A simple guide to Gromacs 5

Michael P. Howard, Zifeng Li, and Scott T. Milner

September 27, 2017

Department of Chemical Engineering
The Pennsylvania State University, University Park, PA 16802
Email: mph5128@psu.edu

Contents

1 Getting started

This tutorial is written to give the new user the tools necessary to start running simulations as quickly as possible using GROMACS (GROningen MAchine for Chemical Simulations). Gromacs changed considerably between versions 4 and 5; this tutorial is written for version 5. It makes no attempt to enumerate all features of the package, which are well-documented in the user manual. Throughout this guide, text written in `this font` are commands and keywords written verbatim.

1.1 Installation

All necessary tools are already installed on the Penn State computing clusters (`https://ics.psu.edu/advanced-cyberinfrastructure/`). You can log into one of the clusters through a UNIX terminal, such as Terminal, iTerm (downloadable), X11/XQuartz, or Exceed OnDemand.

```
ssh -X your-username@hostname
```

There are two families of clusters in use at the time of this writing. The old clusters, soon to be retired, are the Lion-X series and Hammer; the new clusters are ACI-b and ACI-i.

The Lion-X and ACI-b clusters are for running "batch" jobs (jobs which are submitted and run as computing time becomes available). Hammer and ACI-i are for running interactive jobs, and for testing and debugging. More information is provided on running batch jobs in a later section. The Milner research group has multiple queues on ACI-b, discussed

Cluster	`hostname`	Types of jobs
Hammer	`hammer.rcc.psu.edu`	Interactive
Lion-XG	`lionxg.rcc.psu.edu`	Batch (lionxg-milner)
Lion-XV	`lionxv.rcc.psu.edu`	Batch (lionxv-milner)
ACI-i	`aci-i.aci.ics.psu.edu`	Interactive
ACI-b	`aci-b.aci.ics.psu.edu`	Batch

below.

Gromacs 4 is installed by ICS for all users on Lion-X, Gromacs 5 on ACI. The Milner group has installed Gromacs 5 locally on Lion-X, and Gromacs 4 on ACI, to aid in transitioning existing projects from the old clusters to the new. This tutorial, however, assumes you are running Gromacs 5 on ACI.

Gromacs comes in several "flavors". First, calculations can be done either in single precision (recommended for most purposes) or double precision (only necessary for precise energy calculations). Second, calculations can be speeded up by using multiple processors, which either reside exclusively on a single node in the cluster (using "thread-MPI"), or are distributed across multiple nodes (using "MPI"). The difference between thread-MPI and MPI is discussed further below; in short, thread-MPI is for small tasks, and MPI for big simulations with many atoms.

All the many programs within Gromacs are accessed from just one executable, generically called `gmx`. On ACI, `gmx` is present in two versions: `gmx_d`, which is double precision and uses thread-MPI, and `gmx_mpi`, which is single precision and uses MPI.

Logging into ACI

The ACI cluster has two "faces": ACI-i (interactive) and ACI-b (batch). ACI-i is for setting up jobs, debugging, editing scripts, analyzing data interactively, and generally for using any program that is "visual", i.e., opens windows to display things. ACI-b is for submitting and running batch jobs — big calculations that use multiple compute cores and take a long time to run.

ACI-i and ACI-b share a common filesystem, but not all the same software is available on both. In particular, the easy-to-use windows-based editor `gedit` is only on ACI-i, and you can only submit batch jobs from ACI-b. Also, other visual programs such as Mathematica and vmd do not work reliably on ACI-b (sometimes the expected window fails to open, with an error like `Can't open display:`).

So the recommended procedure is: if you will need any visual program, log onto ACI-i using Exceed OnDemand (`https://downloads.its.psu.edu`) OS X users, need XQuartz 2.7.8 installed (`https://www.xquartz.org`) to use Exceed OnDemand.

To login with Exceed, after launching the program, fill in the machine address (aci-i.aci.ics.psu.edu), your username and password, then click "Login";. You will be prompted for a 2FA (two-factor authentication) option. Recommended is "Duo Push", which must be installed on your smartphone. To set up 2FA, see
`https://ics.psu.edu/advanced-cyberinfrastructure/accounts/`.

Then choose "Xconfig: Default Xconfig" and "Xstart: Terminal.xs" and click "Run", which will (finally) give you a terminal window, logged into ACI-i. You can launch other simultaneous ACI-i sessions as "tabs" on the terminal window, by selecting "File / Open Tab". To connect to ACI-b, from such a session, type `ssh <username>@aci-b.aci.ics.psu.edu`; you will be prompted for your password and 2FA. Now you're logged onto ACI-b, and can submit batch jobs (see below).

Getting access to Gromacs

Gromacs is installed on ACI, but not available to you by default. To get access to it, you need to load it with the following commands:

```
module load gcc/5.3.1
module load openmpi/1.10.1
module load gromacs/5.1.4
```

Each time you log in, these commands must be executed for you to use Gromacs. The easy way to do that is to add these lines to your `.bashrc` file, located in your home directory. `.bashrc` is a script that executes each time you log in. (After you first make the change, run `source .bashrc`, which executes the script.)

As of this writing, these commands give error messages, but apparently still work. You can verify that you have access to Gromacs using the `which` command, as in

```
[stm9@aci-lgn-006 ~]$ which gmx_d
/opt/aci/sw/gromacs/5.1.4_gcc-5.3.1_openmpi-1.10.1/bin/gmx_d
```

and similarly for `gmx_mpi` and `mpirun`. (`which` searches for executables on your "search path".) You can verify that the commands are working, with `gmx_d --version` (which prints information about the version installed) or `gmx_d -h` (which prints help on how to use the command).

Making local Gromacs library

We will now create a local copy of the Gromacs "topology" folder `top` in our work directory. This will allow us to make an important modification later — adding new monomers to the predefined list of predefined amino acids, which helps us build topology files for synthetic polymers.

```
cp -r /opt/aci/sw/gromacs/5.1.4_gcc-5.3.1_openmpi-1.10.1/share/gromacs/top ~/work/top
```

Then, add the following line to `.bashrc`

```
export GMXLIB=~/work/top
```

and `source .bashrc` again. Finally, enter `echo $GMXLIB`, and check that the output matches the location of your new local copy of `top`.

1.2 Other useful modules

A complete list of software on ACI is available at
`https://ics.psu.edu/advanced-cyberinfrastructure/support/software/`.
`module spider` lists all modules available to load; `module spider <package>` searches for a specific software package. For some packages, `module spider` specifies other packages that must be loaded first. This can be recursive: `gromacs/5.1.4` requires `gcc/5.3.1` and `openmpi/1.10.1` to be loaded first. When you use `module spider` to find out how to load those, you find that `openmpi/1.10.1` requires `gcc/5.3.1` or `pgi/16.4` to be loaded first. In short, modules can be loaded with one or more `module load <package>` commands, which can be added to `~/.bashrc` to load every time you log in.

1. *Python.* A high order scripting language, useful for processing and analyzing large text files. Multiple versions are available, so it is recommended to choose which one you will support. Recommended are either version 2.6.5 or version 3.1.1 (the newest as of the writing of this tutorial).
   `module load python/<version>`

2. *Mathematica.* Interactive or batch Mathematica, useful for calculations which would take a long time to run on your personal computer. Mathematica is a "windows application", i.e., it opens windows that display things. Such packages run reliably only on ACI-i, and only under Exceed OnDemand (see above).
   `module load mathematica/<version>`

3. *VMD: Visual Molecular Dynamics.* Useful for visualizing configurations and simulation results. This package is also a windows application. As of this writing, `vmd` is *not*

installed on ACI, except as a local install in Milner group storage. To use it, add these lines to your `.bashrc` file (the first line is a comment):

```
# add path to local vmd
export PATH=$PATH:/gpfs/group/stm9/default/SOFTWARES/vmd-1.9.2/build/bin
```

1.3 Run a simple simulation

You will need to upload a copy of `decane_min` to the cluster that you are logged into. One way to do this is from the UNIX terminal using `scp` (Secure CoPy), Examples of using `scp` can be found at `https://www.tecmint.com/scp-commands-examples/`. A more convenient way to upload files is to use the software **Cyberduck** (`http://cyberduck.ch`). If you use a Mac, you should also download **TextWrangler** (available for free in the Mac App Store), a easy-to-use and powerful text editor designed for programmers. Cyberduck can be used together with TextWrangler to edit cluster files, by automatically downloading a copy to edit on your laptop/desktop, then automatically uploading it again when you save. After you have installed both, open Cyberduck, and go to Preferences > Editor. Here you can choose TextWrangler as your default program for opening files (recommended).

Then, click Open Connection to open a connection to the cluster. Choose `SFTP` from the drop down menu. Enter the `hostname` under Server, and your PSU username under Username (you do not need to enter your password now). Click Connect. Enter your password when prompted. Upon successful connection, click the Action menu, and choose New Bookmark from the bottom of the menu. This will save this connection for easy access next time. Change the Nickname to ACI-i (or whatever cluster you logged into). Next time you open Cyberduck, you can double click on this bookmark to log in.

To upload files, simply drag them from your local system onto the Cyberduck window. To download files, double click on them in the Cyberduck window, or drag them to your local system.

Decane minimization

Copy `decane_min` from the tutorial folder to your `work` folder. We will run a simple energy minimization of a single molecule of decane. There are four files in this folder: a configuration (`decane.pdb`), a topology (`decane.top`), a set of run parameters for minimization (`em.mdp`), and a script to run the simulation (`min`). We will skip using the script for now.

Open `decane.pdb`. This is a configuration file for decane made of "ethylene" monomers. It tells Gromacs where all the atoms are initially, and labels each of them.

Open `em.mdp`. The first section tells Gromacs to run an energy minimization with the `steep` algorithm. The second section tells it how much to minimize. The rest of the file defines parameters for the potentials.

We need to "preprocess" the simulation parameters for Gromacs. We do this using the command `grompp` (GROMacs PreProcessor). Enter the following command:

```
gmx_d grompp -f em.mdp -c decane.pdb -p decane.top -o em.tpr
```

You should see Gromacs output telling you that you have successfully preprocessed the simulation. Now, we can feed `em.tpr` to `mdrun`, which will actually run the energy minimization.

```
gmx_d mdrun -s em.tpr -deffnm decane_min -c decane_min.pdb
```

The minimization should take a second or two to run. To test the output, you can use `vmd` to view the final structure:

```
vmd decane_min.pdb
```

It is convenient to combine multiple Gromacs commands into a single script file so that you do not need to retype long commands each time. The script `min` is provided to you as an example of this. To execute it, enter `./min`

2 Anatomy of a simulation

Now that we have run a simulation, let's look at the different parts of the simulation, which are common to almost all Gromacs simulations. In general, a simulation consists of:

1. Define the **atoms** and monomers (**residues**) in your molecules. Atom types are taken from a **force field**.
2. Build polymers from these residues. Generate a starting **trajectory** of molecules, where you list all of the atom **positions** (and **velocities**). Generate a **topology** for the molecules, which lists all atoms, **bonds**, **angles**, and **dihedrals**.
3. Set simulation **parameters**. Set the **simulation box** size. Pre-process the simulation parameters, starting trajectory, and topology.
4. **Run** the simulation, and output trajectories and other useful statistics.
5. Analyze the data using Gromacs or your own tools in Mathematica or Python.

