6O from the RCSB website .

Amino Acids and DNA and RNA Bases

The central dogma of biology is "DNA makes RNA makes protein." DNA consists of base pairs of adenine, cytosine, guanine, and thymine. These four bases (which change to adenine, cytosine, guanine, and uracil in RNA) end up becoming proteins through translation. In translation, three of these bases, known as codons on messenger RNA, code for a specific amino acid. Which amino acid, or residue, that they code for can be found by looking at the codon wheel (shown below). Sequences of these amino acids are what make up proteins.

Figure: Codon Wheel
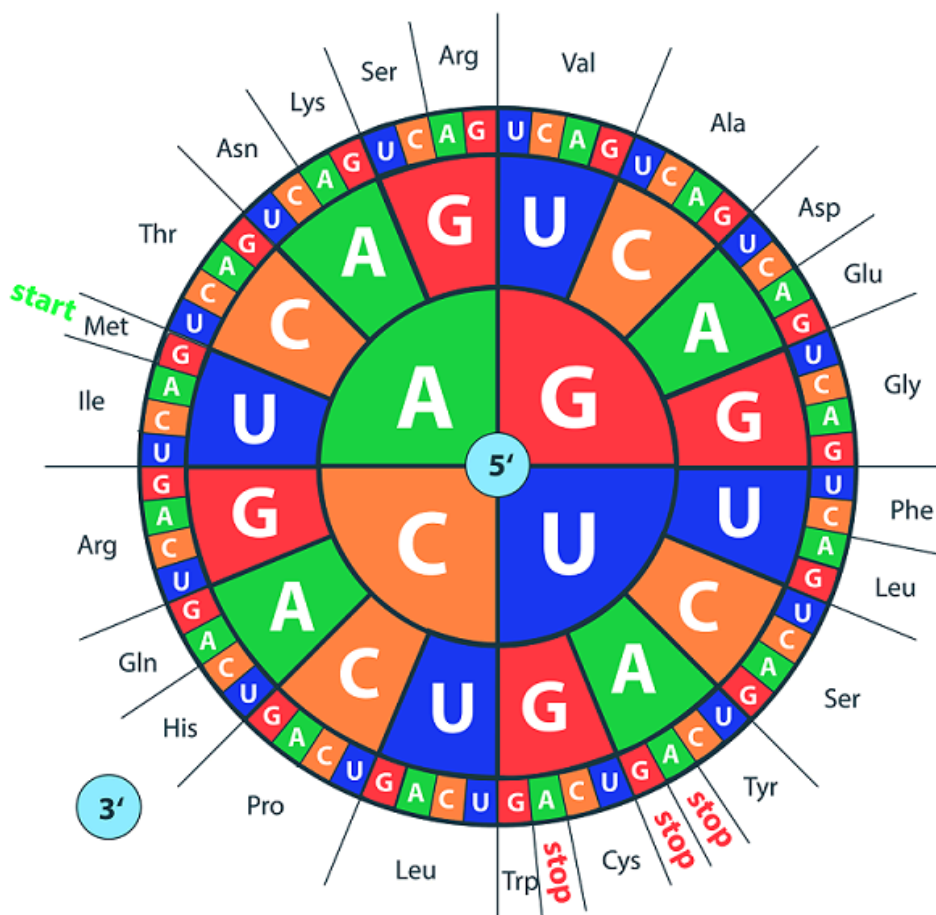
Codon wheel from the Millipore Sigma website .

Different mutations in DNA can propagate through to proteins. Some mutations, known as point-nonsense mutations, result when a mutation at a single residue results in an early stop codon. If a situation like that occurs in a protein that is critical for development, it is unlikely to result in a viable fetus. Similarly, frameshift mutations occur when the addition or loss of a DNA base messes with the reading frame of a group of 3 DNA bases. Thus, single or duplicate insertions would change the outcome of the protein, usually rendering it nonfunctional. Missense mutations are point mutations where a single nucleotide change results in a different amino acid. Other types include insertions, deletions, duplications, and repeat expansions, which will not be discussed here.

DNA consists of a phosphate backbone with different bases. Watson-Crick base pairs (meaning traditional base pairs) for DNA are dG:dC and dA:dT, where A, C, G, and T are adenine, cytosine, guanine, and thymine. In RNA, thymine is replaced by uracil to make up the traditional bases. Adenine and guanine are purines, meaning they have fused imidazole (5-member) and pyrimidine (6-member) rings. Cytosine, thymine, and uracil are pyridines with just the pyrimidine (6-member) ring. The different bases are shown below.

Figure: DNA Bases

Deoxycytidine, dC   Deoxyguanosine, dG   Cytidine, C   Guanosine, G

Deoxythymidine, dT   Deoxyadenosine, dA   Uridine, U   Adenosine, A

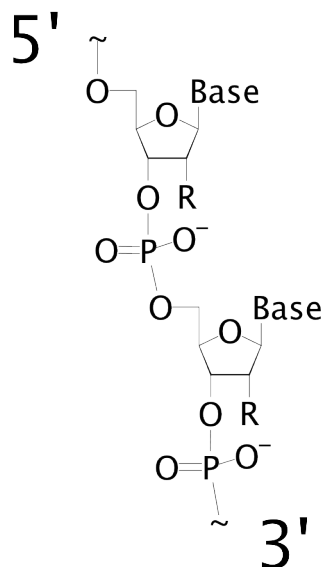Deoxyribonucleosides   Ribonucleosides
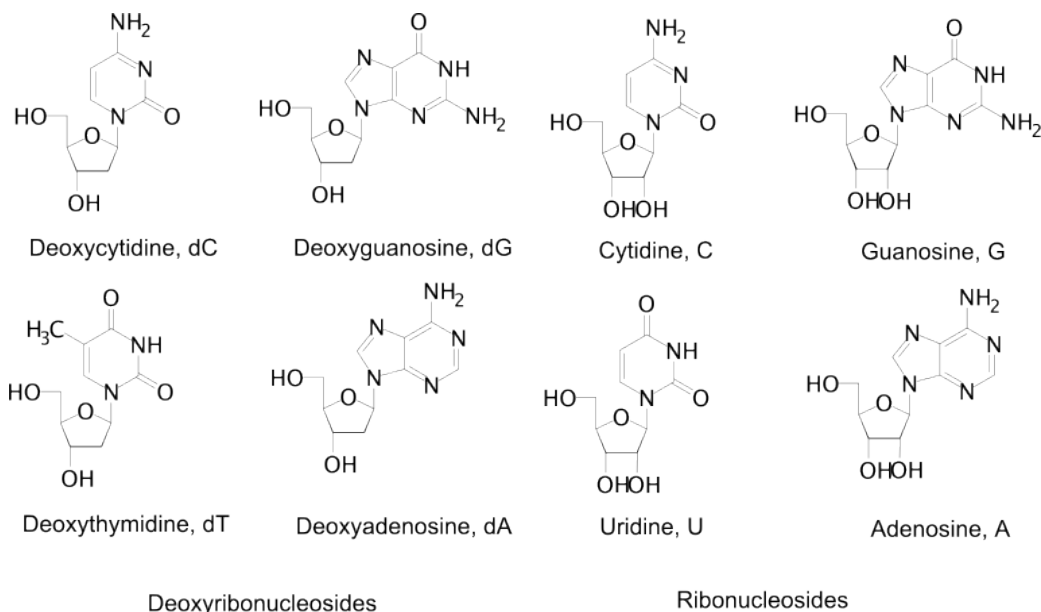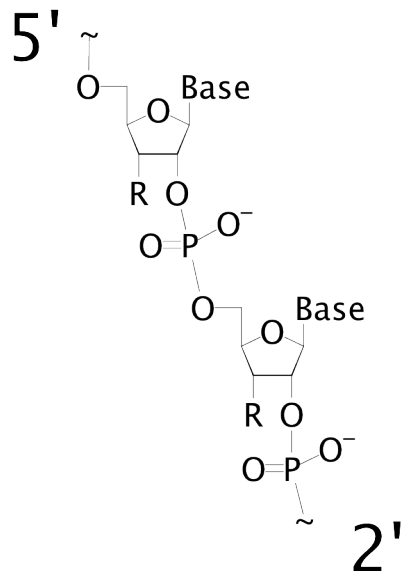
The different DNA and RNA bases.

It is also important to know how the different DNA (and RNA) residues bind to each other. The typical backbone is linked through the 3' and 5' ends. In the *iso* forms, the backbone is linked through the 2' and 5' ends. These differences are shown in below.

Figure: DNA Binding

DNA (R=H)
RNA (R=OH)

iso–DNA (R=H)
iso–RNA (R=OH)

The linking scheme for both the traditional and *iso* forms of DNA and RNA.

Amino acids are known by full names, single-letter codes, and three-letter codes. A list of these names and codes can be found with the skeletal structure in the figures at the end of the page.

The AMBER atom and residue naming is also shown in the following figures. Atom naming follows the Greek alphabet, starting with the alpha carbon (A) and moving on to beta (B), gamma (G), delta (D), epsilon (E), zeta (Z), and eta (H). In AMBER, there are additional 3-letter codes for residues with several protonation states that deviate from the traditional amino acid pattern. Additionally, the naming patterns for DNA and RNA bases are shown.

