A guide to Gromacs

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

September 2, 2020

Department of Chemical Engineering
The Pennsylvania State University, University Park, PA 16802
Email: stm9@psu.edu

Contents

1 Getting started

This tutorial is written to help a new user learn to run simulations using GROMACS (GROningen MAchine for Chemical Simulations). This tutorial is not a substitute for the Gromacs manual, which is well-written and comprehensive, available online and as a pdf. Throughout this document, text in `this font` are commands and keywords verbatim.

Gromacs 5 is installed on the Penn State cluster, called ACI. (`https://ics.psu.edu/advanced-cyberinfrastructure/`). This tutorial 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 run on multiple cores on a single cluster node (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 smaller simulations (up to roughly 10^5 atoms), and MPI for larger simulations.

All the many programs within Gromacs are accessed from just one executable, called `gmx`. The double-precision version, which is rarely used, is called `gmx`.

ACI-i and ACI-b

The ACI cluster has two "faces": ACI-i (interactive) and ACI-b (batch). ACI-b is for submitting and running batch jobs — big calculations that use multiple compute cores and take a long time to run. (For how to submit and run batch jobs, see Section 5 below.) ACI-i is intended for using programs that are intensively "visual", i.e., programs that open windows to display things.

ACI-i and ACI-b share a common filesystem, but not all the same software is available on both. The main difference ultimately is how the two systems are accessed. ACI-i is accessed via a web interface called Open On Demand, at `https://portal.aci.ics.psu.edu`. (For Mac users: this webpage does not work with Safari; use Chrome or Firefox.)

From this webpage, under `Interactive Apps`, the first item is *ACI Interactive Desktop*. Selecting this item, you can request an interactive desktop session, which allocates multiple cores to you for use with computationally demanding visual applications (such as Mathematica, Matlab, COMSOL, Materials Studio, and the like).

After a few minutes, the desktop is ready to launch; once launched, you see a "visual desktop" reminiscent of the desktop on a PC or Mac; The applications however are all Unix apps. The app of main interest is the Terminal Emulator (which provides a command line connection to ACI).

The web-based connection to ACI has pros and cons:

- It is web-based, so requires no software install on your computer.
- The desktop environment resembles a PC or Mac.
- However, the software is all Unix, so the resemblance is not very helpful.
- It takes several minutes to launch a session, so it is inconvenient to quickly logon.

- If you stay logged on, you are preempting multiple cores others may need.

For these reasons, we recommend logging onto ACI-b, as described in the next section.

Logging onto ACI-b

To log into ACI-b from your computer (PC or Mac), use a "terminal application". On the Mac, use either Terminal (comes with OS X) or iTerm2 (improved version of Terminal, downloadable from `https://www.iterm2.com`). On the PC, use PuTTY, downloadable from `https://www.putty.org`.

To run programs on the cluster than are "visual" (open a window to display pictures, or take input from the cursor and mouse), you need an "XWindows" terminal application. On the Mac, use XQuartz (`https://www.xquartz.org`); on the PC, use XMing (`https://sourceforge.net/projects/xming/`).

To logon from a terminal window, enter

```
ssh  <username>@<hostname>
```

where hostname (see below) is the cluster you are logging onto (see table below). You will be prompted for your password, and then 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/`.

If you are logging onto ACI-b and want to use window-based programs, logon with `ssh -X` from a terminal window. The option `-X` enables "X forwarding". On Mac, XQuartz will be launched automatically.

Text editors

To work with a computer like ACI that runs a Unix operating system, a good text editor is necessary, both on your "local" computer (your Mac or PC desktop or laptop), and on the Unix machine itself. Word processor apps such as Microsoft Word insert invisible formatting characters to control the appearance of documents; these formatting characters play havoc with Unix text files. Don't use word processor apps for editing files for ACI or other Unix machines.

On ACI, the simplest text editor for users accustomed to the modern visual mouse-and-menu interface is `gedit`. `gedit` can be launched from the command line, either with or without a filename to edit; it automatically opens a file window in XQuartz or XMing. To avoid a long list of spurious error messages on launching `gedit`, define an alias in your `.bashrc` file,

```
# avoids dbus errors on launching gedit
alias gedit='dbus-run-session gedit'
```

The old-school choice for text editing under Unix is `vi`. `vi` was designed before the advent of the mouse-and-menu interface, so learning `vi` takes some practice. Once learned, `vi` is a fast, powerful, and convenient editor, which is universally available on all Unix computers.

It is also useful to have a text editor on your local computer, to prepare files for use on the cluster without being logged in. On the PC, you might try Notepad++, available at `https://notepad-plus-plus.org`.

On the Mac, a good text editor is BBEdit, available at `https://www.barebones.com/products/bbedit/`. Another option on the Mac is to use `vi`, which can be launched from a terminal window. (Indeed, the terminal window on the Mac is a console for the Unix machine that underlies the Apple visual interface.)

File transfer

A necessary utility for working with ACI or other remote computers is a way to transfer files to and from your local computer to the remote system. One convenient way to upload and download files from ACI is to use **Cyberduck**, available at `http://cyberduck.io`.

In Cyberduck, "bookmarks" can be made for connections to remote machines. 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-b (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, drag them to your local system.

Cyberduck can be used together with BBedit to edit cluster files, by automatically downloading a copy to edit on your laptop/desktop, then automatically uploading it again when you save. From within Cyberduck Preferences/Browser, set double-click to open a file in the editor; in Preferences/Editor, choose BBEdit as the default for opening files.

Finally, the Open On Demand webpage (see above) has an interface to your files on ACI (under the Files top menu), which can be used to upload and download files from your local computer to ACI.

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. The first commands load software used by Gromacs:

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

Then, Gromacs itself is loaded. The Milner research group prefers to use its own compiled version of Gromacs (because the ACI version of Gromacs is double-precision, which is a factor of two slower). The Milner group version of Gromacs is loaded with the following command:

```
source /gpfs/group/stm9/default/SOFTWARES/gromacs-5.1.4STM/single/bin/GMXRC
```

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.)

You can verify that you have access to Gromacs using the `which` command, as in

```
[stm9@aci-lgn-001 stm9]$ which gmx
/gpfs/group/stm9/default/SOFTWARES/gromacs-5.1.4STM/single/bin/gmx
```

(`which` searches for executables on your "search path".) You can verify that the commands are working, with `gmx --version` (which prints information about the version installed) or `gmx -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 /gpfs/group/stm9/default/SOFTWARES/gromacs-5.1.4STM/share/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`.

1.1 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. *Mathematica.* Mathematica is a powerful language for numerical and analytical calculations. Mathematica can be run on ACI, as well as installed and run on your personal computer. To use it, load with:
   `module load mathematica/<version>`