In our simple example, we were given a configuration (Step 1) and topology (Step 2). The run parameters were set in Step 3 in `em.mdp`, and the simulation box size was also given in the configuration file. `grompp` does the preprocessing, and `mdrun` runs the simulation (Step 4). We checked our results using VMD (Step 5).

2.1 Force field

In MD simulation, Newton's equations of motion are solved on many atoms. The forces on each atom are computed according to a force field. The force field can be decomposed into two types of interactions: bonded and non-bonded. Bonded interactions are the forces that arise due to bonds within a single molecule. For example, a bond between two atoms will have a preferred bond length b_{ij}. The potential around this length can be approximated as harmonic for many applications, that is

$$U = \frac{1}{2} k_{ij}^b (r_{ij} - b_{ij})^2 \tag{1}$$

where r_{ij} is the actual length between the atoms, and b_{ij} is the preferred bond length. The force can then be computed from $F = -\partial U / \partial r$. The main bonded interactions are: bonds, angles, and dihedrals.

The non-bonded interactions include things like dispersion forces and electrostatic interactions. These forces are both intramolecular (within the molecule) and intermolecular (between molecules). An "exclusion" is usually applied to the intramolecular dispersion forces for those atoms that are connected within a dihedral (usually called "1-4 exclusions"). The dispersion forces use the Lennard-Jones potential

$$U = 4\epsilon_{ij} \left(\frac{\sigma_{ij}^{12}}{r_{ij}^{12}} - \frac{\sigma_{ij}^{6}}{r_{ij}^{6}} \right) \tag{2}$$

and the electrostatic potential is given by the Coulomb potential

$$U = \frac{1}{4\pi\epsilon_0}\frac{q_i q_j}{\epsilon r_{ij}} \tag{3}$$

Different force fields have different parameters for these potentials. The parameters are applied by using the labels for each atom type given in the force field files. A common force field to use is the OPLS-AA (Optimized Parameters for Liquid Simulation-All Atom). The various force fields and parameters may be found in the `top` folder that you copied.

2.2 Simulation units

Gromacs employs the following fundamental and derived units:

quantity	unit
length	nm
mass	amu
time	ps
temperature	K
charge	e (fundamental charge $\approx 1.6 \times 10^{-19}$ C)
energy	kJ/mol
force	kJ/mol-nm
pressure	kJ/mol-nm^3
velocity	nm/ps

More details about units and nomenclature can be found in the Chapter 2.2 of the Gromacs manual.

2.3 Types of files

You may have noticed that many files were generated when you ran the decane minimization. Here are a few important Gromacs files that you may encounter (note: not all of these were generated during the minimization).

1. `.gro` GROmacs configuration file. Positions are given in nm, and velocities in nm/ps.
2. `.pdb` Protein Data Bank configuration file. Used by biochemists. *Fixed column positions*. Useful for `pdb2gmx` (see below).
3. `.rtp` Residue ToPology File. Originally for amino acids and DNA building blocks; we use this for building polymer chains. Contains information about atom types and connectivity in a residue or monomer.
4. `.top` TOPology file. Tells how atoms in a molecule or system are connected. This file is complicated to build "by hand" for all but small molecules; special tools exist to help. See below.

5. `.ndx` iNDeX file. Lists atoms that are in certain named groups. Useful for controlling behavior of groups (e.g., freezing their positions), or getting data about a subset of the system. Format is `[ group ]` followed by rows of atom numbers in that group.

6. `.mdp` MD Parameter file. Tells Gromacs what it needs to know about the simulation during pre-processing.

7. `.tpr` ToPology Run file. Binary. Output of simulation pre-processing, input to `mdrun` to run the simulation.

8. `.trr` TRajectory Run file. Binary. Output of an MD simulation, includes "snapshots" of the system at different times, which could be turned into a "movie".

9. `.log` LOG file. Output by `mdrun`. Can be used to diagnose problems.

10. `.edr` Energy Data Run file. Binary. Contains run data such as energy, pressure, etc. Can be analyzed with Gromacs tools.

2.4 Configuration file

Configurations can be downloaded (`http://www.rcsb.org/pdb/home/home.do`), generated by you (in Mathematica, for example), or drawn in software like **Avogadro**. Avogadro (`http://avogadro.cc`) is useful for drawing and visualizing molecules on your desktop / laptop.

.gro

First line is a descriptor, second line is number of atoms, each following line is an atom. These columns correspond to: residue number (5), residue name (5), atom name (5), atom number (5), position x, y, z (8, 3 decimals, in nm), velocity x, y, z (8, 4 decimals in nm/ps). The corresponding C format code is

```
%5d%-5s%5s%5d%8.3f%8.3f%8.3f%8.4f%8.4f%8.4f
```

The last line of the file gives the simulation box lengths (in nm).

.pdb

Details can be found online (`http://deposit.rcsb.org/adit/docs/pdb_atom_format.html`). Necessary lines are CRYST1 (which sets the simulation box size), MODEL (starts main part of file), ATOM (which has the necessary atom information), TER (which ends each chain), ENDMDL (which ends the model), and END (which ends the file).

2.5 Parameter file

The `.mdp` file tells Gromacs the details of the simulation that you will run. Every file needs a section of basic run parameters, which choose an "integrator", set the number of simulation steps, the step size, the use of constraints, etc.

An important part of the parameter file is the specification of a "cutoff scheme", i.e., how the long-range nonbonded Lennard-Jones (van der Waals) and electrostatic (Coulomb) interactions will be cut off and neglected beyond some distance.

For choosing van der Waals cutoffs, bear in mind that the typical values of σ range up to 3.5–4Å, with ϵ values smaller than 1.0 kJ/mol (=120K), so that at a separation of of 2.5σ, the force derived from the potential $4\epsilon(r/\sigma/)^6$ is down by a factor of $2.5^7 = 0.0016$. This suggests a cutoff length of 2.5×4Å or about 10Å. Of course, there are still residual contributions to the attractive energy and pressure, but these can be computed with tail corrections.

New cutoff scheme.

Since Gromacs 4.6, a simpler and more efficient Verlet cutoff scheme has been available. (See `http://manual.gromacs.org/online/mdp_opt.html#nl`). This scheme works with PME (particle mesh Ewald) electrostatics, which does not require zero-sum charge groups to be defined (charge groups were a holdover from cutoff electrostatics). GPU acceleration in Gromacs requires the Verlet scheme, and for CPUs with special instruction sets such as SSE and AVX.

The Verlet scheme has an explicit, exact cut-off at `rvdw=rcoulomb`, at which both Lennard-Jones and Coulomb interactions are shifted to zero by default. Currently only cut-off, reaction-field, PME electrostatics and plain LJ are supported. Some `mdrun` functionality is not yet supported with the Verlet scheme, but `grompp` checks for this. With GPU-accelerated PME or with separate PME ranks, `mdrun` automatically tunes the CPU/GPU load balance by scaling `rcoulomb` and the grid spacing.

Here is a set of sample parameters.

```
; cutoff-related parameters
cutoff-scheme  = verlet
; rebuild neighbor list every 20 steps
nstlist  = 20
; build neighbor list from cell list (default)
ns_type = grid
; periodic boundary conditions in x,y,z
pbc = xyz
; Electrostatics
; "particle mesh Ewald" -- efficient Coulomb interactions
coulombtype = PME
; crossover distance from "real space" to "Fourier space"
rcoulomb = 1.2
; van der Waals
; make potential zero at cutoff
```

```
vdw-modifier = Potential-shift-Verlet
```

Old cutoff scheme.

Prior to Gromacs 4.6, the cutoff scheme was more complicated. For van der Waals interactions, the choices were `cutoff` (abrupt cutoff — *not* recommended), `shift` (shift potential to zero beyond cutoff), and `switch`. For `cutoff` and `shift`, the cutoff length is `rvdw`. For `switch`, the interaction is turned off smoothly from `rvdw_switch` to `rvdw`.

For Coulomb interactions, cutoff schemes `cutoff`, `shift`, and `switch` can also be specified; in addition, there is `PME` (particle mesh Ewald). With `PME`, the interpretation of `rcoulomb` is not a cutoff per se, but the crossover between treatment of Coulomb interactions in real space and Fourier space.

The length `rlist` is the range of the neighbor lists maintained for each atom; neighbor lists are updated every `nstlist` steps (typically 10fs or so, hence 5-10 steps). If particle mesh Ewald (PME) electrostatics are used, we must take `rcoulomb` equal to `rlist`.

If we use either `switch` or `shift` for the van der Waals interactions, Gromacs requires us to have `rvdw` less than `rlist`, by an length equal to the size of the largest charge groups.

So if we use PME, hence `rcoulomb` equal to `rlist`, and `cutoff` van der Waals, then we can take `rvdw` larger than `rcoulomb`. In this case, forces from particles beyond `rlist` up to `rvdw` are only updated every `nstlist` steps. This is a "twin-range" cutoff, and can be applied to any simple cutoff interaction.

If instead we use PME and `switch` or `shift` van der Waals, Gromacs requires us to have `rvdw_switch` ¡ `rvdw` ¡ `rcoulomb` = `rlist`, with a width between `rvdw` and `rlist` of the size of the largest charge groups.

This is a complicated set of conditions to satisfy. A set of sample choices are:

```
vdw-type = switch
rvdw_switch = 0.8
rvdw = 1.0
coulombtype = pme
rcoulomb = 1.3
rlist = 1.3
```

For very small systems, with linear dimension 20Åor so, it is hard to wedge in all these lengths, so the simplest cutoff scheme is something like:

```
vdw-type = cutoff
coulombtype = pme
rlist = 1.0
rvdw = 1.0
rcoulomb = 1.0
```

which is not very accurate but at least can run.

2.6 Topology file

The topology file contains all the information about the types of atoms present in a simulation, how they are bonded together, what intramolecular bonded interactions are present and what their potentials are, what are the interaction potentials between nonbonded atoms.

Topology files are typically built and best understood "from the top down". The .top file for the entire simulation can be quite short, as it can consist entirely of references to .itp (Input ToPology) files for the molecules and forcefield used in the simulation

A typical example is (see Gromacs manual 5.7.2)

```
#include oplsaa.ff/forcefield.itp
#include molecule1.itp
#include molecule2.itp

[system]
My System

[molecules]
; name      count
molecule1  1000
molecule2  2000
```

The `#include` declaration reads the contents of the named file at that point in the .top file. Here `oplsaa.ff/forcefield.itp` is the topmost .itp file of the forcefield to be used (OPLS-AA); `molecule1.itp` and `molecule2.itp` are the .itp files of the two molecules present in the simulation. The remaining declarations are simply the name of the system, and the number of each type of molecules.

Molecule .itp file

Typically, each kind of molecule has its own .itp file (this keeps things neat). These can be created by hand (tedious for larger molecules), generated with web-based tools (but should be checked manually for reasonableness), or obtained using `pdb2gmx` (for molecules represented as chains of "residues", see below).

The typical structure of a molecule .itp file is (see Gromacs manual 5.7.2):

```
[moleculetype]
; name  nrexcl
...

[atoms]
; #  type  res#  residue  atom  cg#  charge mass
...

[bonds]
; i  j  func  b0 kb
...
```

```
[angles]
; i  j  k  func  th0  cth
...

[dihedrals]
...

[position_restraints]
; i func fc
...
```

The [moleculetype] declaration gives the molecule name, and the "number of exclusions" — how many bonded neighbors away are excluded from non-bonded interactions (typically 3 if dihedral potentials are present), so that bonded interactions are not "interfered with" by non-bonded contributions.