When a protein complexes with a prosthetic group, such as a DNA strand, then that structure is said to have **apo** and **holo** forms. In short, Mark came up with "Apo absent, holo has."

Apoprotein: the protein part of an enzyme that is missing its prosthetic group. Think of it like a bear that's missing their hat–they can exist without it, but they're much happier with it.
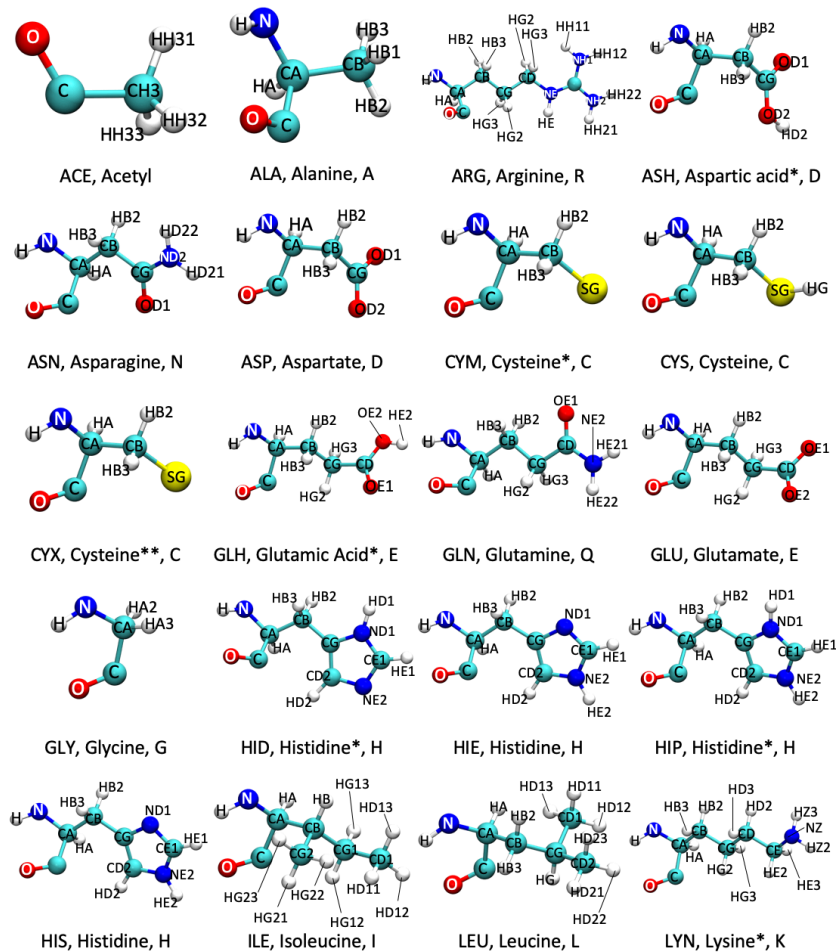
Holoprotein: the apoprotein combined with its prosthetic group. The bear has located their hat again and is now wearing it.

Figure: Amino Acids
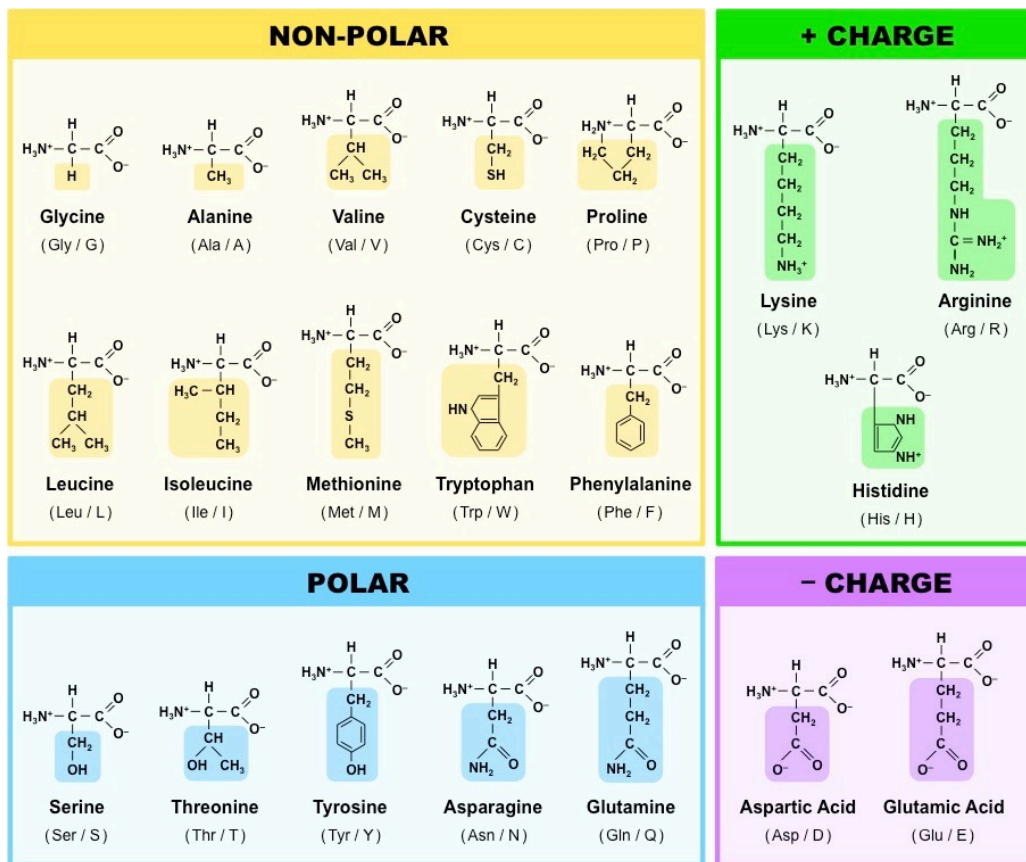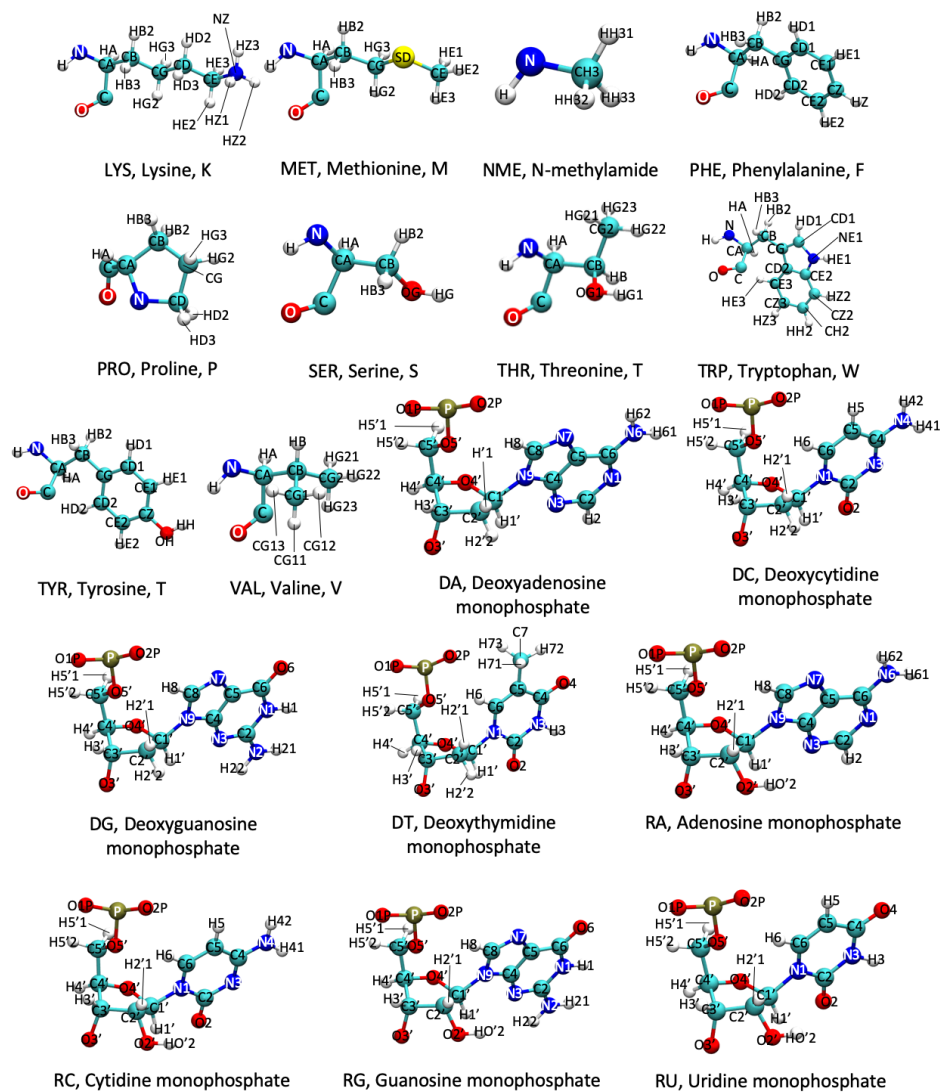

Amino acid chart from the Socratic website .