2. *VMD: Visual Molecular Dynamics.* VMD is useful for visualizing configurations and simulation results. As of this writing, `vmd` is installed on ACI only in the Milner group space. 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
```

2 Example simulation

A simulation run in Gromacs (or any comparable platform) consists of a set of tasks and associated input files. In general, the input to a simulation consists of:

1. The starting *configuration*, i.e., the initial positions and velocities of all the atoms or particles.

2. The *topology* of the system — a list of the number and type of molecules your simulation contains, and the structure of each type of molecule, including what kind of atoms it contains, and how they are bonded together.

3. The *force field*, which determines the forces on each atom in terms of the locations of bonded neighboring atoms and nonbonded nearby atoms.

4. Simulation *parameters*, including things like the system dimensions, temperature, pressure, run time, and timestep, as well as how often to save various simulation coordinates (particle positions, velocities, forces, and energies).

Executing a single simulation involves two steps: preprocessing the input files to produce a "run file" used by the simulation itself, and running the simulation to produce various output files. The output files are what we want, and what we use to analyze the system. They include:

1. The *trajectory* file (positions of all the atoms in a succession of "snapshots" or "frames").

2. The "energy" file (values of the kinetic and potential energy, pressure, temperature, and other such quantities at a succession of timesteps).

3. Other special output files, such as the position of a given group of atoms, or the force acting on that group of atoms, at a succession of timesteps.

A full simulation starting from scratch typically consists of four simulation steps, the first three of which are used to equilibrate the system. Often, the starting configuration is not representative of typical atomic arrangements. If the atoms are awkwardly placed, forces can be large, which would lead to violent motion when a simulation starts. This is dealt with by a first step of energy minimization, which relaxes the atom positions until the forces are small. After this, local arrangements and atom velocities may still be unrealistic, which can be alleviated by simulating for a while at constant volume and temperature (often

written "NVT", for constant Number, Volume, and Temperature). After this, the density of the system may be unrealistic; this can be alleviated by simulating for a while at constant pressure and pressure (often written "NPT"). This sequence of three equilibrations — energy minimization, NVT, and NPT — is typical before a "production run", i.e., a simulation which we actually observe and measure.

Here, we will simulate a lysozyme protein solvated in water, in which this three-step equilibration protocol is used. This example was originally developed by Justin Lemkul; his website (`http://www.bevanlab.biochem.vt.edu/Pages/Personal/justin/gmx-tutorials/`) is an excellent resource for Gromacs help.

2.1 System generation

In this example, we are given a PDB file for the protein (`1AKI.pdb`), which serves as the initial configuration. If this were not supplied, a reasonable starting configuration would need to be generated. (This can be challenging; below, we consider the task of generating a reasonable configuration for a melt of decane molecules.)

Topology

First, we make a topology file for the protein using `pdb2gmx` ("PDB to Gromacs"):

```
gmx pdb2gmx -f 1AKI.pdb -o 1AKI.gro -p 1AKI.top
```

The OPLS-AA force field includes a special .rtp (Residue ToPology) file for amino acids, which tells `pdb2gmx` how to build the topology file. Below, we solvate the protein with water, which we shall represent with the SPC/E water model. Correspondingly, when prompted, type `16` to choose the OPLS-AA force field, and type `7` to choose the SPC/E water model.

The .top file `pdb2gmx` delivers is a "standalone" topology file, which contains in addition to the molecule topology definitions, the additional lines to read in the forcefield and define the system. We prefer to strip this file down to a "molecule .itp" file, which only defines the 1AKI protein. To do this, rename the file `1AKI.top` as `1AKI.itp`, then delete the lines up to `[ moleculetype ]` at the top, and the lines after the position restraint (starting with `; Include water topology`) at the bottom.

Then we write our own `system.top` topology file:

```
#include "oplsaa.ff/forcefield.itp"
#include "oplsaa.ff/spce.itp"
#include "oplsaa.ff/ions.itp"
#include "1AKI.itp"

[system]
1AKI in water

[molecules]
; name      count
1AKI    1
```

Simulation box

Now we use `editconf` to change the system size:

```
gmx editconf -f 1AKI.gro -o 1AKI.gro -c -d 1.0 -bt cubic
```

Our system will have periodic boundaries, so it needs to be big enough that the protein does not interact with its periodic images. 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 interactions are negligible. Note that this use of `editconf` overwrites the file `1AKI.gro` with the new version.

Solvation

Now we surround the protein with water, using `solvate`:

```
gmx solvate -cp 1AKI.gro -cs spc216.gro -o 1AKI_solv.gro -p system.top
```

The option `-cp` denotes the protein configuration, and `-cs` the solvent configuration, which is `spc216.gro` for SPC, SPC/E, and TIP3P water. The option `-p` updates the topology file with the number of water molecules. (this convenience works if we are solvating with water).

The existing file `system.top` is overwritten, and a backup created with `##` wrapped around the name. (Gromacs does this automatically for all files that are overwritten.) In the new `topol.top`, a line has been ended at the end to include the new water molecules. Correspondingly, their configurations have been added to the end of `1AKI_solv.gro`.

Add anions

Because lysozyme is charged (+8), we use `genion` to add anions to neutralize the system. Awkwardly, `genion` requires a `.tpr` file as input. So we create one with `grompp` first, using any plausible `.mdp` file, in this case `em.mdp`, which is the .mdp file we use later for the energy minimization.

```
gmx grompp -f em.mdp -c 1AKI_solv.gro -p system.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 (typically Na^+ and Cl^-). We use `-nn 8` to add 8 anions:

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

Type `13` to declare the ions part of the solvent group. In the new .gro and .top files, 8 solvent molecules have been replaced with 8 chloride ions. The file `system.top` is again updated with the count of CL ions at the bottom.

The system configuration and topology files are now assembled and ready for equilibration, which proceeds in three steps, as described above.

2.2 Energy minimization

First, we perform an energy minimization to ensure there are no large interatomic forces in the starting configuration, which would cause a simulation to "blow up". We run `grompp` to prepare for energy minimization:

```
gmx grompp -f em.mdp -c 1AKI_solv_ions.gro -p system.top -o em.tpr
```

In the .mdp (MD Parameter) file `em.mdp`, we use steepest-descent minimization `steep`, and a reasonably generous force tolerance of 1000 kJ/mol/nm, which is still sufficient to eliminate large forces.

After processing, we run the minimization with:

```
gmx mdrun -deffnm em
```

The option `-deffnm` (DEFine File NaMe) sets a default file name for all inputs and outputs, so we need not specify each file individually. For example, the expected input .tpr (ToPology Run) file is `em.tpr`, created above by our preprocessing step.

By typing this command, the energy minimization job is done "interactively", as we wait. Output from this command, typed at the command line, appears directly on the screen. This is not a good approach if the command takes a long time or generates a lot of output. In fact, this is not how we normally execute `mdrun` commands; instead, we submit batch scripts to the queue (see below).

2.3 NVT equilibration

After minimization, we equilibrate the system at fixed temperature and volume (NVT). In this step, we restrain the heavy atoms in the protein, so that the water can equilibrate around the protein without disturbing the delicate protein folded structure. Details can be found in the .mdp file `nvt.mdp`.

Again, we perform the preprocessing step, as:

```
gmx grompp -f nvt.mdp -c em.gro -p system.top -o nvt.tpr
```

Note how `grompp` takes the initial configuration (`-c` option) as `em.gro`, which was the last frame of the energy minimization.

For a system of this size (over 30,000 atoms), a significant amount of computing power and time is needed to run this simulation. Instead of running the job interactively (as we did with the energy minimization), we submit it to the batch queue, which runs the job. To run faster, we run the job on multiple cores, which share the work of the simulation.

The "batch script" `nvt.sh` contains the code to submit the job. The script `nvt.sh` begins with directives to the queue, which request resources (nodes, cores, running time, and memory). Then the script uses essentially the same `mdrun` command we would type at the console to run the job.

To submit the job, enter:

```
qsub nvt.sh
```

To check the status of your jobs, enter

```
qstat -u <userid>
```

(replace `<userid>` with your userID). Once it starts, the job should take about around 10 minutes to run. Batch scripts will be discussed in more detail below.

After the simulation runs, we can check that the temperature has equilibrated. Use `gmx energy` to extract from the .edr (Energy Data Run) file the variation with time of important quantities:

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

Enter `15 0` to select only the temperature.

The resulting file `temp.xvg` can be opened with Grace (type `xmgrace temp.xvg`). or using the Mathematica notebook `xvg_reader.nb`. Plot the temperature versus time; it should stabilize and fluctuate around 300 K (Figure 1).