Next comes [atoms], which lists all the atoms present in the molecule, with their number, "atom type", residue number and name, atom name, charge group number, and charge. If the charge is not present, the default value from the potential is used. The "atom type" should match one of the types in the `atomtypes.itp` file of the potential you are using, unless you are defining custom atom types. This file gives "hints" as to the structural meaning of the different atom types the potential contains.

Next comes [bonds], which lists all the pairs of bonded atoms (by number), the bonded function type (1 is harmonic spring), rest length `b0` and spring constant `kb`. Then [angles], similarly listing all the sets of three successive bonded atoms by number on which angular potentials are acting, with the function type (1 is harmonic spring), minimum energy angle `th0` and spring constant `cth`. Then comes [dihedrals], which lists all sets of four successive bonded atoms, in similar fashion. There are two types of dihedral functions, each with its own parameters (see Gromacs manual for details).

Note: bond angles are defined as between the first and last atom through the middle atom. The preferred C-C-C angle is around 112°. Dihedral angles are defined as the angle that the first and last atom form over the bond. In butane, the C-C-C-C dihedral angle is 180° in the trans configuration.

If only atom numbers are given for entries in [bonds], [angles], or [dihedrals], the corresponding potential defaults to values specified in the potential (in the file `ffbonded.itp`). This means building a molecule .itp file by hand is manageable if tedious (the number of entries in the [angle] and [dihedral] table get quite large).

The nonbonded (Lennard-Jones) interaction parameters for all the atoms in a molecule are set by default from the atomtype, by referring to the file `ffnonbonded.itp` for the chosen potential (e.g., OPLS-AA).

Force field .itp file

The file `forcefield.itp` for a given potential is likewise defined "top down". The file itself is very short, containing only a single definition and some `#include` statements:

```
[ defaults ]
; nbfunc comb-rule gen-pairs fudgeLJ fudgeQQ
1 3 yes 0.5 0.5

#include "ffnonbonded.itp"
#include "ffbonded.itp"
#include "gbsa.itp"
```

The [defaults] declaration makes choices about the non-bonded potential function (type 1 = LJ, type 2 = Buckingham), and the "combining rule" for determining nonbonded parameters for different atomtypes interacting. (For details on combining rule, see Gromacs manual 5.3.2; comb-rule 3 is the "sigma-epsilon" system.) The two main `#include` files contain the nonbonded and bonded declarations, previously mentioned above.

In turn, the file `ffnonbonded.itp` (see Gromacs manual 5.2.1) simply contains a list of [atomtypes] declarations:

```
[atomtypes]
; name  bondtype  mass  charge  ptype  epsilon  sigma
...
```

Here "name" is the atomtype name (from `atomtypes.atp`); "bondtype" is a more general name, that declares how the atom's bonded potentials are set by entries in the file `ffbonded.itp`; "ptype" is the particle type (A = atom); the last two parameters are the LJ parameters (their interpretation depending on the default form of the nonbonded potential, set by `nbfunc` in `forcefield.itp`).

Likewise, the file `ffbonded.itp` (Gromacs manual 5.3.3) specifies the parameters for the bonded interactions between atoms of different "bondtypes" (specified in `ffnonbonded.itp`):

```
[bondtypes]
; i  j  func  b0  kb
...

[angletypes]
; i  j  k  func  th0  cth
...

[dihedraltypes]
...
```

2.7 Modifying the potential

For the most part, we let Gromacs set the potential parameters, after we have chosen the atom types that best match the atoms in our molecules (by scanning through `atomtypes.atp` for mentions of corresponding moieties). Occasionally, we want to override the default values for some interaction, supply missing bonded potentials, or create new atom types for new situations, possibly by modifying existing types.

To modify or extend the potential, we insert declarations after the `#include forcefield.itp` declaration in our .top file, or inside the various `molecule.itp` files. First, the charge of specific atoms in a molecule, or any bonded interactions between specific atoms in the molecule, can be overridden by explicit declarations in `molecule.itp`.

To override the nonbonded LJ interactions requires a different approach. We can override the LJ parameters for a given atomtype, with declaration lines like those in `ffnonbonded.itp`, inserted immediately after this file is included.

We can also override the nonbonded interactions *between* specific atomtypes, i.e., overriding the "combination rules" used by default to set those interactions, with declarations like this (see Gromacs manual 5.3.2):

```
[nonbond_params]
; i  j  func   epsilon  sigma
...
```

Sometimes we want to create an entirely new atomtype, to correspond to a moiety not included in the extensive OPLS `atomtypes.atp` file, or to correspond to some fictitious "model atom" type. We can do this by adding [atomtype] declarations like those in `ffnonbonded.itp`, immediately after this file is included. If this declaration uses an existing bondtype, then bonded interactions involving the new atomtype will be set by the declarations in `ffbonded.itp`.

If instead the new atomtype is given a new bondtype, declarations like those in `ffbonded.itp` can be added immediately after it is included to specify the necessary new bonded interactions. This is tricky, however, because there can be very many combinations of bondtypes to include in specifying angle and dihedral potentials.

Tabulated potentials

Sometimes it is useful to have complete control of the form of the nonbonded interactions, beyond the predefined options available within Gromacs. For this purpose, Gromacs offers the use of tabulated potentials (Gromacs manual 6.7.2). These are accessed from the .mdp file.

The nonbonded potential takes the general form

$$U(r) = q_i q_j/(4\pi\epsilon_0) f(r) + C_6 g(r) + C_{12} h(r) \tag{4}$$

For `nbfunc` equal to 1 (LJ), the default choices for the functions $f(r)$, $g(r)$, and $h(r)$ are $f(r) = 1/r$, $g(r) = -1/r^6$ and $h(r) = 1/r^12$.

The values of C_6 and C_{12} are determined from [atomtype] entries, which may either give C_6 and C_{12} explicitly, or implicitly through values for σ and ϵ, depending on the combination rule chosen in the [defaults] section at the start of the .top file.

With comb-rule = 2 or 3, the nonbonded parameters input are σ and ϵ, and C_6 and C_{12} are computed from $C_6 = 4\epsilon\sigma^6$ and $C_{12} = 4\epsilon\sigma^{12}$. With comb-rule = 1, C_6 and C_{12} are entered directly. (Comb-rule = 2 specifies Berthelot combination rule; 1 or 3 specifies harmonic mean mixing for C_6 and C_{12}.)

In the simplest case, keyword `vdw-type=user` signals the replacement of $g(r)$ and $h(r)$ (and `coulombtype=user` the replacement of $f(r)$) by tabulated potentials for nonbonded interactions between all particles.

These tabulated potentials are provided by the user in a file `table.xvg`, in which each line contains values for $r, f, -f', g, -g', h, -h'$ (separated by tabs or spaces). The r values must be equally spaced (recommended spacing 0.002 nm), up to the cutoff radius plus 1nm.

Sometimes, we want to specify tabulated interactions only for certain types of particles, or perhaps different tabulated interactions for different combinations of particles. This can be accomplished using energy groups: in the .mdp file, include

```
energygryps = xx, yy ...
energygrp_table = xx xx xx yy ...
```

where `xx` and `yy` are different groups (possibly defined using `make_ndx`). With this declaration, Gromacs uses tabulated potentials `table_xx_xx.xvg` for the xx-xx interactions, `table_xx_yy.xvg` for the xx-yy interactions, and so forth for each successive pair of energy groups listed after `energygrp_table`.

Note: with tabulated interactions, the cutoff scheme must be `cutoff-scheme = Group` (the new Verlet cutoff scheme is not yet compatible with tabulated interactions).

See `http://www.gromacs.org/Documentation/How-tos/Tabulated_Potentials`

3 Simulation: lysozyme in water

Here we will simulate a lysozyme solvated in water using Gromacs. This example was developed by Justin Lemkul; his website (`http://www.bevanlab.biochem.vt.edu/Pages/Personal/justin/gmx-tutorials/`) is an excellent resource for Gromacs help. Here we will present the basic steps to run this simulation; the reader is referred to the original tutorial for more details and explanation.

3.1 System generation

In this example, we are given a PDB file for the protein (`1AKI.pdb`), which serves as an initial configuration file. If this were not supplied, you would somehow need to generate a valid starting configuration (see decane example in the next section).

Topology

We must make a topology file for the protein, and also create a processed .gro coordinate file. This is accomplished using `pdb2gmx` ("PDB to Gromacs"), and the OPLS-AA force field. OPLS-AA has a a residue topology file for amino acids, which tells `pdb2gmx` how to build the topology file. We will solvate the system with SPC/E water.

```
gmx_d pdb2gmx -f 1AKI.pdb -o 1AKI_processed.gro
```

When prompted, type `15` to choose the OPLS-AA force field, and then type `7` to choose SPC/E water for the solvent.

Simulation box

Now we use `editconf` to change the system size. Our system will have periodic boundaries, so it needs to be big enough that the protein doesn't "see" itself in the box. To do this, we place the protein in a cubic box with `-bt cubic`, center it with `-c` and pad it with 1.0 nm on each side with `-d 1.0`. That the nearest image of the protein is then 2.0 nm away, which is far enough that it won't interact with itself.

```
gmx_d editconf -f 1AKI_processed.gro -o 1AKI_newbox.gro -c -d 1.0 -bt cubic
```

Solvation

Now we can solvate the protein in water using `solvate`. We tell `solvate` our protein configuration with option `-cp`, and the solvent we want to use with `-cs`, which is `spc216.gro` for SPC, SPC/E, and TIP3P water. We also need to output a new topology file, which we can do **when solvating with water only** using `-p`

```
gmx_d solvate -cp 1AKI_newbox.gro -cs spc216.gro -o 1AKI_solv.gro -p topol.top
```

This overwrites the existing topology file, and backs up the old one by adding `##` around it. Gromacs does this automatically for all files that are overwritten. If you inspect the new `topol.top`, you will see a line has been ended at the end for the new water molecules. You can also see their configurations. added to the end of the new .gro file.

Add anions

Now we use `genion` to add anions to neutralize the system, since lysozyme has a net positive charge (+8). `genion` requires a `.tpr` file as input. So we create one with `grompp` first, using a dummy `.mdp` called `ions.mdp`, which has parameters for a simple energy minimization.

```
gmx_d grompp -f ions.mdp -c 1AKI_solv.gro -p topol.top -o ions.tpr
```

Now we can use `genion` to add ions to the solvent. Options `-pname` and `-nname` are the names for the ions we will use (sodium and chloride). We use `-nn 8` to add 8 anions, since the protein has a net charge of +8 (which `grompp` reported in its output)

```
gmx_d genion -s ions.tpr -o 1AKI_solv_ions.gro -p topol.top -pname NA -nname CL -nn 8
```

Type `13` to choose to declare the ions part of the solvent group.

If you inspect the new .gro and topology files, you will see 8 solvent molecules have been replaced with 8 chloride ions. Our system is now ready for simulation.