Figure: AMBER Amino Acids 1

Amino acid chart with AMBER naming (consistent with ff99SB and higher).

Figure: AMBER Amino Acids 2

LYS, Lysine, K   MET, Methionine, M   NME, N-methylamide   PHE, Phenylalanine, F

PRO, Proline, P   SER, Serine, S   THR, Threonine, T   TRP, Tryptophan, W

TYR, Tyrosine, T   VAL, Valine, V   DA, Deoxyadenosine monophosphate   DC, Deoxycytidine monophosphate

DG, Deoxyguanosine monophosphate   DT, Deoxythymidine monophosphate   RA, Adenosine monophosphate

RC, Cytidine monophosphate   RG, Guanosine monophosphate   RU, Uridine monophosphate

Amino acid chart with AMBER naming (consistent with ff99SB and higher).

Creating SNP Mutations in Chimera

A number of studies recently have focused on what single nucleotide polymorphisms (SNPs) do to a protein's function. Changes in a single nucleotide that are found in less than 1\% of human populations are SNPs. These SNPs can be missense mutations. PDB files generated from a crystal structure are typically in the wild type, meaning they do not contain the SNP. Changing a wild type to a known SNP can be performed using the program UCSF Chimera .

SNPs should be added after the wild type structure has been prepared (i.e. any linkers or specific modifications to match an experimental study have been added). After the wild type structure has been prepared (and saved), then in can be opened in Chimera. It would be wise to make a copy first, so that you do not accidentally overwrite the wild type structure.

Once the structure is pulled up, follow `Favorites → Command Line` in the menu bar to pull up Chimera's command line. In the command prompt, type the following command, where `123` is the residue number of interest. The specific number will be dependent on the protein numbering on the Protein Data Bank website discussed here (page 59).

```
select :123
```

This selects the specific SNP position, and should show it highlighted in green. To then visualize the residue as a stick, follow `Actions → Atoms/Bonds → Show` in the menu bar.

Using the menu bar again, follow `Tools → Structure Editing → Rotamers` to bring up the rotamers screen (see the image below). The rotamers screen allows the current residue in the SNP position, described as the rotamer type (THR in the rotamers image), to the residue changed in the SNP.
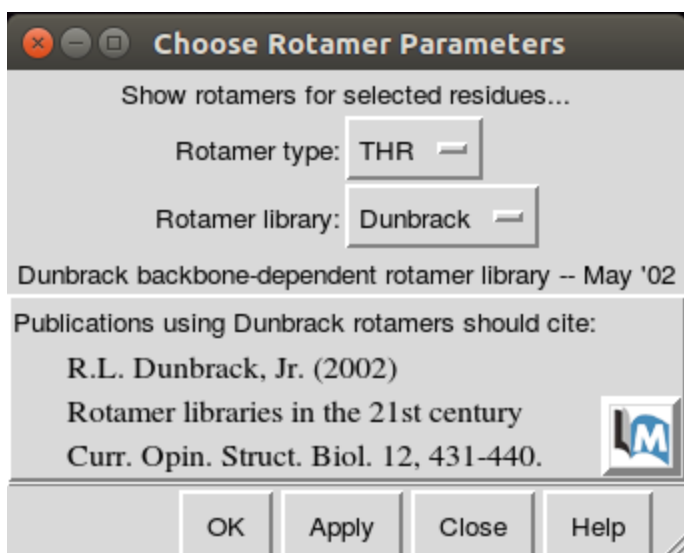
The rotamers screen, which allows you to change the specific residue corresponding to the SNP of interest.

After changing the residue in the drop down menu, select `OK`. You will then be given probabilities to pick from as to how likely it is that that specific rotation of the amino acid would occur. Clicking on each one will show the proposed orientation. Most likely, though, the highest probability rotamer will be the best choice. The "Existing side chain(s)" box should remain as `replace`, as the residue is being replaced by the changed residue. The probability that is highlighted is the one that is being used to replace t he original residue, and its stated probability should be written down in your lab notebook before changing it with `OK`.

Once the residue is changed, then a new PDB should be saved, indicating the SNP in the file name. This can be achieved through `File → Save PDB`. If multiple things are open in Chimera, highlight the appropriate structure under `Save models`. Specify the file location, hit save, and then do a happy dance; you just mutated a residue in Chimera!

Creating SNP Mutations in *LEaP*

SNPs can also be created by removing the GUI Chimera offers and manually changing residues. For instance, if you wanted to change isoleucine (ILE) to leucine (LEU), then you would find the ILE in the PDB and remove everything that was not the atoms named C, CA, N, or O.

Your original ILE

```
ATOM    129  N    ILE    10    -15.372  24.194 -43.088  1.00
0.00
ATOM    130  H    ILE    10    -16.116  24.609 -42.545  1.00
0.00
ATOM    131  CA   ILE    10    -14.509  23.251 -42.446  1.00
0.00
ATOM    132  HA   ILE    10    -13.546  23.173 -42.950  1.00
0.00
ATOM    133  CB   ILE    10    -14.289  23.510 -40.982  1.00
0.00
ATOM    134  HB   ILE    10    -13.900  24.518 -40.840  1.00
0.00
ATOM    135  CG2 ILE    10    -15.631  23.370 -40.244  1.00
0.00
ATOM    136 HG21 ILE    10    -16.020  22.362 -40.385  1.00
0.00
ATOM    137 HG22 ILE    10    -15.482  23.556 -39.180  1.00
0.00
ATOM    138 HG23 ILE    10    -16.343  24.093 -40.643  1.00
0.00
ATOM    139  CG1 ILE    10    -13.167  22.603 -40.448  1.00
0.00
ATOM    140 HG12 ILE    10    -13.563  21.588 -40.404  1.00
0.00
ATOM    141 HG13 ILE    10    -12.348  22.642 -41.167  1.00
0.00
ATOM    142  CD1 ILE    10    -12.652  23.009 -39.067  1.00
0.00
ATOM    143 HD11 ILE    10    -13.470  22.970 -38.348  1.00
0.00
ATOM    144 HD12 ILE    10    -11.863  22.324 -38.757  1.00
0.00
ATOM    145 HD13 ILE    10    -12.255  24.023 -39.110  1.00
0.00
ATOM    146  C    ILE    10    -15.173  21.926 -42.614  1.00
0.00
ATOM    147  O    ILE    10    -16.368  21.785 -42.360  1.00
0.00
```

becomes

```
ATOM    129  N    ILE    10     -15.372  24.194 -43.088  1.00
0.00
ATOM    131  CA   ILE    10     -14.509  23.251 -42.446  1.00
0.00
ATOM    146  C    ILE    10     -15.173  21.926 -42.614  1.00
0.00
ATOM    147  O    ILE    10     -16.368  21.785 -42.360  1.00
0.00
```

Finally, to change the SNP residue, change the residue name (resname) from ILE to LEU.

```
ATOM    129  N    LEU    10     -15.372  24.194 -43.088  1.00
0.00
ATOM    131  CA   LEU    10     -14.509  23.251 -42.446  1.00
0.00
ATOM    146  C    LEU    10     -15.173  21.926 -42.614  1.00
0.00
ATOM    147  O    LEU    10     -16.368  21.785 -42.360  1.00
0.00
```

Now you have what *LEaP* will see as a leucine, auto-filling the missing atoms. You can just leave the wonky non-sequential numbering, as *LEaP* also renumbers everything when it runs.

Addressing Multiple Protonation States

If you've never taken general chemistry, then the title here probably makes zero sense. If you *have* taken general chemistry, then the words "multiple protonation states" might spark panic. Never fear! With help of the Internet (and a little ~chemical intuition~), our systems will be biologically relevant!

So what do I mean by multiple protonation states? You've probably heard of (and used) ammonia at some point. Ammonia is NH_3. Its conjugate acid, an ammonium ion, is NH_4^+. They differ by–you guess it–a single hydrogen. They're pretty similar things, and whether or not that hydrogen is there is pH-dependent.

Why should you care? Certain amino acids, like histidine (*sigh*), can have multiple protonation states. That's actually a good thing, because then you get stuff like acid-base catalysis. But that also means things like histidine can be the difference between accurate or garbage simulations.

There are five amino acids that can have weird protonation states. They are: aspartate, cysteine, glutamate, histidine, and lysine.