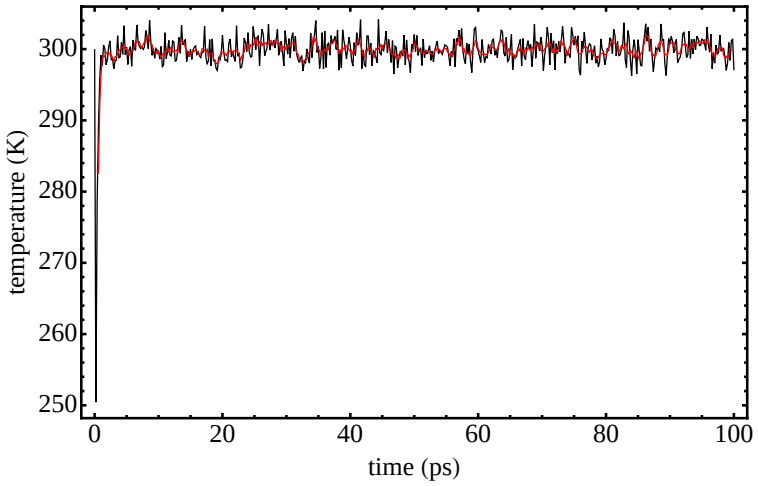


Figure 1: Temperature versus time for NVT simulation (black), and moving average over 5 timesteps (red).

2.4 NPT equilibration

The third step in our equilibration protocol for lysozyme in water is an NPT simulation, in which a barostat is used to set the system pressure to a desired value. This allows the simulation box size to adjust. During this step, we again restrain the protein position to prevent its unfolding as the surrounding fluid equilibrates. The preprocessing command takes the form

```
gmx grompp -f npt.mdp -c nvt.gro -t nvt.cpt -p system.top -o npt.tpr -maxwarn 2
```

The option `-t` uses the "checkpoint" (`.cpt`) file from the previous NVT equilibration as the initial configuration. Alternatively, we could rely on `-c nvt.gro`, which is the last saved

frame of the NVT run. In general, checkpoint files allow simulations to be restarted using the exact positions and velocities they ended with.

Restarting from a checkpoint is useful 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.

We run the NPT simulation as a batch job on ACI-b, just as we did for the NVT simulation, this time submitting the job script with `qsub npt.sh`. Once started, the job will take about 10 minutes to run.

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

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

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

```
gmx 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 the pressure stabilizes to an average of about 1 bar, with substantial fluctuations about this value (Figure 2).

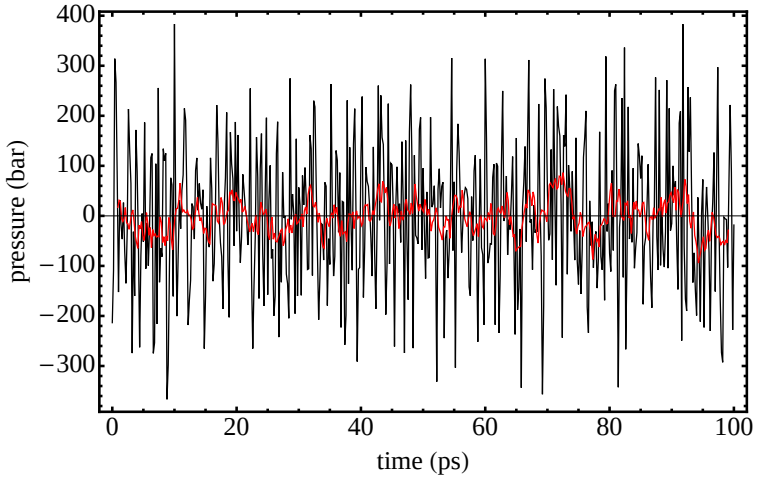


Figure 2: Pressure versus time for NPT simulation (black), and moving average over 10 timesteps (red).

Correspondingly, the density should have stabilized close to the density of water, since the system is mainly water (Figure 3). so we are now equilibrated with respect to temperature, pressure, and density.

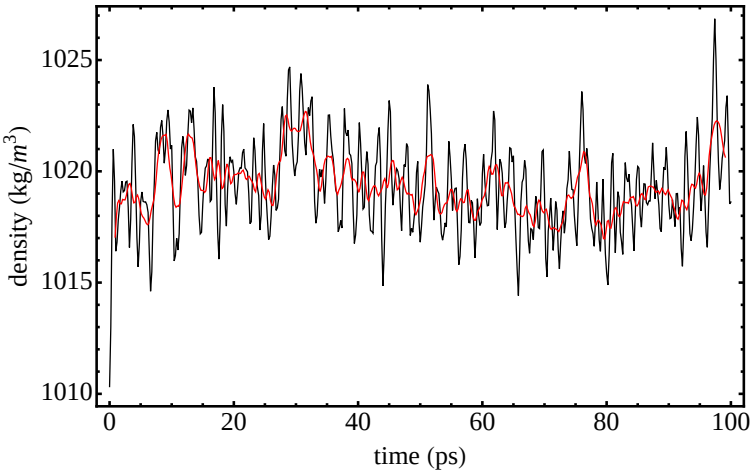


Figure 3: Density versus time for NPT simulation (black), and moving average over 10 time steps (red).

2.5 Production MD

For the final production run, 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 is rather short in practice, but sufficient here for illustrative purposes). The `grompp` command here takes the form

```
gmx grompp -f md.mdp -c npt.gro -t npt.cpt -p system.top -o md.tpr
```

in which the `-t` option again restarts the new simulation from the checkpoint of the NPT run.

The batch script for the production md run is submitted with `qsub md.sh`, which is submitted to the batch queue as before. This run should take about an hour to complete.

Once the simulation has finished, we can use the visualization program VMD to examine the trajectory. (See `https://www.youtube.com/watch?v=LqKNzaHLfi0` for a tutorial on how to use VMD.)

Many different properties can be analyzed from a simulation trajectory .trr file. As an example, we will measure 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 use `trjconv` to convert the output trajectory to remove periodic boundaries.

```
gmx trjconv -s md.tpr -f md.xtc -o md_noPBC.xtc -pbc mol -ur compact
```

Then, we use `gyrate` to compute R_g:

```
gmx gyrate -s md.tpr -f md_noPBC.xtc -o rg.xvg
```

Enter `1` to select just the protein.

Plot the resulting data in the file `rg.xvg`. We are most interested in 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.

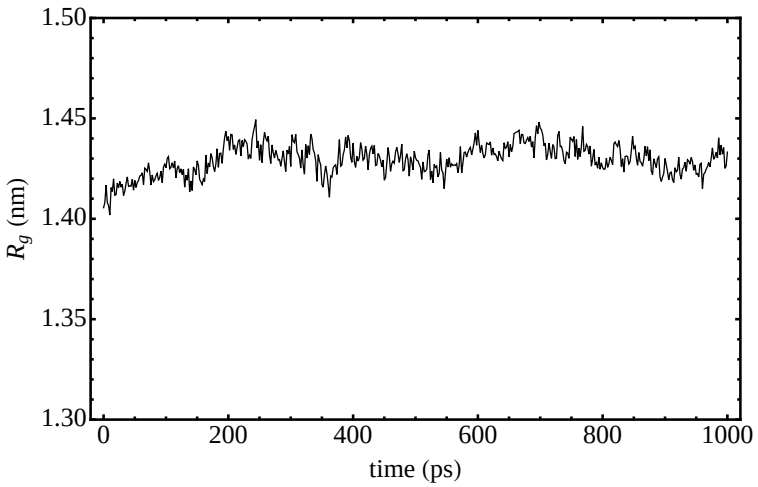


Figure 4: Radius of gyration versus time for lysozyme protein in SPC/E water.

2.6 Grompp and mdrun options; workflow

The calculations above exhibit a variety of `grompp` and `mdrun` jobs. It is useful to have a table of the different "option flags" used by these two routines, which highlights the common syntax of the different commands. The Gromacs manual online (`http://manual.gromacs.org/documentation/5.1-current/index.html`) has a complete description of each command and its many options. For a list of commands by name, see `http://manual.gromacs.org/documentation/5.1-current/user-guide/cmdline.html#commands-by-name`; for a list of commands organized by topic, see `http://manual.gromacs.org/documentation/5.1-current/user-guide/cmdline.html#commands-by-topic`. Most options pertain to files that are either input to the command (i), or output by the command (o). Some inputs and outputs are required, some are optional.

Here, we summarize the most important options for both `grompp` and `mdrun`.. There are default names for the input and output 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.

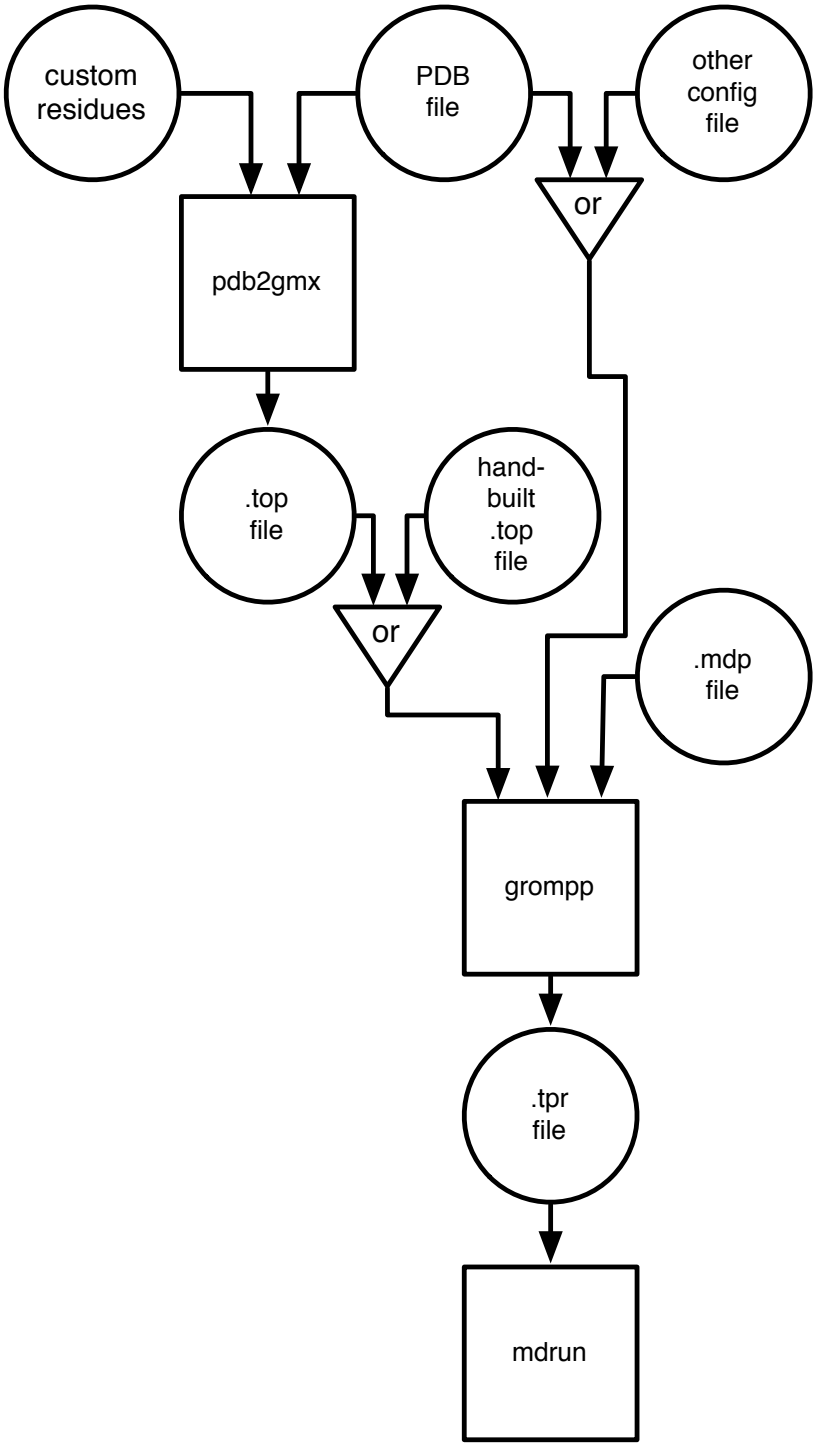


Figure 5: Flowchart of workflow, from configurations to topology files to mdrun.

3 Simulations in more detail

Now that we have seen an example of a simulation, with its progression of equilibration steps, preprocessing and job submission, and subsequent analysis routines, we turn to a discussion of the elements underlying the simulation.

3.1 Simulation units

Gromacs employs the following fundamental and derived units:

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

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

3.2 Types of files

In the example simulations for lysozyme, we encountered many different types of input and output files used by Gromacs. Here is a list of different files used by Gromacs (not all of which we have yet encountered):

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. `.itp` Input ToPology file. The `.top` file is often short, reading in several `.itp` files that specify the topology of each kind of molecule in the simulation.

6. `.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.

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

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

9. `.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".

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

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

3.3 Configuration file

All simulations from scratch start from a configuration file, that gives the initial positions of all the atoms. Configuration files for systems containing many molecules are all ultimately built from single-molecule configurations. Configurations of individual molecules can be downloaded from repositories of molecular structures such as the Protein Data Bank (`http://www.rcsb.org/pdb/home/home.do`). Molecular structures can also be built and visualized using **Avogadro** 1.2, available at `http://avogadro.cc`) for both PC and Mac.

Files from the Protein Data Bank are in PDB format (described below); Avogadro can read and write PDB format among many others. Gromacs can import configuration files in PDB format, but its "native" format is the .gro file. Both .pdb and .gro files are human-readable text files. Less conveniently, both are also "fixed format", meaning that fields on each line must occur in specified column ranges, or the file cannot be read by the computer.

The .pdb format is quite old, dating from the days of 80-column punch cards. Punch cards were originally invented in the 1890s (!), standardized by IBM in the 1920s (decades before the computer age), and used until the early 1980s. The information density on punch cards was minuscule: to store 4MB of data required a stack of cards 30 feet high. And yet, traces of the 80-column format still persist today.

.gro

In a .gro file, the first line describes the file; the second line gives the number of atoms, and each following line describes one atom. In the atom lines, the column widths are: 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 form 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 (commences a list of several molecules), ATOM (information about each atom in a molecule), TER (ends each chain), ENDMDL (ends the model), and END (ends the file).

Typical lines in a PDB file look like this:

```
        1         2         3         4         5         6         7         8
12345678901234567890123456789012345678901234567890123456789012345678901234567890
ATOM      1  N   LYS A   1      35.365  22.342 -11.980  1.00 22.28           N
ATOM      2  CA  LYS A   1      35.892  21.073 -11.427  1.00 21.12           C
ATOM      3  C   LYS A   1      34.741  20.264 -10.844  1.00 16.85           C
```

Figure 6: Example atom lines from a .pdb file. Numbering at top is guide to counting columns, not part of file.

The column ranges for fields in an atom line for a .pdb file are given in the following table.

Columns	Entry	Comment
1 – 6	Record type	`ATOM`
7 – 11	Atom #	integer
13 – 16	Atom name	`C,H,...`
18 – 20	Residue name	
22	chainID	
23 – 26	Residue number	
31 – 38	x position	8.3 format, Angstroms
39 – 46	y position	
47 – 54	z position	

Table 3: Most important entries in PDB fixed format file.

3.4 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.

It is helpful to collect a few typical .mdp files as starting points for creating new .mdp files, which are often different in only a few settings. A typical .mdp file for an MD simulation looks like this:

```
; run control
integrator = md
```

```
dt = 0.001 ; 1 fs
nsteps = 1000000 ; steps, = 1 ns

; output control
nstxout      = 10000 ; steps per position save
nstlog       = 10000 ; steps per log entry
nstenergy    = 100 ; steps per energy file entry

; cutoffs
cutoff-scheme = Verlet
nstlist = 10 ; neighbor list update frequency
ns_type = grid ; neighbor list method (simple, grid)
pbc = xyz  ; periodic boundary conditions
rlist = 1.2 ; short range cutoff
coulombtype = PME ; method for electrostatics
rcoulomb = 1.2 ; Short-range electrostatic cutoff
rvdw = 1.2 ; van der Waals cutoff
DispCorr = EnerPres ; long-distance contributions to E, P