3.2 Energy minimization

Before running any MD simulation, we run a brief energy minimization. This ensures that there are no large interatomic forces in the starting configuration, which would cause the system to "blow up" if you started running MD immediately. We run `grompp` to prepare for energy minimization. In the MD parameter file `minim.mdp`, we choose the steepest-descent minimizer (since it is the fastest), and a reasonably large force tolerance of 1000 kJ/mol/nm, enough to ensure there are no serious problems with our starting configuration. Other parameters are described in the Gromacs manual.

```
gmx_d grompp -f minim.mdp -c 1AKI_solv_ions.gro -p topol.top -o em.tpr
```

After processing, we can pass `em.tpr` to the MD engine, and run our minimization. The option `-deffnm` (DEFine File NaMe) sets a default file name for all inputs and outputs, so that we don't have to specify each file individually. Option `-v` (Verbose) causes `mdrun` to print what is happening at each step (not recommended for ordinary use).

```
gmx_d mdrun -v -deffnm em
```

3.3 NVT equilibration

After minimization, the next step is to equilibrate the solvent (water) and solute (protein). We want the system to come to the right temperature and pressure before we begin complete simulation. First, we equilibrate the temperature with an NVT simulation (constant Number of atoms, Volume, and Temperature). In this step, we restrain the heavy atoms in the protein, so that the water can equilibrate around it without disturbing the delicate protein folded structure. Details can be found in `nvt.mdp`.

Again, we `grompp` the run parameters.

```
gmx_d grompp -f nvt.mdp -c em.gro -p topol.top -o nvt.tpr
```

Because of the system size (over 30,000 atoms), it will take a significant amount of computing power to run this NVT simulation, so we submit it to the cluster queue rather than running it interactively. To run faster, we will parallelize the job with MPI; basically, we will distribute the computer's work over four separate compute cores.

The code to submit this job on ACI-b is provided in the "batch script" `nvt.pbs`. To submit the job, log into ACI-b (see above) and enter:

```
qsub nvt.pbs
```

To check the status of your jobs (again, only from ACI-b), enter

```
qstat -u your-username
```

It should take about 12 minutes for your job to run once it has started. Details of this .pbs (Portable Batch System) script will be discussed later.

Once the simulation has finished, we want to check that we have correctly equilibrated the temperature. Use `energy` to extract the variation with time of important system quantities from the .edr file:

```
gmx_d energy -f nvt.edr -o temp.xvg
```

and enter `15 0` to select only the temperature to be extracted.

The resulting file `temp.xvg` can be opened using Grace (from ACI-i, type `xmgrace temp.xvg`) or using the enclosed Mathematica notebook (`xvg_reader.nb`). Plot the temperature versus time, and you should see it stabilize around 300 K (Figure 1). There will be temperature fluctuations, but that is perfectly normal.

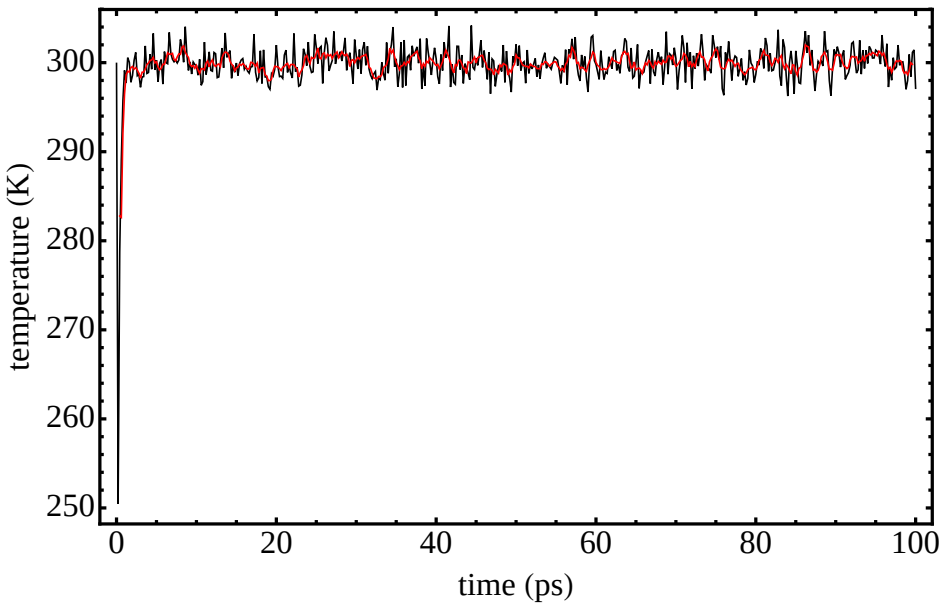


Figure 1: Temperature versus time for NVT simulation. Black shows actual temperature, red shows moving average over 5 time steps.

3.4 NPT equilibration

Now that the temperature is equilibrated, we want to equilibrate the system pressure and density using NPT simulation (constant Number of atoms, Pressure, and Temperature). This allows the simulation box size to adjust. During this NPT equilibration, we again restrain the protein position. We `grompp` for the NPT, but this time we use `-t` to use the "checkpoint" (`.cpt`) file from the NVT equilibration as the initial configuration. Checkpoint files allow simulations to be restarted using the exact positions and velocities they ended with. This is also good practice when transferring trajectories to a simulation with different conditions (as from NVT to NPT); .gro files have limited numerical precision, which can cause problems. A "continuation" parameter is required in the `.mdp` file, which prevents Gromacs from randomly setting the initial velocities.

```
gmx_d grompp -f npt.mdp -c nvt.gro -t nvt.cpt -p topol.top -o npt.tpr
```

We can then run the NPT simulation as a batch job on ACI-b, in the same way as for the NVT simulation. This job will also take about 12 minutes once it has started to run.

Once the simulation has finished, we use `energy` to check the pressure and density fluctuations:

```
gmx_d energy -f npt.edr -o pres.xvg
```

and enter `16 0` to extract only the pressure versus time. Then, extract the density versus time into a separate file, with

```
gmx_d energy -f npt.edr -o density.xvg
```

Enter `22 0` to get the density. Plot both of these files using either Grace or `xvg_reader.nb`. You should find that although the pressure is fluctuating wildly, it has stabilized on average around 1.0 bar (Figure 2). The density should also have stabilized close to the density of water (since the system is mainly water, and SPC/E is a good model for liquid water). Note that the density is fairly stable (Figure 3), so we are now equilibrated with respect to temperature, pressure, and density.

3.5 Production MD

For "production" MD, we remove all constraints on the protein, to see how the protein moves under thermal agitation, and whether it retains its folded shape. We will run the simulation for 1.0 ns, which should be sufficient to collect a sample set of data. We `grompp` our new set of parameters, and use a continuation of the NPT equilibration we have just completed. We name the topology run file `md_0_1.tpr`, where the `0_1` is meant to show the times have simulated (e.g., we may want to restart and run from 1.0 to 2.0 ns).

```
gmx_d grompp -f md.mdp -c npt.gro -t npt.cpt -p topol.top -o md_0_1.tpr
```

Then, we will `mdrun` our simulation. Because of the computation requirements, we will parallelize the job over 8 compute nodes. The batch script for this is `md_0_1.pbs`, which you should submit to the queue on ACI-b. This may require a longer queue time if the cluster is particularly busy. Your simulation should take just over an hour to complete once it starts.

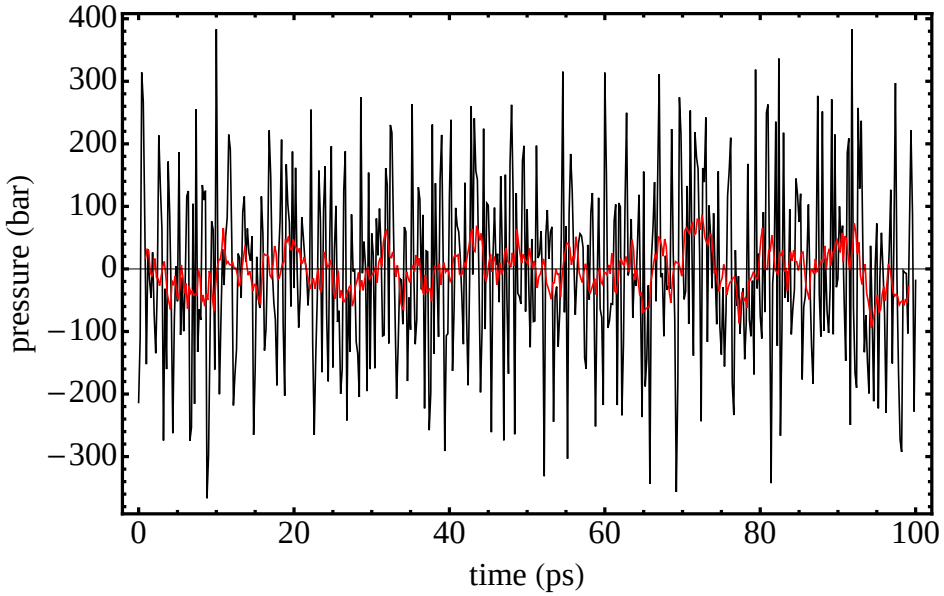


Figure 2: Pressure versus time for NPT simulation. Black shows actual pressure, red shows moving average over 10 time steps.

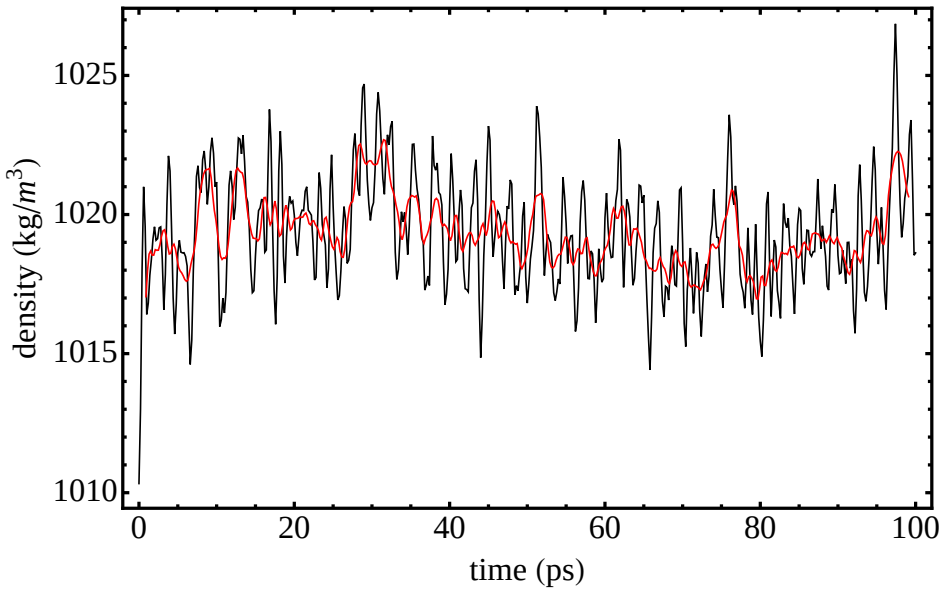


Figure 3: Density versus time for NPT simulation. Black shows actual density, red shows moving average over 10 time steps.

Once the simulation has finished, there are many different properties that we could analyze. As an example, we will analyze the radius of gyration of the protein. If R_g is stable over time, we infer that the protein has not unfolded during the simulation. First, we will convert the output trajectory to remove the periodic boundaries with `trjconv`. This can be useful for many types of analysis.

```
gmx_d trjconv -s md_0_1.tpr -f md_0_1.xtc -o md_0_1_noPBC.xtc -pbc mol -ur compact
```

Then, we can use the Gromacs tool `gyrate` to compute R_g

```
gmx_d gyrate -s md_0_1.tpr -f md_0_1_noPBC.xtc -o rg.xvg
```

Enter `1` to choose just the protein. Plot the data in `rg.xvg`. We only want the second column, which contains the 3D R_g. The resulting plot (Figure 4) should be stable over time, showing that the protein has not denatured.