Figure: Protonation States
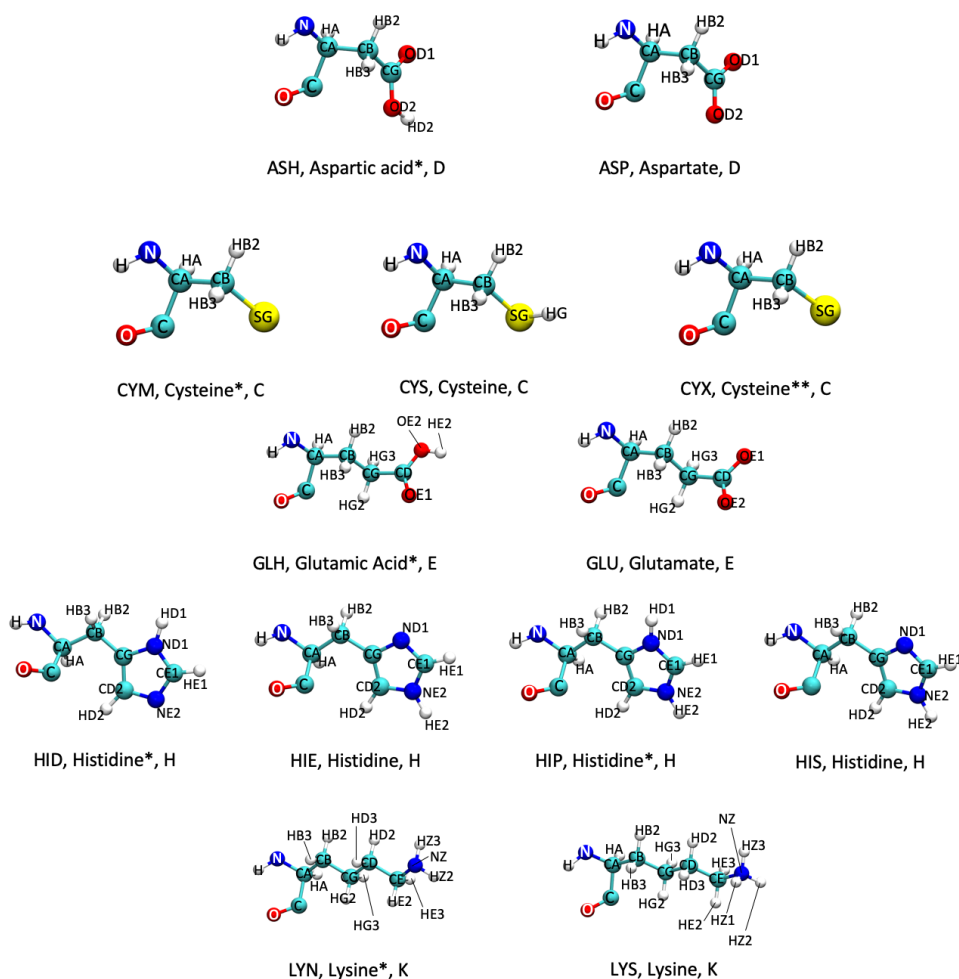
The different protonation states of aspartate, cysteine, glutamate, histidine, and lysine.

There are a few ways to ensure that your protein is protonated correctly. These are described below.

PROPKA (or the web version PDB2PQR) is a program intended to predict the correct protonation states of protein residues. When a PDB file is downloaded from RCSB , there are no hydrogens. This is because hydrogens move too quickly and have so little mass that they cannot be resolved using the determination method (e.g., x-ray crystallography). Most MD programs will interpret residue names literally, which means that `HIS` will be interpreted as a singly-protonated histidine residue. However, that histidine residue could be protonated in other ways, such as doubly-protonated or oppositely singly-protonated (see the residue definitions here (page 62) for more information). That's why it is important to use something like PROPKA or the H++ webserver to properly protonate the system.

Cleaning the Starting Structure

The PDB downloaded from RCSB has a lot of information in it (as mentioned on the PDB page (page 59)). There's a lot of stuff that is important for crystallographers that is honestly not well-taught to computational chemists. Luckily, RCSB created a website called PDB 101 that can explain a lot of that stuff.

As part of the PDB, there are often multiple chains. Sometimes you need multiple chains (like in hemoglobin) for a complete system. If there are multiple chains, there is still a chance you don't need more than one–often, symmetric dimers will be simulated as a monomer because it cuts down on the computational cost. There are assuredly times when you need the dimer, like if you want to study non-symmetric mutations, but that'll be fleshed out when you're preparing (or re-preparing) your structure. These chain IDs come *after* the residue name (e.g., `ACE A`).

Before the residue name, there will occasionally be additional letters (e.g., `AACE`). These letters specify that there are alternate conformations for that residue. The `REMARK` lines would provide more guidance as to how much either of those conformations are favored. Usually, using `A` is fine, but it's better to spend more time investigating at the beginning than to have to redo everything.

The following script (`clean-pdb.sh`) will read in the PDB downloaded from RSCB. Only the relevant lines will be read (i.e., `REMARK` likes will be ignored). Any B-conformations will be removed (only `A` will be saved). Water lines labeled `HOH` will be renamed `WAT` . The final lines that are commented out would get rid of common inhibitors. You can check the PDB's RCSB page to see what small molecules were included. The paper *hopefully* published with the structure would explicitly state which inhibitor was used.

```bash
#!/bin/bash

## Define your files
thing=RCSB.pdb
thing_clean=RCSB_clean.pdb

## Clean the RCSB PDB (Remove crystal junk)
grep -e '^ATOM\|^HETATM\|^TER\|^END' $thing > $thing_clean

## The remainder will only act on lines containing the specifie
d info

## Remove any B chain lines
sed -i '/BALA/d' $thing_clean
sed -i '/BARG/d' $thing_clean
sed -i '/BASN/d' $thing_clean
sed -i '/BASP/d' $thing_clean
sed -i '/BCYS/d' $thing_clean
sed -i '/BGLU/d' $thing_clean
sed -i '/BGLN/d' $thing_clean
sed -i '/BGLY/d' $thing_clean
sed -i '/BHIS/d' $thing_clean
sed -i '/BILE/d' $thing_clean
sed -i '/BLEU/d' $thing_clean
sed -i '/BLYS/d' $thing_clean
sed -i '/BMET/d' $thing_clean
sed -i '/BPHE/d' $thing_clean
sed -i '/BPRO/d' $thing_clean
sed -i '/BSER/d' $thing_clean
sed -i '/BTHR/d' $thing_clean
sed -i '/BTRP/d' $thing_clean
sed -i '/BTYR/d' $thing_clean
sed -i '/BVAL/d' $thing_clean

## Rename A chain as only chain
sed -i 's/AALA/ ALA/g' $thing_clean
sed -i 's/AARG/ ARG/g' $thing_clean
sed -i 's/AASN/ ASN/g' $thing_clean
sed -i 's/AASP/ ASP/g' $thing_clean
sed -i 's/ACYS/ CYS/g' $thing_clean
sed -i 's/AGLU/ GLU/g' $thing_clean
sed -i 's/AGLN/ GLN/g' $thing_clean
sed -i 's/AGLY/ GLY/g' $thing_clean
sed -i 's/AHIS/ HIS/g' $thing_clean
sed -i 's/AILE/ ILE/g' $thing_clean
```

```
sed -i 's/ALEU/ LEU/g' $thing_clean
sed -i 's/ALYS/ LYS/g' $thing_clean
sed -i 's/AMET/ MET/g' $thing_clean
sed -i 's/APHE/ PHE/g' $thing_clean
sed -i 's/APRO/ PRO/g' $thing_clean
sed -i 's/ASER/ SER/g' $thing_clean
sed -i 's/ATHR/ THR/g' $thing_clean
sed -i 's/ATRP/ TRP/g' $thing_clean
sed -i 's/ATYR/ TYR/g' $thing_clean
sed -i 's/AVAL/ VAL/g' $thing_clean

## Rename HOH as WAT
sed -i 's/HOH/WAT/g' $thing_clean

## Optional: Delete inhibitor/artifact lines
## Check the "Small Molecules" section of RCSB
## And compare against paper/common inhibitors for enzyme class
#sed -i '/ACT /d' $thing_clean
#sed -i '/OGA /d' $thing_clean
#sed -i '/SO4 /d' $thing_clean
#sed -i '/GOL /d' $thing_clean
#sed -i '/FSU /d' $thing_clean
#sed -i '/EDO /d' $thing_clean
```

PROPKA

The PDB2PQR web version of PROPKA has a list of selections to make. By default, it uses the `PARSE` forcefield, but you will likely want to select `AMBER`. Similarly, you will want to pick an output naming scheme, which will again likely be `AMBER`. The remaining things are preselected, and they should be fine:

- Ensure that new atoms are not rebuilt too close to existing atoms

- Optimize the hydrogen bonding network

- Create an APBS input file (this also enables the option to run APBS and visualize your results through the web interface, if it has been installed) Finally, there are pKa options. You should be able to use `pH 7`, but if there was a crystal structure paper for the PDB and that has a pH listed, you may want to consider using that pH. Keep the default "Use PROPKA to assign protonation states at provided pH" selection.

Once you've submitted the job, download the files. The `PQR` file will contain the residues with differing protonation states. The following script (`ph_changes_propka.sh`) will select all these lines and save them to a new file.

```bash
#!/bin/bash

## Define your files
thing=RCSB.pqr
thing_clean=RCSB_ph_list.pqr

## Get the non-standard protonation states
grep -e 'ASH\|CYM\|CYX\|GLH\|HID\|HIP\|LYN' $thing > $thing_cle
an
```

That didn't have to be a script, but it seemed like it would be annoying to type out in the future.