; temperature control
tcoupl         = v-rescale  ; velocity rescaling thermostat
tc_grps       = system ; coupled to entire system
tau_t           = 1.0 ; time constant
ref_t            = 300 ; temperature (K)

; pressure control
pcoupl = berendsen ; barostat type
tau_p = 10.0 ; time constant
ref_p = 1.0 ; pressure (bar)
compressibility = 4.5e-5 ; pressure bath compressibility (of water, bar^-1)
```

The .mdp file is "free form"; parameters can appear in any order. However, for good readability it makes sense to group parameters by topic, as above. Brief comments (prefaced by ";") as to the meaning of different parameters are helpful as well, particularly for inexperienced users.

Timestep. A typical value is 1fs (0.001ps). This can be increased several fs with various tricks (see "Running Faster" in the final section below).

Output control. Parameters `nstxout`, `nstlog`, and `nstenergy` are the number of steps between saves of the configuration in the .trr file, entries in the .log file, and saves of run parameters (energy, pressure, volume, etc.) in the .edr file. To avoid large .trr files, choose `nstxout` with care; here, we take `nstxout nsteps/100`, to save 100 frames in the .trr "movie" (enough to visualize the motion). The .edr file is much smaller, so we can save many entries (here, 10000) for better statistics. When we need to average quantities over the .trr trajectory, we set `nstxout` to save more frames for better statistics.

Temperature control. The `v-rescale` thermostat adjusts kinetic energies up or down by

"rescaling" particle velocities (multiplying all by some factor), to regulate the temperature. `ref_t` is the target temperature in Kelvin. `tau_t` governs how quickly the thermostat tries to settle the temperature. Too short a value can give unstable behavior; too long a value leads to long transients before equilibrating. Values of 1–10 ps work well.

Pressure control. The Berendsen barostat is robust but gives incorrect volume fluctuations; the Parinello-Rahman barostat gives correct fluctuations, but tends to be unstable when used on poorly equilibrated systems. (So use Berendsen to equilibrate, and if volume fluctuations matter then switch to P-R.) `ref_p` is the target pressure in bar. `tau_p` is analogous to `tau_t` for the thermostat; `compressibility` is the compressibility of the "pressure bath". The values given above work well for most cases.

The above .mdp file is rather concise, in part because it relies on default values of parameters not explicitly specified. When `grompp` runs, one of its outputs is a "complete" .mdp file, default name `mdout.mdp`, with absolutely all parameters included. A brief look at `mdout.mdp`, which is thick with parameters and thus not very readable, makes clear the importance of defaults.

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. Beyond this distance, there are residual contributions to the attractive energy and pressure; these can be computed with "tail corrections", using the `EnerPres` .mdp directive.

Verlet cutoff

Since Gromacs 4.6, a simpler and more efficient Verlet cutoff scheme has been introduced (See `http://manual.gromacs.org/online/mdp_opt.html#nl`). This scheme works with PME (particle mesh Ewald) electrostatics. GPU acceleration in Gromacs requires the Verlet scheme. However, Verlet cutoffs are not yet available for all simulation methods and conditions; for example, Verlet cutoffs cannot be used with tabulated interactions. This limits the use of GPU acceleration at present.

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 Verlet cutoffs; `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. The sample .mdp file above uses Verlet cutoffs, and presents typical parameter values.

Group cutoff

Prior to Gromacs 4.6, specifying cutoffs was more complicated. For van der Waals interactions, the choices were `cutoff` (abrupt cutoff; potential is discontinuous), `shift` (shift

potential to zero beyond cutoff; forces are discontinuous), and `switch` (potential is smoothly transitioned to zero). 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, an analogous cutoff schemes `cutoff`, `shift`, and `switch` can be specified. In addition, there is `PME` (particle mesh Ewald). With `PME`, `rcoulomb` is not a cutoff per se, but rather the crossover between treatment of Coulomb interactions in real space and Fourier space. Generally speaking, PME gives a more accurate treatment than a cutoff of slowly-decaying $1/r$ Coulomb interactions.

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 a length equal to the size of the largest charge groups.

If we use PME, hence `rcoulomb` equal to `rlist`, and `cutoff` van der Waals, 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 that `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 20Å or 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.

3.5 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 the .top file we constructed for the lysozyme example (for more details on .top files see Gromacs manual 5.7.2):

```
#include "oplsaa.ff/forcefield.itp"
#include "oplsaa.ff/spce.itp"
#include "oplsaa.ff/ions.itp"
#include "1AKI.itp"

[system]
1AKI in water

[molecules]
; name       count
1AKI          1
SOL           10636
CL              8
```

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); `oplsaa.ff/spce.itp` and `oplsaa/ff/ions.itp` are the .itp files for the SPC/E water model and for ions like Cl^- present in the simulation. Finally, `1AKI.itp` is the molecule .itp file we made using `pdb2gmx`. The remaining declarations are the name of the system, and the number of each type of molecules.

3.6 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, which includes both bonded and nonbonded interactions. Bonded interactions describe how the energy of a single molecule varies with shape, as defined by the bond lengths, bond angles, and dihedral angles of the molecule.

For example, a bond between two atoms will have a preferred bond length b. The potential around this length can be approximated as harmonic:

$$U = \frac{1}{2}k^b(r-b)^2 \tag{1}$$

where r is the separation between the atoms, and b the preferred bond length. The force is then given by $F = -\partial U/\partial r$.

Non-bonded interactions act between atoms that are not bonded, either within the same molecule or between different molecules. Non-bonded interactions are of three types: short-range repulsive, dispersion (van der Waals), and electrostatic (between charges).

Short-range repulsive and dispersion forces are described by the Lennard-Jones potential

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

where r_{ij} is the separation between atoms i and j. Here the first term represents the repulsions between atoms as they begin to overlap, and the second term represents attractive dispersion forces. The electrostatic interaction between atoms i and j is given by the Coulomb potential,

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

in which q_i and q_j are the charges of the atoms.

The parameters that appear in these interactions for the different types of atoms present in complex molecules are defined by a "force field", such as OPLS-AA (Optimized Parameters for Liquid Simulation, All-Atom). Here, different types of atoms means not just the element itself (C, N, O, ...), but the local chemical environment of that atom bonded to its neighbors (C in -CH_2-, C in -CH_3, C in -CH=CH-, ...).

How the force field organizes these parameters, as well as how they are determined and validated, is described below.

forcefield.itp

Similarly to the .top file, the file `forcefield.itp` for a given potential is defined "top down". `forcefield.itp` contains only a single definition and three `#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 chooses the non-bonded potential function (type 1 = LJ, type 2 = Buckingham), and the "combining rule" for nonbonded parameters between different atom types. (For details on combining rule, see Gromacs manual 5.3.2; comb-rule 3 is the "sigma-epsilon" system.)

The first two `#include` files contain nonbonded and bonded declarations for all atom types. The file `ffnonbonded.itp` (see Gromacs manual 5.2.1) contains [atomtypes] declarations for all atoms:

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

Here "name" is the atomtype name (from `atomtypes.atp`). "ptype" is the particle type (A = atom, V = virtual site). The last two parameters are LJ parameters, the meaning of which depends on the default form of the nonbonded potential, set by `nbfunc` in `forcefield.itp`.

The "bondtype" is a more general name, that declares how the bonded potentials are set by entries in `ffbonded.itp`. Many different but related atom types share the same bond type, which helps to reduce the potentially vast number of bonded interactions between different molecules. For example, the saturated hydrocarbon C with different numbers of bonded H (`opls_136` through `opls_139`) all have bonding type CT.

Likewise, the file `ffbonded.itp` (Gromacs manual 5.3.3) specifies parameters for 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]
...
```

3.7 Molecule .itp file

To keep the topology files neat, each kind of molecule should have its own .itp file. A molecule .itp file contains the type of each atom in the molecule, and the set of bonds, angles, and dihedrals to be included in the intramolecular potentials.

The 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 must match one of the types in the `atomtypes.itp` file of the potential you are using, unless you are defining custom atom types (see below).