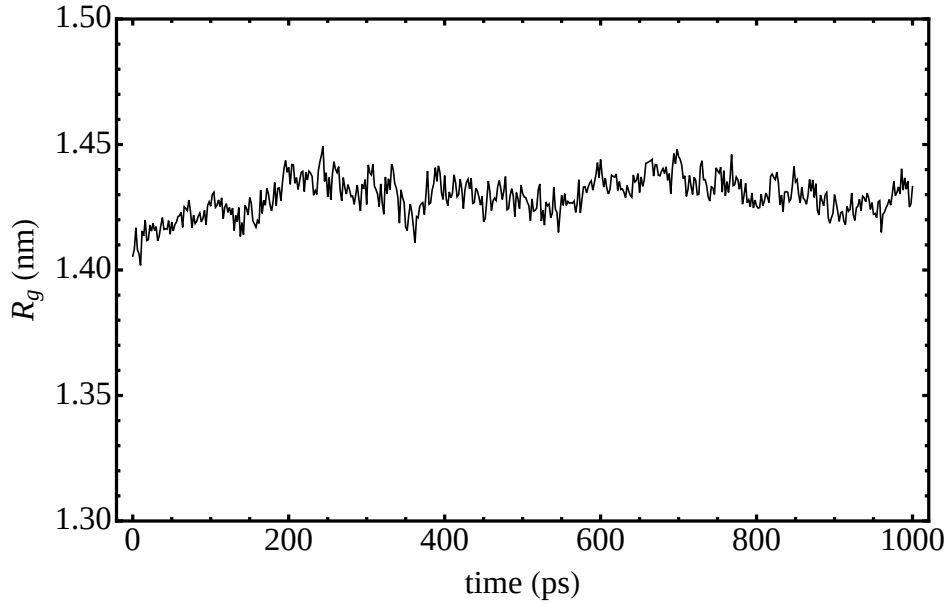


Figure 4: Radius of gyration versus time for lysozyme protein in SPC/E water.

3.6 Summary of grompp and mdrun options, workflow

The calculations above provide a good variety of `grompp` and `mdrun` jobs. However, it is also useful to have a table of the different "option flags" used by these two routines. The Gromacs manual online has a complete description of each command and its many options. Almost all the options pertain to files that are either input to the command (i), or output by the command (o). Some of the inputs and outputs are required, some are optional.

Here, we summarize the most important options for both commands. For the mandatory inputs and outputs, there are default names for the files; for example, `topol.top` is the default name for the topology file. Below, we give the default name for each file after the flag, and indicate whether this is an input (i) or output (o) file; a prime (i' or o') indicates an

optional file. If the mandatory option flags are omitted, or if the filename is omitted with an optional flag, the default name is used. Supplying an output option with filename renames the output file.

Option	Default filename	I/O/optional	Description
`-f`	grompp.mdp	(i)	MD parameter file
`-p`	topol.top	(i)	topology file
`-c`	conf.gro	(i)	configuration file
`-t`	traj.trr	(i')	high-precision configuration (often .cpt)
`-n`	index.ndx	(i')	index file, needed if .mdp refers to certain groups
`-o`	topol.tpr	(o)	topology run file
`-po`	mdout.mdp	(o)	.mdp file with all parameters

Table 1: Most common option flags for grompp.

Option	Default name	I/O/optional	Description
`-deffnm`		(i')	"base name" to construct all filenames
`-s`	topol.tpr	(i)	topology run file
`-o`	traj.trr	(o)	trajectory run file
`-e`	ener.edr	(o)	energy data run file (E, P, etc. vs. time)
`-g`	md.log	(o)	log file
`-c`	confout.gro	(o)	final configuration

Table 2: Most common option flags for mdrun.

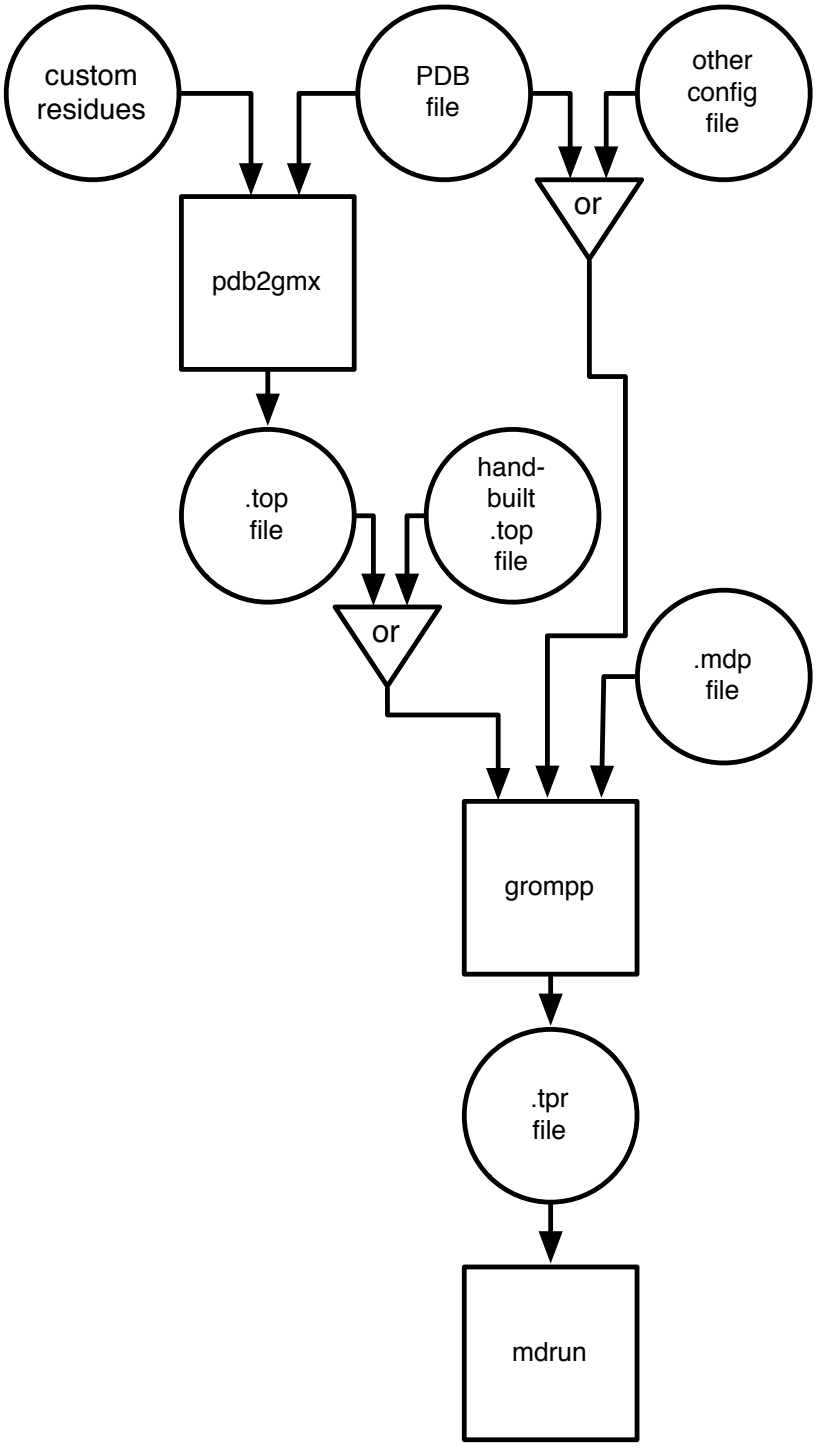


Figure 5: Flowchart of workflow, from configurations to topology files to mdrun.

4 Simulation: decane molecule

In this example, we will practice building molecules from residues which are not part of the default set (such as the amino acids for OPLS-AA). We will define new residues for decane, build a crude structure for the molecule, generate a topology file using `pdb2gmx`, and then energy minimize it to get the correct structure for a single decane molecule.

For this example, you will need the molecular visualization / drawing software **Avogadro**. (`http://avogadro.cc`).

4.1 Create new residues

We will again use the OPLS-AA force field because it represents alkanes well. On the cluster, go to your local copy of `top` that you created. From `top`, navigate to `oplsaa.ff` and open the file `aminoacids.rtp`. This file contains the definitions for all **residues** in the OPLS force field. Each residue contains at two sections: `[ atoms ]` and `[ bonds ]`. Some more complicated residues also contain sections for overriding dihedrals and for setting improper dihedrals. However, we won't worry about these sections for a simple molecule like decane.

The `[ atoms ]` section lists all of the atoms in a residue. It gives each one a name and a specific OPLS atom type. A complete listing of atom types can be found in `atomtypes.atp`. The `[ atoms ]` section also lists the partial charge on each atom, and classifies each atom into a **charge group**. A charge group is (loosely) defined as the group that, when all the partial charges are added together, combine to some integer net charge (usually zero). In a simple alkane like decane, each charge group should be net neutral. An example of a single charge group is a CH_2 in decane, since the carbon takes a partial negative charge equal to the sum of the hydrogen partial positive charges.

The `[ bonds ]` section lists which pairs of atoms are bonded to each other. You don't need to duplicate listings: for example, A B is equivalent to B A, and should only be listed once. If an atom bonds to an atom in a previous residue, you put a `-` before the atom, and if the atom bonds to the next residue, you put a `+` before the atom.

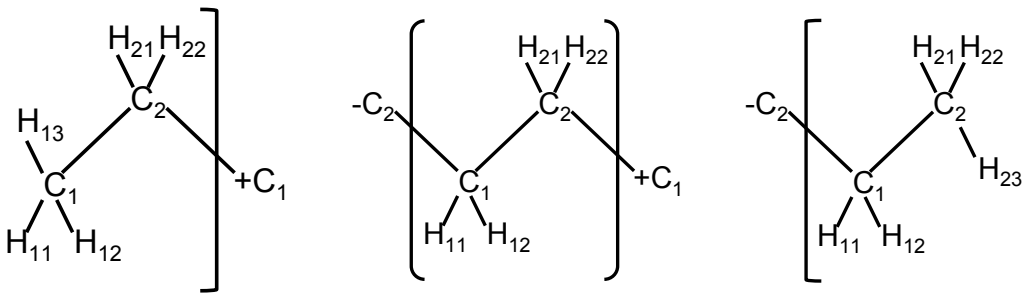
与前后残基键连原子

[atoms]

Let's make residues for decane. We will treat decane as if it is a short piece of polyethylene. The ethylene monomer is shown in Figure 6. We need to make separate monomers for the "beginning" and "end" of the chain, since at these points the residue is connected to an extra hydrogen rather than the next residue. We name each residue with a three-letter name: PolyEthylene Beginning (PEB), PolyEthylene Middle (PEM), and PolyEthylene Terminal (PET). (These names need to be unique, and not accidentally the name of an existing residue.) We have named each atom in such a way that each CH_2 or CH_3 unit is a charge group.

We need to classify each atom in the OPLS force field, and get the information for the `[ atoms ]` section of each residue.