You can then copy these non-standard protonation lines into the original PDB.

After using PROPKA, you can then use MolProbity (page 82) to add missing hydrogen atoms and check whether ASN, GLU, and HIS residues need to be flipped.

H++

The H++ webserver can also be used to determine the protonation state of titratable residues. It has a few additional advantages, including checking if ASN, GLU, and HIS should be flipped and adding all missing hydrogen atoms. This means H++ is more of a "one stop shop" for PDB preparation.

To use it, select the "Process a Structure" tab.

Figure: H++ Upload

The `Process a Structure` page for H++.

Then, submit! After submission, you get to select some options.

Figure: H++ Options

H++ options.

> ⚠ **Important:** Make sure you've added any missing loops and addressed any non-standard residues before checking the protonation states! H++ **cannot** run on a structure with missing residues!

You will immediately get the following error message if there are missing loops on non-standard residues:

```
THE CALCULATION HAS STOPPED. It appears that some residues are
missing in the middle of the uploaded
structure. When residues are missing (that is the sequence in t
he PDB file is discontiguous) the
accuracy of pK estimates of any kind is affected, especially i
n the vicinity of the gaps. Make sure
there are no "gaps" in your structure, see the FAQ for suggesti
ons. It is also possible that your
structure contains non-standard residue names, sometimes liste
d as HETATM, in the main sequence.
These will not be recognized by the system and will be treated
as missing, resulting in the above
error. Please change the names to comply with the standard amin
o-acid nomenclature. Alternatively,
you can supply your own PQR file -- the naming compliance is no
t enforced then. See the FAQ.
```

The added benefit of H++ is that it *could* be used to solvate and neutralize the system.

MolProbity: Correcting Bad Contacts

MolProbity is a webserver that can be used to add hydrogen atoms to crystal structures, flip ASN, GLU, and HIS residues, and check the overall structure for bad contacts.

> ⚠ **Important:** Run the system through PROPKA (page 77) before using MolProbity!

After determining the protonation states of titratable residues, upload the structure to MolProbity. The system will then be processed and remove all the hydrogens. The next screen is the selection menu.
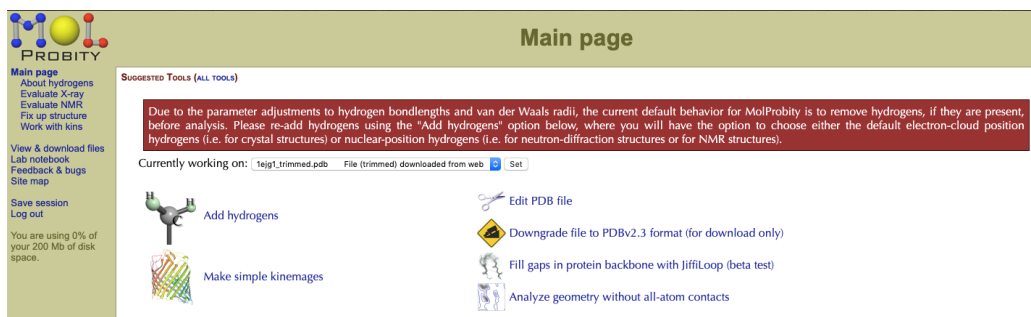
The selection menu for MolProbity.

Select the `Add hydrogens` button to add the hydrogens back to the structure. On the next screen, choose `Asn/Gln/His flips` (the advanced options don't matter). For the x-H bond-length selection, choose the one that matches the PDB's experimental method. As the description says, choose `Electron-cloud x-H` for x-ray crystal structures and `Nuclear x-H` for NMR or neutron diffraction structures.

After a few more pages with details on adding hydrogens or flips, you'll get to a new selection menu. Now you'll have the choice to `Analyze all-atom contacts and geometry`. Select this option, keeping the `FH` version of the PDB selected. The default selections are likely fine, unless you need a particular setting.

The all-atom contact analysis gives a clashscore. Structures with less than 90% likely need to be fixed a little bit. The downloadable files from MolProbity will provide insight into which residues are clashing, and you can use a program like Chimera to select rotamers at those positions.

> ℹ **Note:** The downloads are located in the block under the selection menu.

Non-Standards

Sometimes new systems will have a non-standard residue that needs to be parameterized in order to run molecular dynamics. What does that mean? Well, it means your research project is so special that nobody's every needed (or if they did, they didn't make it public) the set of parameters for something in your system. That said, a few hours scouring the internet could save you the pain and suffering (fine, maybe just the time and mild annoyance) of having to generate parameters for non-standard residues. Two good places to check are the **R**ESP **E**SP charge **DD**ata **B**ase Home Page (they were trying real hard to stay with a theme…) and the University of Manchester AMBER parameter data base .

If you're not lucky enough to have pre-generated parameters, then you get the pure and utter joy of either using *antechamber* in AMBER yourself (which isn't all that bad for small organic compounds) or uploading some information to PyRED program interfaced by RED Server Development . PyRED is an interactive submission process that generates the wanted `frcmod` , in addition to the `mol2` required for making the `prepi` file. PyRED has a tutorial available on how to generate certain types of non-standard residues.

Uploads to PyRED

First things first: PyRED knows what it wants. If it wants something titled a particular way, it will only accept things with that particular title. So treat PyRED like the royalty it is and don't disobey it. Second things second: if your school has a Gaussian license, you can apply for an academic account that will allow you to run the quantum calculation with Gaussian (you don't have to use Gaussian, in this instance, so don't worry).

> ☑ **Tip:** PyRED's account registration is finicky, and the Captcha is intense. Pay attention to the differences between uppercase and lowercase letters when making an account, and save your assigned login and password, since you don't get to change it.

Nucleotide Fragment Generation

PyRED creates parameters for nucleotides in fragments.

Dimethylphosphate Portion of PDB (`Mol_red1.pdb`)

```
ATOM      1  C1  DMP     1       1.100   0.000   0.000
ATOM      2  H11 DMP     1       0.000   0.000   0.000
ATOM      3  H12 DMP     1       1.476   1.034   0.000
ATOM      4  H13 DMP     1       1.476  −0.484   0.913
ATOM      5  O3' DMP     1       1.524  −0.687  −1.135
ATOM      6  P   DMP     1       3.138  −0.847  −1.398
ATOM      7  O1P DMP     1       3.811   0.378  −0.942
ATOM      8  O2P DMP     1       3.281  −1.435  −2.738
ATOM      9  O5' DMP     1       3.475  −2.001  −0.280
ATOM     10  C2  DMP     1       3.096  −3.315  −0.544
ATOM     11  H21 DMP     1       2.018  −3.403  −0.661
ATOM     12  H22 DMP     1       3.568  −3.689  −1.444
ATOM     13  H23 DMP     1       3.401  −3.925   0.302
```

Example `Project.config`

```
# Provide informative titles
MOLECULE1-TITLE          = Dimethylphosphate
MOLECULE2-TITLE          = NucleotideOfInterest

# Providing the total charge for molecule 1 is mandatory
MOLECULE1-TOTCHARGE      = -1
# Providing the total charge for molecule 2 is not mandatory
MOLECULE2-TOTCHARGE      =  0

# Define two inter-mcc between molecule 1 and molecule 2
MOLECULE-INTER-MCC1      = 0.0 | 1 2 |  1  2  3  4 | 1 2
MOLECULE-INTER-MCC1      = 0.0 | 1 2 | 10 11 12 13 | 3 4
```

In the `Project.config` file, the lines with `MOLECULE-INTER-MCC1` describe where the connections need to be made between the two compounds. In the example, molecule 1's `1 2 3 4 | 1 2 `refers to C1, H11, H12, H13, H12, and H13. Molecule 2's` | 10 11 12 13 | 3 4 `refers to C3', H3', C4', H4', O3', and H3T.