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 force field (in the file `ffbonded.itp`). 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 force field (e.g., OPLS-AA).

Methods for building

A molecule `.itp` file contains all the bonds, angles, and dihedrals that contribute to the intramolecular potential. Constructing .itp files for polymers and other large molecules is a challenging and tedious task.

Fortunately, there are several useful tools to help automate this task. There are four basic approaches to build topology files:

1. *By hand.* This is tractable for small molecules, or very simple polymers such as a linear bead-spring chain, because of the very simple structure (bead n is bonded to bead $n+1$; angle interactions if present connect beads n, $n+1$, and $n+2$).

2. *Use a web-based tool.* A useful website exists that takes a `.pdb` configuration file as input, and (sometimes) returns a usable `.itp` file [`\http://erg.biophys.msu.ru/tpp/`].

This tool assigns atom types and builds the bonds, angles, and dihedrals from the molecular structure. Unfortunately, it fails to produce useful output if the OPLS force field does not include all the parameters for bonded forces between the given atoms.

3. *Use "residues" and* `pdb2gmx`. Proteins are polymers, with 23 different kinds of "residues" (monomers). Molecular biologists have simulated proteins for a long time, and developed automated tools to build the topology files for these molecules. In Gromacs, this tool is `pdb2gmx`, which takes advantage of the fact that the topology of the residues is relatively simple to specify, and the topology of the entire protein is determined by the sequence of residues. We can use this tool to create topology files for polymers, by defining new residues that correspond to our monomers (described in detail below).

4. *Use* `topology utilities.nb`. The set of bonds, angles, and dihedrals for a given molecule are completely determined by the list of bonded neighbors. (The list of bonded neighbors is not painful to generate by hand for even large molecules, and anyhow can be generated automatically for `.pdb` files from the distances between atoms.) It is simply a bookkeeping problem to list all the sets of three consecutive neighbors that contribute an angular interaction, and all the sets of four nonrepeating consecutive neighbors that contribute a dihedral interaction. Our group has written a set of Mathematica routines that generate the list of angles and dihedrals from the list of bonded neighbors.

Choosing atom types.

Generating the structure of the molecule .itp file is only half the battle. If the .itp file is built by hand (or using `topology utilities.nb`), or if residues are invented for monomers to use with `pdb2gmx`, atom types for every atom in the molecule or residue must be chosen.

When using the OPLS-AA force field, the first step in choosing atom types is to scan through `atomtypes.atp`, which lists the many atom types with brief comments as to their chemical nature. Another route to choosing atom types is to examine small molecules in the catalog of topology files archived at `http://virtualchemistry.org`, looking for analogous structures to see what atoms are used.

Comprehensive as it is, the OPLS-AA force field does not contain all possible atom types. And even if appropriate entries are found in the list of existing OPLS-AA atom types, sometimes not all bond, angle, and dihedral interactions present in a given molecule are listed in `ffbonded.itp`. The number of possible bonded interactions grows rapidly in the progression from bonds to angles to dihedrals with the increase in number of participating atoms, so it is not surprising that some bonded interactions are missing. When appropriate atom types cannot be found, or bonded interactions are missing, the user must supply the missing parameters in an "interaction" .itp file, that serves to augment the definitions in `opls-aa/forcefield.itp`.

Modified potentials

Determining missing force field parameters is a complex task. Generally speaking, bonded interactions and atomic "partial charges" can be computed using modern quantum chemistry

methods (density functional theory). Repulsive interactions between atoms can likewise be fitted to DFT results. But attractive dispersion interactions are beyond the capability of DFT methods, and must be fitted to experimental results for liquid densities and boiling points.

Generally, we use potential parameters from OPLS as much as possible. Sometimes, we want to override the default values for a given atom type. The most common reason for doing this is to adjust atomic charges to better represent the charge distribution of a given molecule. (Partial charges can be obtained by electrostatic fitting to quantum chemistry calculations performed in a package like Gaussian.)

Or, we may need to supply bonded potentials, when the OPLS force field does not contain definitions for some particular bonds, angles, or dihedrals in a given molecule. Finally, we may need to create atom types for new situations, possibly by modifying existing types.

There are two ways to modify or extend the force field. The simplest is to modify the molecule .itp file. If we use only existing OPLS atom types, we can override atomic charges in the `[ atoms ]` section. We can likewise override any bonded interaction or supply any missing bonded parameters in the `[ bonds ]`, `[ angles ]`, or `[ dihedrals ]` sections, instead of relying on `grompp` to look them up from the OPLS force field.

We cannot override the Lennard-Jones parameters for given atom types in this way, because there is no place to include them in the `[ atoms ]` section. Instead, we must use the more general approach to modifying the force field, which is to include in the .top file an "addendum" .itp file, immediately after the force field is read in:

```
#include "oplss-aa.ff/forcefield.itp"
#include "additions.itp"
```

The addendum .itp file (here `additions.itp`) can contain additional `[ atomtypes ]`, `[ bondtypes ]`, `[ angletypes ]`, or `[ dihedraltypes ]` sections. These can redefine existing atom types or add new types, and can likewise override existing or add new bonded interactions. Such addendum .itp files must appear *before* any molecule .itp files are read in. Once molecule .itp declarations (`[ atoms ]` etc.) have begun, no more forcefield declarations (`[ atomtypes ]` etc.) can be read.

Finally, we can also override the nonbonded interactions *between* specific atomtypes, i.e., overriding the "combination rules" used by default to set those interactions, by including in an addendum .itp file 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 OPLS `atomtypes.atp` file, or to correspond to some fictitious "model atom"; we do this with an addendum .itp file. First, we add an `[ atomtype ]` declaration those in `ffnonbonded.itp`, which gives the atom type name, "bonding type", mass, charge, and LJ parameters. Then we add in `[ bondtype ]`, `[ angletype ]`, and `[ dihedraltype ]` sections any missing bonded interactions. If we use an existing bonding type, we can make use of bonded interactions defined for that type, already present in `ffbonded.itp`.

4 Building a decane liquid

In this example, we simulate a liquid of decane. Setting up such a simulation requires us to 1) build a molecule .itp file for a large molecule, and 2) construct a reasonable initial configuration for the liquid.

For the first task, we demonstrate an approach that can be applied more generally to polymer chains. First, we use the molecular visualization / drawing software Avogadro (available at `http://avogadro.cc`) to create a .pdb file for a single decane molecule in an energy-minimized all-trans configuration (we will use this file also for building the initial configuration, below). Then, we define new residues to build alkane chains, and adapt the Gromacs utility `pdb2gmx` to generate the molecule .itp file. To use `pdb2gmx`, we must reorder and relabel the .pdb file to match the order and name conventions of the residue entries in the .rtp file.

For the second task, we use the simulation itself to construct a reasonable initial configuration for the decane liquid, starting from a configuration that is not at all realistic but at least easy to build. We start from regular array of all-trans chains, with sufficient space between them to ensure the chains do not overlap and have plenty of room to wiggle around when the simulation starts. Then, we make use of the .mdp option `-deform`, which allows us to shrink the system dimensions at a constant rate. We choose the rate to reach the correct density after some few nanoseconds, during which time the decane molecules can randomize and interpenetrate, achieving a liquid-like packing.

4.1 Create new residues

In the topology folder `top/oplsaa.ff`, the residues for amino acids are contained in the file `aminoacids.rtp`. We will add entries for our new residues to this file. Each residue contains two sections: `[ atoms ]` and `[ bonds ]`. Some more complicated residues also contain sections for overriding dihedrals and for setting improper dihedrals; we will not need these for moieties to build alkanes.