1. Open `atomtypes.atp` and look for the atoms near `opls_135`. These are the atom types for an alkane.



(a) [PEB] (b) [PEM] (c) [PET]

Figure 6: Naming schemes for polyethylene monomers

2. Classify each atom type in each of the monomers. You should need to use `opls_135`, `opls_136`, and `opls_140`.

3. Open `ffnonbonded.itp` and find the same OPLS atoms. Record the partial charges for each atom you classified.

Check that each charge group (there are two in each monomer) are net neutral, as we would expect for an alkane.

Create a new text file called `PE.rtp`. We will work on our new residues here, and then copy our work into `aminoacids.rtp` at the end. Start with the middle monomer (PEM). The residue should look like this

```
[ PEM ]
[ atoms ]
C1     opls_136     -0.120     1
H11    opls_140      0.060     1
H12    opls_140      0.060     1
C2     opls_136     -0.120     2
H21    opls_140      0.060     2
H22    opls_140      0.060     2
```

Repeat this yourself for the PEB and PET residues.

[bonds]

Now, we need to list all of the bonds. Let's start with the middle monomer (PEM) again. The C_1 atom is bonded to H_{11}, H_{12}, and C_2, as well as to the C_2 in the previous residue. Similarly, C_2 is bonded to H_{21}, H_{22}, and C_1, as well as the C_1 in the next residue. So, this bond section should look like:

```
...
H22    opls_140      0.060     2

[ bonds ]
C1     -C2
```

```
C1      H11
C1      H12
C1      C2
C2      H21
C2      H22
C1      +C1
```

Notice that we have not duplicated the C_1-C_2 bond. Repeat this exercise for PEB and PET, and add these bond sections. After you have reviewed your work, compare your result to the `PE.rtp` file included with this tutorial.

Once you are sure that your residues are correct, open `aminoacids.rtp`. Copy and paste your residues onto the end of this file. Finally, these residues must be added to the file `residuetypes.dat` in your `~/work/top` directory, with lines

```
PEB     Other
PEM     Other
PEB     Other
```

You have now added the polyethylene residues to the OPLS force field, and can use `pdb2gmx` to create a topology from a configuration file in PDB format.

4.2 Make a PDB configuration file

Open Avogadro. We will make a crude drawing of a decane molecule, and use this as our starting configuration. To draw the molecule, select the pencil tool, and make sure `Ball and Stick` is checked in the `Display Types` menu. Click to form your first carbon (it will appear as CH_4, with hydrogens automatically added). then click on a hydrogen to add another carbon. (You can also click on a carbon and drag, to form a carbon-carbon bond, with hydrogens automatically adjusted). All-trans decane looks like a zig-zag from the side, so do your best to draw the molecule this way.

Once you have placed ten carbon atoms, click on the blue `Navigation Tool` in the tool bar. You can now drag around your screen and inspect your molecule to make sure that it is properly bonded. At this point, use the Avogadro tool to roughly energy minimize the structure (by clicking the `E` in the tool bar, then `Tool Settings`, and finally `Start` to begin minimization).

Save your file as a PDB to `decane_avogadro.pdb` file format. Then, make a second copy of this PDB file called `decane.pdb`. We will be modifying `decane.pdb`, and we don't want to lose our original work in case we make a mistake.

Atom renaming

Open `decane.pdb` in TextWrangler. Avogadro doesn't know the atom or residue names we have chosen to assign, so we must modify the PDB file.

First, delete all of the `CONECT` records and the `MASTER` record. These are not needed for Gromacs. Then, replace all occurrences of `HETATM` with `ATOM` ("ATOM" followed by two white spaces). PDB files are *fixed format* files: each number or keyword appears at a fixed

position (number of characters from the start of the line). So make sure you *preserve the spacing* when making modifications.

Column #2 are the atom numbers; we will number these sequentially at the very end of our modifications, when all the atoms are in the proper order in the file. Right now, the atom numbers correspond to the way Avogadro numbered them. But which atom is which? To find out, open `decane_avogadro.pdb` in Avogadro. In `Display Settings`, make sure `Label` is checked. Each atom should have its number on it. (If not, check the "wrench" next to `Display Settings`, to make sure the label is displaying the atom number, and not something else.)

We want the atoms to appear in the PDB file "residue by residue", and within each residue with all the carbons first, followed by all the hydrogens. We use the Avogadro picture to tell us which is the first carbon from the starting end (choose whichever end as the starting end), which hydrogens it is connected to, which is the next carbon along the chain, and so on.

So: find the number of the first carbon on the chain. Move the line for that carbon to the top of the PDB file; change its entry in Column #3 to `C1`, by replacing the space to the right of the `C` with a `1`. Find the number of the next carbon, and move that line just below the first one; make this carbon a `C2`.

Now find the hydrogens that are bonded to this `C1` and `C2`, and move them all just below the line for the `C2`. There should be five hydrogens, since this is the beginning residue. Change their atom names to the appropriate labels, by replacing the two spaces just after the `H` with 11, 12, 13, 21, or 22 as appropriate. It doesn't matter which `H` becomes which number on a specific carbon, just so long as it is supposed to be bonded to that carbon.

Repeat this process for each subsequent pair of carbons along the chain, using the Avogadro picture as a guide to identify which atom numbers you want where. For the next three pairs, there will be four hydrogens each (since they are middle residues), and then five again for the terminal residue.

Once you have all atoms renamed and organized into residues, you can replace the generic residue `LIG` in Column #4 with the proper residue name, either `PEB`, `PEM`, or `PET`. Make sure that you replace *exactly* the three characters in `LIG` to preserve the correct spacing.

Column #5 contains the residue numbers, which to start with are all `1`. These should be $1, 2, 3, 4, 5$ in sequence (this is where it useful to have atoms organized by residue). Finally, correct the atom numbers in Column #2 so that they run from `1` to `32` in order. Again, be sure to watch that the white space is preserved — the last digit should be in column 11.

To check that your PDB is correctly formatted, open the unedited `decane_avogadro.pdb` in TextWrangler, copy one of its atom lines, and paste it before the first atom line in your edited PDB file. Check that everything lines up correctly (and then make sure to delete the added line). You can check your work against `decane.pdb` which is included in this tutorial. However, note that the atom positions (columns six through eight) will be different, and the atoms may be numbered differently if you organized them in a different way.

4.3 Prepare simulation

Create a new folder on the cluster called `decane`, and upload your `decane.pdb` as well as the enclosed `em.mdp` and `min` script. The point of relabeling the PDB file and adding the

residue types, was to make a topology automatically using `pdb2gmx`.

```
gmx_d pdb2gmx -f decane.pdb -o decane.pdb -p decane.top
```

Enter 15 for the OPLS-AA force field, and then 8 not to include any water.

We set the simulation box as a cubic box with 1.0 nm of padding to ensure we have an isolated molecule.

```
gmx_d editconf -f decane.gro -o decane.gro -c -d 1.0 -bt cubic
```

4.4 Run minimization

To check that we have successfully created a valid topology file, we run an energy minimization as before, with `grompp` and `mdrun`:

```
gmx_d grompp -f em.mdp -c decane.gro -p decane.top -o em.tpr
gmx_d mdrun -s em.tpr -deffnm decane_min -c decane_min.pdb
```

Alternatively, you can execute the script `./min`, included with this tutorial. (Note the output file we produce this time has extension .pdb, which we can open in Avogadro.)

After the minimization is finished, download the PDB output file `decane_min.pdb` and open it in Avogadro. Under `Display Settings`, check `Van der Waals Spheres`. Do the same for the original `decane_avogadro.pdb`. You should see that the new decane molecule is more regular in structure, and that the van der Waals spheres overlap nicely (Figure 7).

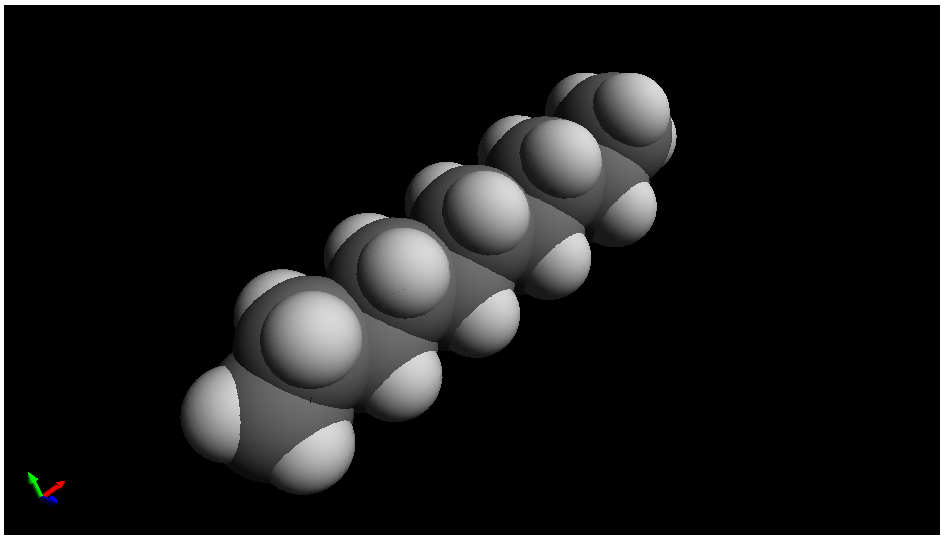


Figure 7: Energy minimized structure of decane molecule.

5 Batch job submission

During our lysozyme simulation, we submitted three batch jobs to to run on ACI-b. All other jobs were run interactively on ACI-i; that is, we executed them ourselves at a terminal session, and waited for output to appear on our screen. Batch jobs have several advantages over interactive jobs. They run on dedicated processors, so you can do other work while you wait for the job to finish. And, batch jobs can be parallelized across multiple computing nodes, to run much faster than they would on a single processor. More memory is also accessible to batch jobs than to a terminal session.

At Penn State, batch job submission is controlled by PBS. You submit jobs to a PBS queue on a given cluster; PBS manages the queue. When it is your turn, PBS executes the UNIX commands in the script you submitted, as if you were entering them yourself interactively from the command line. Any output that would have gone to the screen, is printed to a file you can read after the job has finished.

5.1 PBS Script

Each batch script start with commands that tell PBS about the job, including: which queue to submit to, the maximum run time, how much memory is needed, and how many cores (processors) are requested. Jobs with long run times or many processors requested tend to wait in the queue longer before running because the resources must become available.

Batch scripts are text files, conventionally with file extension `.pbs`. The file name should tell you something about the job; for example, `md_0_1.pbs`, which runs an MD simulation from 0 to 1 nanoseconds. Use the filename wisely.

A typical batch script on ACI-b looks like this:

```
#PBS -A stm9_b_t_bc_default
#PBS -l nodes=1:ppn=1
#PBS -l walltime=24:00:00
#PBS -j oe
echo "Job started on ‘hostname‘ at ‘date‘"
cd $PBS_O_WORKDIR
...
echo " "
echo "Job ended at ‘date‘"
echo " "
```

The first line gives the queue (account, `-A`) to access. ACI-b is awkward in that our resources there are artificially separated into three accounts:
`stm9_a_g_sc_default`, `stm9_b_t_bc_default`, and `stm9_c_t_sc_default`, with 1, 7, and 4 nodes respectively (each node with 20 cores or "processors"). Jobs that exceed the confines of the queue can run in "burst" mode, which may wait longer in queue.