Example `System.config`

```
FFPARM = AMBERFF99SB
```

Using *antechamber* and *parmchk* for Ligands

The *antechamber* program is a helpful tool for ligand (i.e. drug and inhibitor) parametrization, assuming you have a pretty typical organic molecule. (If not, you'll need to do this the Gaussian way, or the REDD way.) First, make a PDB file with **only** the ligand molecule–no protein, no metal, just ligand. If your ligand is in the PDB with everything else, copy the lines with the ligand into a new PDB file. If your ligand PDB does not have hydrogens, then those will need to be added. Luckily, AMBER's *reduce* program can do this!

```
$AMBERHOME/bin/reduce ligand_missing_H.pdb > ligand_with_H.pdb
```

Amazing, now you should have hydrogens where they need to be. Now you're ready to use *antechamber*, which may take a little bit of time depending on the size of your ligand, because AMBER will be running a quantum calculation.

```
$AMBERHOME/bin/antechamber –i ligand_with_H.pdb –fi pdb –o liga
nd_with_H.mol2 –fo mol2 –c bcc –s 2 –nc 0 –m 1
```

So, what the fruitcakes did that all mean? First, you're reading in the PDB file (and saying that your `f`ile `i`n is a PDB), telling it what file to write out (and saying that your `f`ile `o`ut is a mol2 file). The `–s 2` tells the program to be verbose, so that all of the information is printed to the Terminal (it's helpful for debugging this command). The `–c bcc` specifies what type of quantum calculation you're running–in this case, it's AM1-BCC (Austin Model 1-Bond Charge Corrections). It's not meant to be super great–it's a quick and dirty calculation, because you're likely doing this same parametrization for a great number of compounds. If you really care about this ligand's parametrization, then you'll want to consider using RESP charges (and now we're back to the **R**ESP **E**SP charge **DD**ata **B**ase Home Page). There are other charge options for *antechamber*, too, which can be found by doing `antechamber –L`. Anyway, the `–nc 0` says that the `n`et `c`harge is zero. If your ligand has a +3 overall charge, use `–nc 3`; for a -2 overall charge you'd use `–nc –2`. If you're lazy and guess 0 when there's really a charge, you'll probably get an error at this step–though you really should take the 20 seconds to determine if there's a charge. Finally, the `–m 1` specifies the multiplicity of the ligand. This is determined through $2S + 1$, where S is the total number of unpaired electrons in the system. Thus, with zero unpaired electrons, this is 1.

Hooray, we've made it through the *antechamber* step, which gave us a ligand with charges in a mol2 file. With that information, we can use *parmchk* to make a force field modification file (known as an frcmod) for the ligand.

```
$AMBERHOME/bin/parmchk -i ligand_with_H.mol2 -f mol2 -o ligand_with_H.frcmod
```

Great! That's done! You now have an frcmod file that has parameters specific to your ligand based on the Generalized Amber Force Field (GAFF) for organic molecules. You're now ready to move onto LEaP (page 91).

AMBER Atom Types

AMBER has specific atom names to describe the environment (and associated bonding) of that atom. Atom typing is not found in the PDB, but it is found in anything dealing with a force field (specifically prepi and frcmod files). The traditional AMBER force field atom types can be found here . The following are the atom types in GAFF, or the General AMBER Force Field, taken from the AMBER website .

Table: Amber Atom Types in General AMBER Force Field (GAFF)

Atom Name	Description	Category
c	sp^2 C in C=O, C=S	basic
c1	sp^1 C	basic
c2	sp^2 C, aliphatic	basic
c3	sp^3 C	basic
ca	sp^2 C, aromatic	basic
n	sp^2 N in amide	basic
n1	sp^1 N	basic
n2	sp^2 N with 2 substituted double bond	basic
n3	sp^3 N with 3 substituted	basic
n4	sp^3 N with 4 substituted	basic
na	sp^2 N with 3 substituted	basic
nh	amine N connected to the aromatic rings	basic
no	N in nitro group	basic
o	sp^2 O in C=O, COO-	basic
oh	sp^3 O in hydroxyl group	basic

Atom Name	Description	Category
os	sp^3 O in ether and ester	basic
s2	sp^2 S (p=S, C=S etc)	basic
sh	sp^3 S in thiol group	basic
ss	sp^3 S in -SR and SS	basic
s4	hypervalent S, 3 substituted	basic
s6	hypervalent S, 4 substituted	basic
hc	H on aliphatic C	basic
ha	H on aromatic C	basic
hn	H on N	basic
ho	H on O	basic
hs	H on S	basic
hp	H on P	basic
p2	sp^2 P (C=P etc)	basic
p3	sp^3 P, 3 substituted	basic
p4	hypervalent P, 3 substituted	basic
p5	hypervalent P, 4 substituted	basic
f	any F	basic
cl	any Cl	basic
br	any Br	basic
i	any I	basic
h1	H on aliphatic C with 1 electron-withdrawing group	special
h2	H on aliphatic C with 2 electron-withdrawing groups	special

Atom Name	Description	Category
h3	H on aliphatic C with 3 electron-withdrawing groups	special
h4	H on aliphatic C with 4 electron-withdrawing groups	special
h5	H on aliphatic C with 5 electron-withdrawing groups	special
cc(cd)	inner sp^2 C in conjugated ring systems	special
ce(cf)	inner sp^2 C in conjugated chain systems	special
cp(cq)	bridge aromatic C	special
cu	sp^2 C in three-memberred rings	special
cv	sp^2 C in four-memberred rings	special
cx	sp^3 C in three-memberred rings	special
cy	sp^3 C in four-memberred rings	special
n	aromatic nitrogen	special
nb	inner sp^2 N in conjugated ring systems	special
nc(nd)	inner sp^2 N in conjugated chain systems	special
sx	conjugated S, 3 substituted	special
sy	conjugated S, 4 substituted	special
pb	aromatic phosphorus	special
pc(pd)	inner sp^2 P in conjugated ring systems	special
pe(pf)	inner sp^2 P in conjugated chain systems	special
px	conjugated P, 3 substituted	special
py	conjugated P, 4 substituted	special

Using *LEaP*

First things first: the AMBER LEaP tutorial is incredibly explanative. *LEaP*, in the forms *tleap* or *xleap* is used to generate AMBER systems.

## Table: Commands in the LEaP syntax

Command	Objective	Example	Ex. Explanation
source	load in force field parameters	`source leaprc.ff14SB`	loads in the ff14SB force field
loadpdb	load in a PDB file	`loadpdb foo sys-tem.pdb`	loads in the PDB, all future references are to `foo`
loadamberprep	load in a prepi file for a non-standard residue	`loadamberprep residue234.prepi`	loads in the topology information for the non-standard residue file `residue234.prepi`
loadamberparams	load in an frcmod file for a non-standard residue	`loadamberparams residue234.frcmod`	load in the force field information for the non-standard residue file `residue234.prepi`
check	make sure that there aren't errors	`check foo`	checks the loaded `foo` for errors
select	choose specific atoms for the Unit editor	`select foo.135`	selects residue 135 of `foo`

Command	Objective	Example	Ex. Explanation
edit	opens the selection in the Unit editor	`edit foo`	opens `foo` the Unit editor; any selected residues will the be highlighted
solvateoct	solvate the system as a truncated octahedron	`solvateoct foo TIP3P 12.00`	solvates the loaded `foo` with TIP3P water extending at least 12.00 Å from the protein's surface
solvatebox	solvate the system as a square box	`solvatebox foo TIP3P 12.00`	solvates the loaded `foo` with TIP3P water extending at least 12.00 Å from the protein's surface
addions	add ions to neutralize the system (commonly K^+, Na^+, or Cl^-)	`addions foo K+ 0`	neutralizes `foo` with potassium ions to a net charge of `0`
saveamberprep	saves a prepi file	`saveamberprep R234 res234-fix.prepi`	saves a new prepi file, which means that fixed systems can be rebuilt with the modified prepi
saveamberparm	save the parameter and topology file	`saveamberparm foo sys-tem_wat.prmtop system_wat.inpcrd`	saves the parameter and topology files for `foo` that will be used for simulation
savepdb	saves a PDB file	`savepdb foo sys-tem.pdb`	saves the PDB file for `foo` system that can serve as an informative reference

Command	Objective	Example	Ex. Explanation
quit	exit out of the program	`quit`	you guessed it… it quits

With LEaP, there are several solvent shapes to pick from. We commonly use periodic solvent boxes, which are generated using either `solvateOct` or `solvateBox` (both shown in the figure below). `solvateOct` solvates the system in a truncated octahedron and `solvateBox` solvates the system in a cuboid box. The `solvateOct` command makes space-filling spherical shape. This reduces solute rotation and often results in smaller systems. Having a smaller system can save time in simulations. Occasionally `solvateOct` has issues with centering itself correctly, but those are few and far between. Choose the solvation command you want and be consistent across that project. Additional details on these commands can be found on page 232 of the Amber18 Manual .
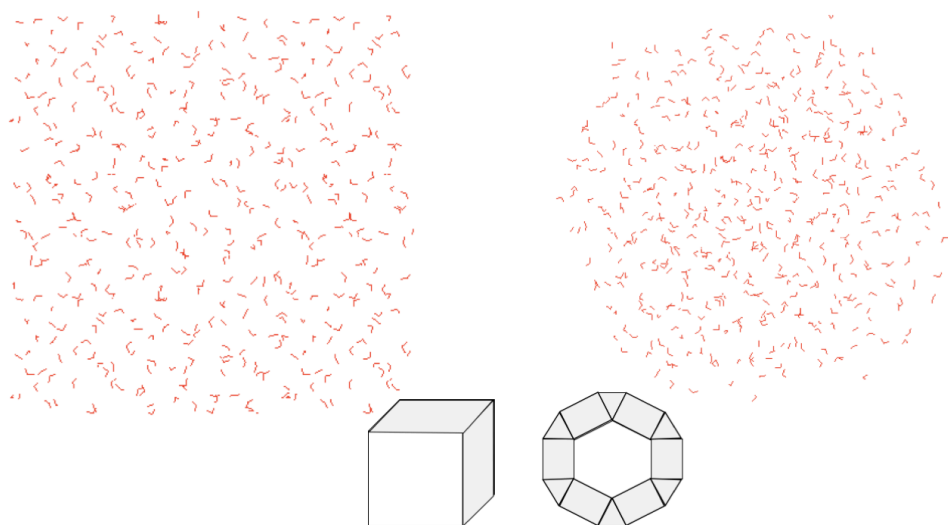
The different types of periodic solvent boxes for explicit solvent, `solvateBox` *(left) and* `solvateOct` *(right).*

tleap: for Command-Line Lovers

tleap is a program that will generate the system from the command line based on an input file containing all the necessary information, written in the same syntax that is used in *xleap*.