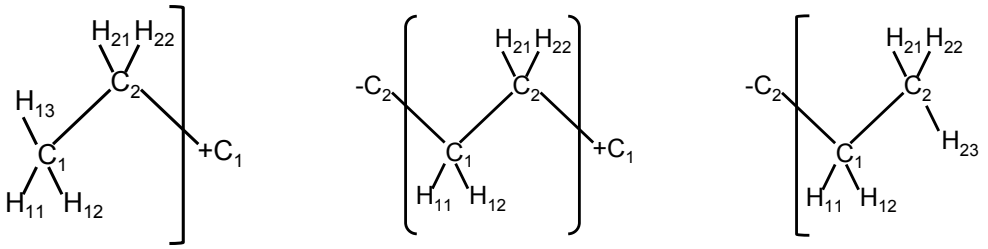
The `[ atoms ]` gives each atom in the residue a name, a type, and a partial charge. It also classifies each atom into a **charge group**. A charge group is (loosely) defined as the group that, when all the partial charges are added together, combine to some integer net charge (usually zero). An example of a simple charge group is a CH_2 in a long alkane, 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. Duplicate entries are not required: A B is equivalent to B A, and should only be listed once. If a given atom bonds to the previous (next) residue, place a `-` (`+`) before the bonded atom.

[atoms]

To make residues for decane, we treat it as a short piece of polyethylene. The ethylene monomers are shown in Figure 7. We need separate monomers for the beginning and end of the chain; for example, the beginning monomer has an extra hydrogen, and does not bond to the previous residue. We name each residue with a three-letter name: PolyEthylene

Beginning (PEB), PolyEthylene Middle (PEM), and PolyEthylene Terminal (PET). (All residue names in `aminoacids.rtp` must be unique.) FIgure 7 includes our naming convention for the atoms in each residue; the carbon atoms are C1 and C2, and their bonded hydrogens H11, H12, and so forth.



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

Figure 7: Naming schemes for polyethylene monomers

We must identify an OPLS atom type for atom, and get the information for the `[ atoms ]` section of each residue.

1. Open `atomtypes.atp`; atoms near `opls_135` are for alkanes.
2. Classify each atom type in each monomer. You will use `opls_135`, `opls_136`, and `opls_140`.
3. Open `ffnonbonded.itp` and find the partial charges for these atoms.

Check that each charge group (there are two in each monomer) is net neutral.

Create a new text file `PE.rtp` in which to create the new residues; later, we will append them to `aminoacids.rtp`. The residue for the middle monomer (PEM) 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 for the PEB and PET residues.

[bonds]

Now, we list the bonds, starting with the middle monomer (PEM). 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, the bond section for PEM should look like:

```
...
H22    opls_140      0.060     2

[ bonds ]
C1     -C2
C1     H11
C1     H12
C1     C2
C2     H21
C2     H22
C1     +C1
```

Repeat for PEB and PET, and add these bond sections. Compare your result to the `PE.rtp` file included with this tutorial.

Once you are sure that your residues are correct, Copy and paste your residues at the end of `aminoacids.rtp`. Finally, append the new residue names to the file `residuetypes.dat` in your `/top` directory:

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

4.2 Make a PDB configuration file

We use Avogadro to make a .pdb configuration file for decane, in the energy minimized all-trans configurations. To draw the molecule, select the pencil tool; click to form your first carbon (it will appear as CH_4, with hydrogens automatically added). Click on a hydrogen to add another carbon (you can also click on a carbon and drag to form a C-C bond, with hydrogens automatically adjusted). All-trans decane looks like a zig-zag from the side, so draw the molecule this way, either by choosing the correct H to add the next C, or by dragging to form the next C-C bond in the right direction.

Once you have placed ten carbon atoms, click on the blue `Navigation Tool` in the tool bar. You can now drag your screen and inspect your molecule. Use the energy minimization tool to clean up the structure (click `E` in the tool bar, and `Start` to minimize).

Next, align your molecule along the z-direction. To do this, select the alignment tool (two parallel lines icon); select the 1st and 9th carbon, choose "z axis" and "Align Everything", then click "Align". This places the first atom at the origin, and the second along the z axis (this alignment will be useful when we use the .pdb file to build the initial configuration). Save your file as `decane_avogadro.pdb`. Then copy this file as `decane.pdb`, in which we will reorder and relabel the atoms.

Atom renaming

Avogadro does not know the atom or residue names we have chosen, so we must modify `decane.pdb` using BBEdit.

Delete the `CONECT` records and `MASTER` record, which are not needed for Gromacs. Replace all occurrences of `HETATM` with `ATOM__` ("ATOM" followed by two 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). Make sure to *preserve the spacing* when making modifications.

Column 2 are the atom numbers; we will number these sequentially after we reorder the atoms to correspond to the order in our residues. Initially, the atoms are numbered in an order set by Avogadro. To find out which atom is which, 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.) Sometimes the labels do not work; another way to find out which atom is which, is to select View/Properties/Atom Properties... from the top menu, and click on atoms in the first column; the corresponding atom in the picture will be highlighted.

We want the atoms to appear in the .pdb file "residue by residue", and within each residue in the order specified in the .rtp file. Starting from one end of the chain in Avogadro, find the numbers of the atoms in the first residue; moves these lines to the top of the .pdb file, and place them in the proper order. Change the atom names (third column entries) to match those used in the .rtp file (C1, H11, H12, and so forth). It does not matter which H is H11 and which is H12 on a given residue, since both are structurally equivalent, bonded to the same C1. As you change names, remember to maintain the column widths and positions in the .pdb file. Repeat this process residue by residue along the chain, using Avogadro as a guide to identify which atoms go where in the modified .pdb file.

Once all atoms are in order, organized into residues, and renamed, replace the generic residue `LIG` in Column 4 with the proper residue name, either `PEB`, `PEM`, or `PET`. To preserve the spacing, replace the three characters in `LIG` with the three-letter residue name.

Column 5 contains the residue numbers, which initially are all 1. These should be $1, 2, 3, \ldots$ in sequence by residue. Finally, the atom numbers in Column 2 should be renumbered in order starting from 1. Again, be sure to preserve the column locations when renumbering.

To check that your .pdb file is correctly formatted, open the unedited `decane_avogadro.pdb` in BBEdit, copy one of its atom lines, and paste it before the first atom line in your edited PDB file. Check that the columns line up correctly (and then delete the added line). Compare your file with `decane.pdb`, included in this tutorial.

4.3 Prepare and run simulation

We carry out the following steps to prepare and run a simulation of a decane liquid:

- use `pdb2gmx` to prepare the molecule .itp file for decane;
- build the initial configuration of all-trans decane moleclues in an ordered array;
- compute the deformation rate needed to shrink the system to the correct density;
- create the .mdp and .sh files by modifying files from the lysozyme example;
- submit and examine the equilibration job.

To begin, create a new folder on the cluster called `decane`, and upload `decane.pdb`. Use `pdb2gmx` to create a topology file `decane.top`:

```
gmx pdb2gmx -f decane.pdb -o decane.gro -p decane.top
```

(when prompted, enter 16 for the OPLS-AA force field, and then 8 not to include any water).

`pdb2gmx` produces a complete .top file; we only want the molecule definition. So make a copy named `decane.itp`, and edit it to remove the beginning and ending material, as was done in the lysozyme tutorial. Change the `[ moleculename ]` to "decane".

The system topology file `system.top` can be readily adapted from the lysozyme example, as:

```
#include "oplsaa.ff/forcefield.itp"
#include "decane.itp"

[system]
decane liquid

[molecules]
; name count
decane 100
```

Use `editconf` to create `decane.gro` with the appropriate dimensions, with the molecule centered in the box. Then, use `genconf` to create a system consisting of a 10 x 10 x 1 array of boxes, each containing one all-trans decane:

```
gmx editconf -f decane.gro -box 0.4 0.4 4.0 -o decane.gro
gmx genconf -f decane.gro -nbox 10 10 1 -nmolat 32 -o system.gro
```

Here `-nmolat` specifies the number of atoms per molecule (32 for decane, $C_{10}H_{22}$).