The next line tells the cluster the processors that you need. "Nodes" are collections of processors that share common memory. Nodes on ACI-b have two CPU chips with 10 cores each, so 20 cores per node. As a general rule, a job should have something like 5000 atoms per core, otherwise the cores spend too much time exchanging information about atoms on

the boundary between regions handled by neighboring cores. PBS directives that request nodes and processors interact with corresponding options in the `mdrun` command, discussed in detail below (Parallelizing).

The third line sets the "wall time" for the process — the maximum amount of time that the job can run. If the job exceeds the wall time, it is immediately killed, so it is important to request enough wall time for the job. However, jobs with longer wall times wait longer in the queue. The maximum walltime for our queues is 192 hours (8 days). Long simulations often take longer than that, so we break them up into a succession of jobs, each of which starts up from the checkpoint file of the last job.

The fourth line combines the job screen output and error messages into a single file. This file will be named after your script and the computer assigned job ID number, like `md_0_1.pbs.o12345`. You can open this file after the job has finished running to analyze any written output.

After this, we use `echo` to print the start time for the job, and change directories to `$PBS_O_WORKDIR`, which is the directory **where the job was submitted**. (You typically submit jobs from the directory you want them to run in; whereas by default, PBS scripts are executed from your home directory.)

In place of the . . . , you put the actual commands you want executed, typically including some instance of `mdrun`. We don't do `pdb2gmx` or `grompp` or Gromacs analysis commands in PBS scripts, because they don't take enough compute time to be worth submitting to batch.

Finally, we use `echo` to print the end time for the job.

5.2 PBS commands

Submitting jobs

You **must** be logged into the same cluster that you specified in your PBS queue directive. For ACI, that means you must be logged into ACI-b.

```
qsub pbs_script
```

Checking job status

To check the status of all jobs running on a cluster:

```
qstat
```

To check the status of the jobs that you have submitted:

```
qstat -u <username>
```

To check the status of a specific job (with `job_id` obtained from `qstat`. You only need to use the number):

```
qstat -n job_id
```

To check what process this job is running and how much memory it is using, you can log into the node returned by `qstat -n` (it will be listed as the last line, like `lionxj12/2`. The node is `lionxj12`). Then, SSH to the node and use `top` to view all running tasks

```
ssh node_name
top
```

You can quit `top` by pressing `Ctrl+C`. Remember to `logout` from the node when you are done checking.

Modifying jobs

To delete a specific job (queued or running):

```
qdel job_id
```

To delete all jobs you have submitted:

```
qdel $(qselect -u <username>)
```

To alter a job after it is submitted but before it has started running:

```
qalter pbs_directives job_id
```

So, for example, to change the walltime to 24 hours on job 12345:

```
qalter -l walltime=24:00:00 12345
```

To alter all jobs (will throw an error if you have any running jobs):

```
qalter pbs_directives $(qselect -u <username>)
```

5.3 Parallelizing

There are two ways to run parallel jobs in Gromacs: multiple threads on a single node, or multiple nodes. These use different compiled versions of Gromacs: `gmx_d` is for multthreading, `gmx_mpi` is for multinode jobs. Each approach involves both a PBS request for the compute resources (nodes and cores), and the corresponding version of the `mdrun` command.

To execute a multithread job (multiple cores on a single node), use

```
gmx_d mdrun -nt N ...
```

which requests `N` threads on a single node. Multithread jobs are only run in batch. The batch file needs to contain a matching directive,

```
#PBS -l nodes=1:ppn=N
```

To execute a multinode job (one core on multiple nodes, communicating with OpenMPI [Message Passing Interface]), use

```
mpirun -np N gmx_mpi mdrun ...
```

where `N` is the number of cores you want to use. Note the prefacing command `mpirun`. Multinode jobs are only run in batch. The batch file needs to contain a matching directive,

```
#PBS -l nodes=N:ppn=1
```

Which is better? On Lion-X, the queue was designed to keep the machine busy, and a job that needed an entire node to run would have to wait a while for a complete node to become idle. Whereas, the queue on ACI-b prioritizes access to our resources at the expense of leaving cores idle. So on Lion-X, it made sense to submit jobs that requested many cores, one per node, because nodes were not kept idle to be ready for their owners. On ACI-b, it may make sense to do the opposite — to request 10 cores and 1 node, so as to stay within our queue resources.

The full range of options within Gromacs for running on multiple cores and nodes are a lot more varied and complicated than the above: see `http://manual.gromacs.org/documentation/5.1/user-guide/mdrun-performance.html`.

5.4 GPU support

GPUs (Graphical Processing Units) are video cards used to do parallel calculations for simulations. In Gromacs, GPUs can efficiently handle the parallel calculation of nonbonded forces. Using one or more GPUs to assist the cores running the simulation, Gromacs can be speeded up by a factor of 2–3, depending on what kind of simulation is run and what hardware is used.

A key question is how many cores per GPU is the most "efficient" ratio. The answer depends on the relative cost of cores and GPUs: if GPUs were free, it would make sense to use more of them. This question is most relevant to designing and buying computers, since a workstation or cluster node has a fixed ratio of CPUs per GPU, which limits the range of ratios you can use.

For atomistic simulations, our group has run tests that suggest ratios of 8–12 cores per GPU give a speedup in the range of 2.5–3 over using the same cores and no GPU. Reducing the ratio below about 8 cores per GPU begins to have diminishing returns, as there are not enough cores to do the rest of the simulation work, to keep the GPU "busy".

In the simplest arrangement, one or more GPUs work with some or all of the cores on a single CPU. Given the cost of GPU cards, the number of cores on present-day CPUs, and our observations of speedup, our GPU-assisted workstation has dual 14-core CPUs and 2 GPUs (instead of e.g., dual 14-core CPUs and 4 GPUs, for a ratio of 7:1).

With a rough guide of 2500–5000 atoms per core for jobs with no GPUs, 12 cores is sufficient for jobs of 30–60,000 atoms, which is the scale of most simulations our group runs. Such jobs can then run 2.5–3 times faster on the cores of a single CPU with one GPU assisting.

To run such jobs on our workstation, the command is

```
nohup gmx mdrun -nt 12 -gpu_id x ... &
```

where `x` is either 0 or 1 (the GPU ID number). Option `-nt 12` requests 12 "threads" (cores). It's a good idea to leave 2 or more cores free on each CPU, so that the workstation can continue to respond to interactive commands.

There is no batch queue on our workstation; compute jobs are submitted interactively and "run in the background". In the above command, `&` indicates the job is a background

job — the system prompt returns immediately after the command is executed. The preface command `nohup` (NO HangUP) keeps the job running until it completes, even after you log out (without `nohup`, all background jobs are killed when you log out).

To find out the status of background jobs, use `ps` (Process Status); to kill running background jobs that go awry, use `kill`.

Some XSEDE clusters at national supercomputer center sites have nodes with GPU support (Bridges at the Pittsburgh Supercomputing Center is one), with a ratio similar to our workstation. These machines use PBS; to request a single node with 12 cores and one GPU, use

```
#PBS -l nodes=1:ppn=12:gpus=1
```

which requests resources for the Gromacs command

```
gmx mdrun -nt 12 -gpu_id 0 ...
```

5.5 Automated submission with Python

If you have a series of related jobs to submit, it can be tedious to submit them all by hand. Commonly, you want to run a series of simulations in which some parameter is systematically varied, for example the temperature or pressure.

Scripts can help you automate a large series of related job submissions. Basically, you iterate over the series of jobs, create a folder for each job named in some systematic way, modify a "template" .mdp file and .pbs batch script to create .mdp and .pbs files for each job, and submit the entire series of batch jobs to the queue. Our research group uses a language called Python to write such scripts. The easiest way to learn how to write such scripts is to look at a working example.

There are two basic approaches to use a script to create a series of related .mdp and .pbs files. One approach is to prepare "template" .mdp and .pbs files, with the parameter to be varied replaced by some variable name. The Python script can then read in the file, replace the variable name by a series of values, write new files for each job, and submit those files.

The other method is to include all the text of the files to be written within the Python script, which then executes a series of `echo` commands to write the text to a new file. This can make for a rather long Python script, which does not closely resemble the .mdp and .pbs files it ultimately creates. With the "template" approach, nearly all the lines in the template files look exactly as they do in ordinary .mdp and .pbs files.

We have included with this tutorial a pair of Python scripts `write_mdp.py` and `write_sh.py`, and a pair of template files `md.mdp` and `job.sh`, which arise in computing the solvation free energy (to take a solute from vacuum into solvent). In this calculation, simulations are run at a series of values of "visibility" of the solute to the solvent, which parameter (called lambda) appears in the .mdp file. The script `write_mdp.py` creates a series of .mdp files from the template `md.mdp` (and three other similar templates not included); the script `write_sh.py` creates a corresponding series of batch script files from the template file `job.sh`.

The two Python scripts are commented so you can at least see what is going on, if not precisely why the Python commands have the form they do. To really learn to write in Python, there are several good resources online, including (in order of complexity)

- https://www.stavros.io/tutorials/python/
- http://www.tutorialspoint.com/python/index.htm
- http://www.diveintopython3.net
- https://docs.python.org/3/tutorial/
- http://openbookproject.net/thinkcs/python/english3e/

6 Advanced topics

In this section, we will briefly highlight some advanced features of Gromacs. More details can be found in the Gromacs manual, online, or by asking experienced group members.

6.1 Running faster

Everyone wants their simulation to run faster. There are several ways to coax a few more nanoseconds per day out of Gromacs.

Add processors. The same size system running on twice the number of cores will run twice as fast, until each core is handling so few atoms that cores waste lots of time swapping information with neighboring cores about atoms on the boundaries between domains. But diminishing returns is a crossover, so it may be tempting to push the number of atoms per core down from 4–5000 to closer to 1–2000. Do timing studies: see how many ns/day/core you are getting with your system before wasting valuable resources.

Use GPUs. GPUs (Graphical Processing Units, i.e., video cards) have amazing parallel computation capability, now being harnessed by Gromacs and other simulation codes to handle nonbonded (Lennard-Jones) interactions. At present, the ACI cluster has no nodes with GPU support; our group has a single-node workstation, with 2 GPUs supporting twin CPUs with 14 cores each. The CPUs on this workstation are comparable to those on ACI nodes, and the GPU support speeds typical jobs by factors of 2.5–3. Some national supercomputing center (XSEDE) machines also have GPU support. See above for details on how to run Gromacs jobs on GPU nodes.

Constrain bonds. Molecular bonds are stiff and vibrate at high frequencies. It takes a very short timestep to follow this rapid motion accurately, which for most purposes is not interesting anyhow. In the .mdp file, you can specify that bond lengths be constrained to be constant. The dynamics then uses a special algorithm called LINCS to satisfy the constraints.

With constrained bond lengths, timesteps can often be boosted from the usual 1fs to more like 2fs. To check whether your timestep is too large, check the stability of the energy versus time in a short simulation with the thermostat turned off.

Typical parameters are

```
constraints = all-bonds
constraint_algorithm = lincs
lincs-iter = 1 ; a single iteration
```

```
lincs-order = 6 ; sets accuracy
```

See Gromacs manual section 7.3.18 and
`http://www.gromacs.org/Documentation/How-tos/Removing_fastest_degrees_of_freedom`.