An example *tleap* script is:

```
source leaprc.protein.ff14SB
source leaprc.DNA.OL15
source leaprc.gaff
source leaprc.water.tip3p

loadoff ZN2.lib
loadamberparams ZN2.frcmod

loadamberprep NSA.prepi
loadamberparams NSA.frcmod

WTP = loadpdb WT_protein_system.pdb

savepdb WTP WT_protein_system_vac.pdb
saveamberparm WTP WT_protein_system_vac.prmtop WT_protein_syste
m_vac.inpcrd

addions WTP K+ 0.0
solvatebox WTP TIP3PBOX 12.0

savepdb WTP WT_protein_system_wat.pdb
saveamberparm WTP WT_protein_system_wat.prmtop WT_protein_syste
m_wat.inpcrd
quit
```

When running a *tleap* script, you can specify the input file with the `-f` flag.

```
$ $AMBERHOME/bin/tleap -f tleap-script.in
```

tleap prints a lot of information to the Terminal, and you can watch for any errors. Of course, you can send this information to an out file, if you don't want to watch everything fail before your eyes (just make sure it didn't fail before moving on). The

information that is printed is also saved to an appending `leap.log` file, so the log from every attempt at creating your system is saved without overwriting by default.

```
$ $AMBERHOME/bin/tleap -f tleap-script.in > tleap-attempt.out
```

xleap: for GUI Lovers and the Paranoid

xleap is a GUI version of AMBER's *LEaP* program. However, it can be deeply frustrating to work with for a few reasons. First, you cannot select a specific cursor position–you must backspace any previous parts of a command that you've made mistakes in typing. Second, you cannot have numlock enabled when using *xleap* because of various components in the edit module. Third, you can't copy and paste commands into *xleap*. I'm sure by now you see why it can be a painstaking module to use. Why use it at all, then? *xleap* enables you to check what you've done, and make clear modifications. Sometimes your prepi files are failures, but have enough right that you don't want to forcibly fix them. You can redraw bonds, move atoms, and more in the editor. In the words of Alice, "only *LEaP* knows what *LEaP* wants." If it's right in *LEaP*, then it'll be right in your AMBER simulations, even if your solvated PDB looks incorrect in VMD (obviously recheck it after minimization–it should look correct in VMD by then).
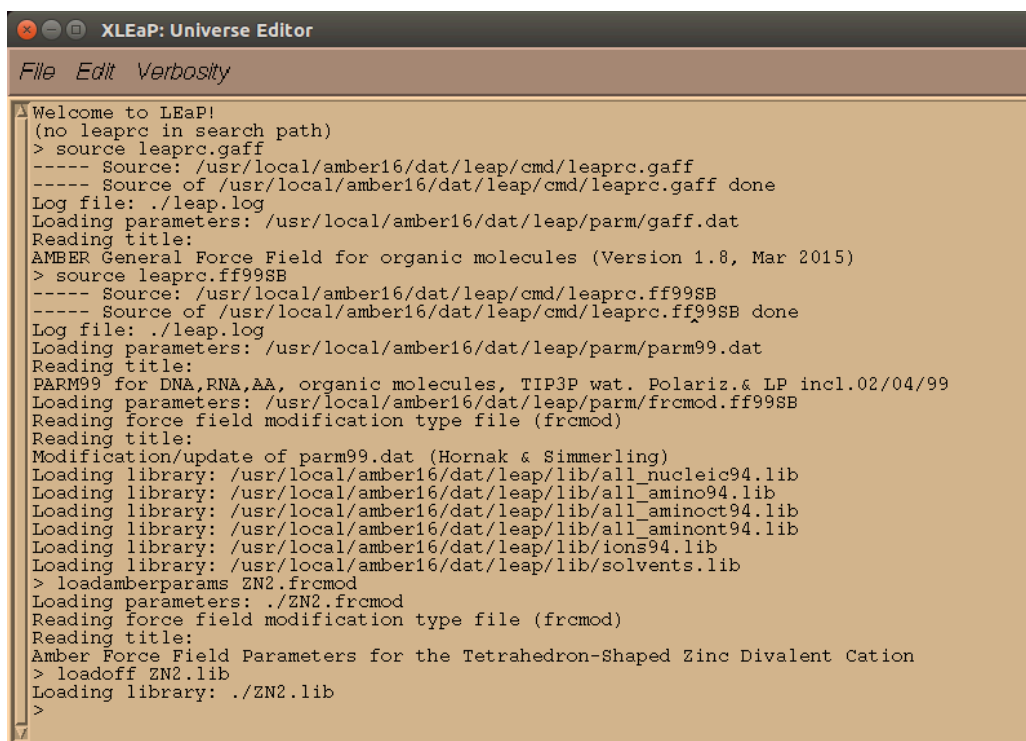
xleap uses the same command syntax that was seen with *tleap* (see the table (page 91)). To open the program, open a Terminal, change directories to the folder with the files you need to set-up the system (so PDBs, prepi files, frcmod files, mol2 files, and lib files), and enter

```
$ AMBERHOME/bin/xleap
```

into the command line.

> ❶ **Note:** If you're doing this remotely (i.e. through an ssh connection) you'll need to make sure X11 forwarding is enabled (meaning you used `ssh -X` or `ssh -Y` when you began the connection). After doing this, the GUI will pop up. The Terminal is now useless until you close *xleap* through (a) typing `quit` in its command line, (b) clicking the orange X (not recommended, it's not the safe way to quit), or (c) following `File → Quit`.

Once the GUI appears (see below), you can start entering the commands to set up your system.

```
❌ ⊖ ⊡  XLEaP: Universe Editor

File  Edit  Verbosity

Welcome to LEaP!
(no leaprc in search path)
> source leaprc.gaff
----- Source: /usr/local/amber16/dat/leap/cmd/leaprc.gaff
----- Source of /usr/local/amber16/dat/leap/cmd/leaprc.gaff done
Log file: ./leap.log
Loading parameters: /usr/local/amber16/dat/leap/parm/gaff.dat
Reading title:
AMBER General Force Field for organic molecules (Version 1.8, Mar 2015)
> source leaprc.ff99SB
----- Source: /usr/local/amber16/dat/leap/cmd/leaprc.ff99SB
----- Source of /usr/local/amber16/dat/leap/cmd/leaprc.ff99SB done
Log file: ./leap.log
Loading parameters: /usr/local/amber16/dat/leap/parm/parm99.dat
Reading title:
PARM99 for DNA,RNA,AA, organic molecules, TIP3P wat. Polariz.& LP incl.02/04/99
Loading parameters: /usr/local/amber16/dat/leap/parm/frcmod.ff99SB
Reading force field modification type file (frcmod)
Reading title:
Modification/update of parm99.dat (Hornak & Simmerling)
Loading library: /usr/local/amber16/dat/leap/lib/all_nucleic94.lib
Loading library: /usr/local/amber16/dat/leap/lib/all_amino94.lib
Loading library: /usr/local/amber16/dat/leap/lib/all_aminoct94.lib
Loading library: /usr/local/amber16/dat/leap/lib/all_aminont94.lib
Loading library: /usr/local/amber16/dat/leap/lib/ions94.lib
Loading library: /usr/local/amber16/dat/leap/lib/solvents.lib
> loadamberparams ZN2.frcmod
Loading parameters: ./ZN2.frcmod
Reading force field modification type file (frcmod)
Reading title:
Amber Force Field Parameters for the Tetrahedron-Shaped Zinc Divalent Cation
> loadoff ZN2.lib
Loading library: ./ZN2.lib
>
```

The xleap Universe Editor GUI shown after sourcing force fields and adding zinc parameters.

Individual components can be edited and checked in *xleap* after all the necessary things have been added. For a loaded PDB, individual residues (and their connections) can be edited using the name it was loaded in with (ex. `select PDB.123` would pick residue 123). These different components can then be edited with something like `edit PDB`, which will open the Unit editor. Remember, the editor will not work properly if numlock is enabled, so make sure that that is toggled off. Following `Edit → Show selection only` will show the residues you highlighted. The highlighting can be turned off by typing something like `deselect PDB` in the command line Universe Editor.

To zoom inside the Unit editor, hit the `Ctrl` key while simultaneously right-clicking and dragging. Pressing the `Ctrl` key and moving the mouse will reorient the system. You can translate the system with just right-clicking, like you would in VMD. The atom names and types can be displayed by using the options under the `Display` heading. Highlighting an atom in the editor box will select it (provided `Select` is marked under `Manipulation`) and choosing the other effects under `Manipulation` can adjust it. A system that needs editing in the Unit editor is shown below.
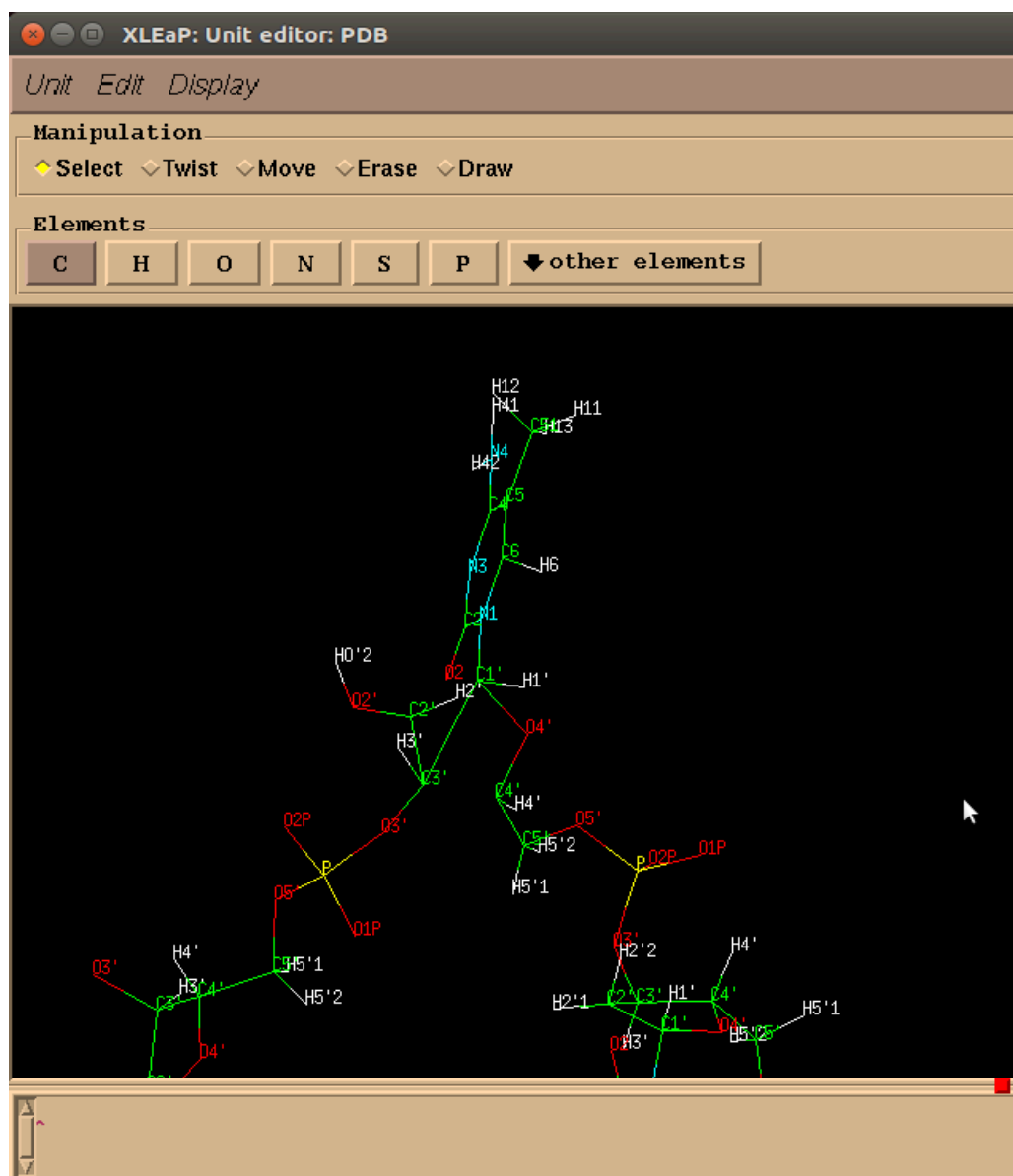
The xleap Unit Editor, where a ring that needs to be fixed is shown. The C3' to C1' bond should be between C3' and C4'.

Every system is different, and each has its own little quirks. There are general trends, though, and things that should be saved along the way. An example of system set up with *xleap* is:

```
> source leaprc.gaff
> source leaprc.protein.ff14SB
> loadoff atomic_ions.lib
> addAtomTypes { { "DZ" "Zn" "sp3" } { "Zn" "Zn" "sp3" } }
> loadamberparams ZN2.frcmod
> loadoff ZN2.lib
> check ZN2
> loadoff akg.lib
> check AKG
> loadamberprep NonStandardA.prepi
> loadamberparams NonStandardA.frcmod
> check NSA
> loadamberprep NonStandardB.prepi
> loadamberparams NonStandardB.frcmod
> check NSB
> list
> PDB = loadpdb wildtype-system-AB.pdb
> check PDB
> savepdb PDB wildtype-system-AB-vac.pdb
> saveamberparm PDB wildtype-system-AB-vac.prmtop wildtype-syst
em-AB-vac.inpcrd
> solvatebox PDB TIP3PBOX 12.00
> addions PDB K+ 0
> savepdb PDB wildtype-system-AB-wat.pdb
> saveamberparm PDB wildtype-system-AB-wat.prmtop wildtype-syst
em-AB-wat.inpcrd
> quit
```

Library Errors

AMBER…. breaks. Frequently. Some things are so common that you're mildly frustrated you have to deal with it AGAIN. Several of these relatively common problems will be addressed here. Other things are new and only really answered after consulting (or even emailing!) the AMBER Mailing List .

One of the most common problems is that for some reason, the script that explains AMBER, wasn't accessed before trying to use AMBER. The error message looks something like:

```
cpptraj: error while loading shared libraries: libsander.so: ca
nnot open shared object file: No such file or directory
```

So, if you're using a bash shell, here's what you do to fix it.

```
$ source $AMBERHOME/amber.sh
```

If you're using a c-shell environment, use:

```
$ source $AMBERHOME/amber.csh
```

If you're not sure which shell you're using, first use the `$ echo $SHELL` command.

No $AMBERHOME

We talked in the UNIX manual (page 0) about these silly variable types that start with `$` , like `$AMBERHOME` . But what happens when there isn't actually an `$AMBERHOME` ?

```
$ $AMBERHOME/bin/cpptraj
-bash: /bin/cpptraj: No such file or directory
```

Check your `~/.bashrc` file.

The following two lines need to appear in your `~/.bashrc` file (with the correct version of AMBER referenced (16, 18, etc.), and sourcing `amber.csh` if you're using a c-shell environment).

```
export AMBERHOME="/usr/local/amber18"
source "$AMBERHOME/amber.sh"
```

If they're not there, then you need to add them with either vi (page 0) or gedit. After adding them, make sure you use `$ source ~/.bashrc` to actually make them available.

Atoms Not in Residue Templates

Occasionally while using a LEaP-based system, you'll get errors like:

```
Created a new UNIT for residue: WAT sequence: 585
Created a new atom named: O within residue .R<WAT 585>
Created a new atom named: H1 within residue .R<WAT 585>
Created a new atom named: H2 within residue .R<WAT 585>
   total atoms in file 7658
   The file contained 7658 atoms not in residue templates
```

These occur when you didn't source the correct leap files with the atom types you needed (leaprc.ff99SB, for example).

You can also get these errors if you have non-standard residues in your files, but you didn't get the parameters you needed for them.

```
Created a new UNIT for residue: AKG sequence: 455
Created a new atom named: O1 within residue .R<AKG 455>
Created a new atom named: C1 within residue .R<AKG 455>
Created a new atom named: O2 within residue .R<AKG 455>
Created a new atom named: C1 within residue .R<AKG 455>
Created a new atom named: O3 within residue .R<AKG 455>
---
Created a new atom named: C4 within residue .R<AKG 455>
   total atoms in file 7658
   The file contained 69 atoms not in residue templates
```

Go back and load in the things you need–check the force field and check your non-standards. Effectively, begin the LEaP process all over again.

Corrupt *cpptraj* Warning

Have you ever been trying to analyze a simulation that you KNOW worked, but for some reason the analysis script that worked on every other replicate isn't working for one of them? And then you go to look at it and there are frames that are missing? No? Just me? Fine. If you did, though, your *cpptraj* log (or error file or wherever you get the *cpptraj* information printed to) may have this message:

```
Warning: Frame XXXX coords 1 & 2 overlap at origin; may be corr
upt.
```

This can mean a few things, including but not limited to, a corrupted simulation (the horror). If you had rewritten the trajectory, though, it could just mean that the rewrite was corrupted. Corruption can happen for all sorts of reasons, like input/output errors, problems with the computer just because it's a computer, random losses of power… you get the idea.

Here's two things that *may* fix the problem.

1. Try rewriting the trajectory again.

2. Rerun *cpptraj* with the `check skipbadframes` option. (You may be shocked that this skips frames that aren't good.)