Note that bond lengths in water are automatically constrained (using a special algorithm called SETTLE), unless water is explicitly declared flexible in the .mdp file with `define -DFLEXIBLE`.

Two times when you should not constrain bond lengths are 1) minimizing the energy (the minimizer does not work well with constraints), and 2) computing normal modes (see below).

Virtual sites for hydrogen. A step beyond constraints is to "make H atoms virtual sites", i.e., to determine the position of H atoms at each timestep by applying deterministic rules about bond lengths and angles. For molecules with .itp files built by `pdb2gmx`, this can be done by calling `pdb2gmx` with option `-vsite h`.

A leading Gromacs developer (David van der Spoel) reports that his usual setup is to use virtual site hydrogens, fixed bond lengths and LINCS, and a 4fs timestep.

Note that when boosting the timestep in this way, the parameter `nstlist` specifying the steps between neighbor list rebuilds should be decreased from the usual value of around 10 to more like 5 (as the timestep is doubled from 2fs with only bond constraints, to 4fs with vsite hydrogens), so that the elapsed time between neighbor list rebuilds is constant. See `http://www.gromacs.org/Documentation/How-tos/Speeding_Up_Simulations`

Shorten the cutoff. If you're really desperate, one way to speed up a simulation at the expense of quantitative accuracy is to shorten the nonbonded cutoff distance, which reduces the number of pairwise interactions computed. Decreasing the cutoff by a factor λ reduces the number of particles within range by a factor of λ^3, so reducing the cutoff from 2.5σ (a standard recommendation for LJ interactions) to 2σ is worth a factor of 2 in speed.

Change the physical parameters. If you're simulating a glassy fluid, things will be slow near the glass transition; maybe you need to raise the temperature a bit. If you're simulating a polymer melt or concentrated solution, things will be slow if the chains are long enough to become entangled; maybe you need to shorten the chains a bit. (If your *aim* is to study glassy fluids or entangled polymers, you must think carefully about how close to T_g or how entangled a melt you can approach.)

Simplify the model. Atomistic simulation is appealing because it represents real molecules with chemical specificity, but some systems are too big and too slow to simulate atomistically. There are various levels of "coarse-graining" that can be applied. For example:

1. "united-atom" models, in which hydrogens are lumped together with the atoms to which they are bonded;
2. Martini models, in which 2–3 heavy atoms at a time are lumped together as a single "bead";
3. bead-spring models, in which a polymer chain is idealized as a sequence of beads bonded by springs, with the bead representing some small polymer segment of several monomers or more.

6.2 Walls

Gromacs provides support for included Lennard-Jones 9-3 or 10-4 walls. A 9-3 wall is equivalent to interacting with a semi-infinite slab of a given atom type, and can be useful for equilibration. The 9-3 wall potential is

$$U = \pi\rho_j \left(\frac{C_{ij}^{(12)}}{45h^9} - \frac{C_{ij}^{(6)}}{6h^3} \right) \tag{5}$$

where

$$C_{ij}^{(12)} = 4\epsilon_{ij}\sigma_{ij}^{12} \tag{6}$$

$$C_{ij}^{(6)} = 4\epsilon_{ij}\sigma_{ij}^{6} \tag{7}$$

Walls are placed along the z-axis, and you can choose to include either one or two walls. A typical set of `.mdp` parameters looks like

```
; wall
nwall          =  2 ; number of walls
wall_atomtype  =  opls_135 opls_135 ; atom type for each wall
wall_type      =  9-3 ; choose the potential
wall_r_linpot  =  0.03 ; 0.3 A
wall_density   =  100 100 ; wall density (nm^-3)
```

The parameter `wall_r_linpot` lets you set a distance beyond which the potential is continued linearly (with a constant force). This forces atoms that lie beyond the wall back into the system. It is a good idea to always include this parameter if there is a chance atoms may be placed beyond the wall, since a fatal error will result without it. Setting the distance as very close to the wall will have negligible effect on your simulation physics (since atoms will almost always prefer to be farther than the vdW radius from the wall).

Sometimes you may want to simulate a wall as an interaction with many atoms. However, you are limited to a single wall-type in Gromacs. A way to avoid this is by introducing your own "custom" wall atoms. To do this, create a dummy name for your atom (such as "wallatm") and append it to `atomtypes.atp` and `ffnonbonded.itp` in your force field. For `atomtypes.atp`, you can enter `0.0` for the mass column because it will not matter for a wall. For `ffnonbonded.itp`, you can again add zeros to the columns. Follow the example of the section of `special dummy-type-particles` in `oplsaa.ff`. Then, create a new file called `walls.itp` in your force field. Append the following line to `forcefield.itp`:

```
#include "walls.itp"
```

Open `walls.itp`. Here, you can define the Lennard-Jones parameters for each atom type in your system interacting with your dummy wall atom. A sample `walls.itp` for the OPLS-AA force field looks like this

```
[nonbond_params]
; i        j      func   sig   eps
```

```
; iPP wall @ 300 K from s.c. lattice
; wall_density = 100
ppwall     opls_135     1  3.36558e-01  2.09866e-01 ; wall-CH3
ppwall     opls_136     1  3.36558e-01  2.09866e-01 ; wall-CH2
ppwall     opls_137     1  3.36558e-01  2.09866e-01 ; wall-CH
ppwall     opls_140     1  2.84443e-01  1.41492e-01 ; wall-H
```

The Lennard-Jones parameters are chosen according to the following formulae

$$\bar{C}_i^{(12)} = \frac{1}{\rho_w} \sum_j^n 4\rho_j \epsilon_{ij} \sigma_{ij}^{12} \tag{8}$$

$$\bar{C}_i^{(6)} = \frac{1}{\rho_w} \sum_j^n 4\rho_j \epsilon_{ij} \sigma_{ij}^{6} \tag{9}$$

$$\bar{\sigma}_i = \left(\frac{\bar{C}_i^{(12)}}{\bar{C}_i^{(6)}} \right)^{1/6} \tag{10}$$

$$\bar{\epsilon}_i = \frac{(\bar{C}_i^{(6)})^2}{4\bar{C}_i^{(12)}} \tag{11}$$

where ρ_w is a free choice for the wall density. It is advisable to choose a wall density so that the Lennard-Jones parameters are of the same order of magnitude as typical nonbonded parameters. Remember that your wall density in your `.mdp` file must match the wall density that you choose.

6.3 Chain building

We illustrated one scheme for building polymer configurations using Avogadro and topology files using `pdb2gmx`, for the simple example of decane. This scheme is workable but increasingly tedious for very long molecules. It is particularly painful if the system of interest contains *random* copolymers, in which different chains have a different sequence of the same types of monomers. Then each different sequence is in effect a different molecular species, requiring its own unique topology file. An example of such a system is atactic polypropylene, in which the methyl groups on each monomer are randomly on one side or the other of the all-trans chain. If we want to have a sufficient variety of structures for such chains, we have an unpleasantly large number of PDB files to generate and clean up.

In such cases, we can use Mathematica to help us build multiple PDB files in which random choices are made between left- and right-handed monomers. The multiple PDB files can then be processed by `pdb2gmx` to generate multiple topology files. The basic idea is to create small PDB files for the monomers in your molecules, hand-edit the atom and residue names in those PDB files. Then we "glue" the monomers together in Mathematica, renumbering the atoms and the residues in sequence, to create large PDB files for a polymer chain.

First, we generate small PDB files for the monomers. One way to do this is to draw an oligomer in Avoagdro, energy minimize it, and then delete atoms until only the monomer you want remains. It's helpful to retain as well the "previous" and "next" carbon atoms, so you know how to translate the monomer positions to build an all-trans chain. Then, you strip the PDB file to just the bare minimum information about the atoms, and carefully reorder and rename the atoms to correspond to the residue definitions you have made.

The Mathematica notebook `pdb2chain.nb` has methods for assembling the chains from monomer PDB files, and for writing out the properly formatted PDB file for the entire chain. This notebook is pretty well documented, so you can read through it for details. Each new monomer is translated by an appropriate vector from an arbitrarily chosen origin along a new chain "backbone". This builds the chain in the all trans configuration. Rotations and alignments can then be performed on the final chain.

6.4 Self-connectivity through periodic boundary

It is sometimes desirable to bond a molecule to itself through the periodic boundary. This effectively creates an infinitely long chain, which can be useful for studying polymer crystals without end effects. However, `pdb2gmx` cannot generate topologies for these systems. The simplest approach in such cases is to build the ordinary molecule first, use `pdb2gmx` to generate its topology file, and then hand-edit the topology to remove the terminal atoms, and include the "extra" bonds, angles, and dihedrals resulting from the connection through the periodic boundary.

If you instead only want to build a simple ring polymer, you can still use `pdb2gmx`, if you edit the file `specbond.dat` (Special Bonds) to construct the bond between the beginning and the end of the chain. Documentation of `specbond.dat` is available on the Gromacs website. This method does not work for linear chains self-connected through the boundary because `pdb2gmx` is not aware of the periodic bonding of the molecule when it checks the tolerance radius for building the "special" bonds.

6.5 Normal mode analysis

The phonon spectrum for a system can be computed with normal mode analysis. This is most useful for crystalline systems, for which the set of deformations is well described by a set of normal modes about the lowest-energy state. The total free energy of a system can be computed analytically, in the approximation that the normal modes are not interacting (which is true if the amplitudes of thermal oscillation are small enough). The total free energy of a system of decoupled quantum harmonic oscillators is

$$F = U_0 + \sum_i^{3N-3} \frac{\hbar\omega_i}{2} + \frac{1}{\beta} \sum_i^{3N-3} \ln\left(1 - e^{-\beta\hbar\omega_i}\right) \tag{12}$$

where U_0 is the simulated potential energy, and ω_i are the phonon frequencies. The sums are taken over all non-zero normal modes. There should 3 zero modes for a system with periodic boundaries in all directions and no walls, since there are modes of motion (bulk translation in x, y, z) that result in no change for the system.

For a normal mode analysis, thorough energy minimization (force tolerance much less than 1 in Gromacs units) is required. Typically, a force tolerance between 10^{-4} and 10^{-2} is sufficient, although this will depend on your system. Minimization should be done using the L-BFGS minimizer. If machine precision is reached before the force tolerance is met, you may need to perturb the structure and try minimizing again, since machine precision is not sufficient to guarantee you are close enough to the energy minimum. Then, modify the `.mdp` file to use the `nm` integrator. This will construct a Hessian matrix for you in `.mtx` format. Then, use `g_nmeig` to diagonalize the matrix, and obtain the eigenvalues.

Memory usage for the matrix construction and diagonalization scales very unfavorably, so a limit of about 7,000-8,000 atoms is all that is practical for this type of analysis.

6.6 Center-of-mass pulling

You can use Gromacs to "pull" on certain groups of atoms. This can be useful for umbrella sampling, for example. Another potential that can be applied is constant force pulling. You use the `.ndx` file (most easily written in Mathematica) to define two or more pull groups. Then, you use the `.mdp` file to specify the pull potential, the dimensions for pulling, and how hard to pull. Numerous options are available, and are documented in the Gromacs manual and (better) documented on Justin Lemkul's website. Here is an example `.mdp` file for 1D constant force pulling.

```
; pull code
pull          = constant_force
pull_geometry = distance
pull_dim      = N N Y
pull_ngroups  = 1
pull_group0   = Cleft
pull_group1   = Cright
pull_k1       = -1500; MINUS the force you want
```