The initial system size of the regular array (a cube 4 nm on a side) is much larger than the corresponding decane liquid. We compute the final liquid volume from the density of decane (0.730 g/cm^3), the molar mass (142.29 g/mol), and the number of molecules (100), as equal to a cube 3.187 nm on a side. To reach the final system size, We deform the linear dimensions of the system in 1 ns (= 1000 ps) from 4 nm to 3.187 nm, at a rate of -0.000813 nm/ps. The corresponding .mdp option is

```
# resize from 4^3 to 3.187^3 in 1ns
deform = -0.000813 -0.000813 -0.000813 0. 0. 0.
```

(The last three `deform` values, all zero, are shear deformations, not used here.)

We can build the .mdp file `resize.mdp` by adapting the file `md.mdp` from the lysozyme example, with the following changes:

- no pressure coupling (hence like `nvt.mdp`)
- not a continuation; generate velocities (again like `nvt.mdp`)

- constrain the lengths of only h-bonds, not all-bonds (constraining all bond lengths does not work well for long-chain molecules)
- correspondingly, reduce the timestep to 1 fs

Compare your result to the file `resize.mdp`, included with this tutorial.

Running `grompp` for this simulation is a straightforward adaptation of the lysozyme example, taking account of the names of the various files:

```
gmx grompp -f resize.mdp -c system.gro -p system.top -o resize.tpr
```

Correspondingly, the job script `resize.sh` (submitted as usual with `qsub resize.sh`) can be based on `md.sh` from the lysozyme example, with the appropriate name change (replacing `md` with `resize`). Also, since we only have 100 chains of 32 atoms totaling 3200 atoms, two cores is plenty for the job.

After the resize run has completed, visualize the trajectory using VMD, to verify that the regular array of all-trans decane chains has been compressed to liquid density, and the chains have randomized their configurations.

5 Batch job submission

During our lysozyme simulation, we submitted three batch jobs to to run on ACI-b. All other tasks were run interactively; that is, we executed them in a terminal session, and waited for output to appear on the 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. Jobs are submitted 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 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 Batch scripts

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 `.sh`. The file name should remind you something about the job; for example, `md.sh`, which runs an MD simulation. Choose filenames wisely.

A typical batch script may perform several simulations in succession. For example, we can automate the entire sequence of lysozyme equilibration followed by the production MD run, as follows:

```
#PBS -A <accountName>
#PBS -l nodes=1:ppn=4
#PBS -l walltime=24:00:00
#PBS -j oe

cd $PBS_O_WORKDIR

# energy minimization
gmx grompp -f em.mdp -c 1AKI_solv_ions.gro -p system.top -o em.tpr
gmx mdrun -deffnm em

# NVT equilibration
gmx grompp -f nvt.mdp -c em.gro -p system.top -o nvt.tpr
gmx mdrun -nt 4 -deffnm nvt

# NPT equilibration
gmx grompp -f npt.mdp -t nvt.cpt -p system.top -o npt.tpr
```

```
gmx mdrun -nt 4 -deffnm npt

# production MD
gmx grompp -f md.mdp -t npt.cpt -p system.top -o md.tpr
gmx mdrun -nt 4 -deffnm md
```

The first line gives the queue (account, `-A`) to access (an example is `stm9_b_t_bc_default`, not currently active). Ask your PI for the queue where you should submit your jobs. Queues with the designation "sc" run on "standard" nodes, and those with designation "bc" run on "basic" nodes. Jobs that exceed the confines of the queue can run in "burst" mode, which may wait longer in queue.

One possibility is the "open" queue, (`-A open`, running on basic nodes), which offers extra computing cycles for free. However, if your job is long, it may be "suspended", preempted by a job from someone who "owns" the node on which you are running. If this happens, your job waits until the preempting job finishes, which may take several days to a week.

The next line requests nodes and processors (cores) per node; `odes=1:pp`=4* means 4 cores on one node. "Nodes" are collections of processors that share common memory. "Basic" nodes on ACI-b have two CPU chips with 12 cores each, so 24 cores per node ("standard" nodes have two CPUs with 10 cores each, or 20 cores per node). As a general rule, a job should have a few thousand 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 more detail below. Here, the cores are used to run `mdrun` with four "threads", indicated by the option `-nt 4`, which divides the simulation work among the 4 requested cores.

The third line sets the "wall time" for the process — the maximum time 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 they must be broken into a succession of jobs, each of which starts up from the checkpoint file of the last job (see below).

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.sh.o12345`.

After this, we 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, PBS scripts are executed from your home directory.)

5.2 Checkpoint files

Some jobs take longer than the maximum run time (typically 1 week, or 168 hours). To run jobs that take longer, we restart from checkpoints. A checkpoint (`.cpt`) file with complete state of the system is written by default every 15 minutes (unless overridden by option `-cpt`).

A simulation can be continued by reading the full state from the checkpoint file by calling `mdrun` with option `-cpi <filename>`. If no checkpoint file is found, Gromacs assumes a normal run, and starts from the first step of the `.tpr` file. By default, outputs will append to existing files; with `-noappend` new output files are opened and the simulation part number is added to all output file names.

5.3 Timing studies

Before launching a big job that will take a long time to run, it is a good idea to do a timing study. By running the same job with a much shorter total time, the speed of the job (in simulated nanoseconds per day) can be measured, and the total execution time predicted.

A short test job also helps ensure the big job will run without errors and perform as expected. Usually, if a simulation job is going to go awry, the error will appear either in the preprocessing step, or immediately on starting the simulation (because some input files or options are incorrect), or shortly after the simulation begins (because the simulated system misbehaves in some way, as a result of a bad initial configuration, poor equilibration, too large a timestep, or problems with the pressure control).

5.4 PBS commands

The PBS system has its own language of commands. Most commonly used are:

- To submit a batch job, `qsub <scriptname>`.
- To check the status of your batch jobs, `qstat -u <userid>`.
- To check the status of a specific job, `qstat -n <job_id>` where `<job_id>` is the number of the running job (obtained from `qstat -u`).
- To delete a specific job, `qdel <job_id>`.
- To delete all your jobs (hopefully you will never need this!), `qdel $(qselect -u <userid>`.

5.5 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` 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 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 some clusters (e.g., the now-retired Lion-X clusters), 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. In contrast, the queue on ACI-b prioritizes access to resources owned by a given PI, at the expense of leaving cores idle.

So on now-defunct 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 makes sense to do the opposite — to request 10 cores and 1 node, so as to stay within our queue resources. This remains true even for submitting to the "open" queue.

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`.

To efficiently use a single node to run multiple jobs, we should consider using a number of cores for each job that corresponds to an integer number of quarter nodes. In this way, multiple jobs can run alongside each other without cores sitting idle.

On ACI-b, "basic" nodes have two CPUs with 12 cores each, hence 24 cores per node; "standard" nodes have two CPUS with 10 cores each, so 20 cores per node. So in a queue running on the basic nodes, we should aim to run jobs on 6, 12, 18, or 24 cores; on the standard nodes, jobs should be sized for 5, 10, 15, or 20 cores.

5.6 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.the

ACI has a few nodes with GPU support, but they are priced very expensively compared to the basic nodes, on a basis of dollars per nanosecond of simulation time. As a result, the Milner group does not own any such nodes. ACI does not at present provide any access to GPU nodes in the open queue.

However, another cluster at Penn State with GPU-enabled nodes that does permit open access is Cyberlamp, once permission is granted by its owner group. Cyberlamp shares the same file system as ACI, so running on Cyberlamp is simple.

To run on 8 cores with 1 GPU, in the job script include the lines

```
#PBS -A cyberlamp
#PBS -l nodes=1:ppn=8:gpus=1
```

and for the `mdrun` command, use

```
export OMP_NUM_THREADS=8
...
gmx mdrun -nt 8 -gpu_id 0 -deffnm <jobname>
```

Here option `-nt 8` requests 8 "threads" (cores), and option `-gpu_id 0` requests GPU "number 0" (i.e., the first one you requested, numbering starting from 0).

Milner group cluster and workstation

The Milner research group owns two computing resources, both supplied with GPU support, available for group members. The first of these is a workstation called "closet" (it was once in a telecom closet), consisting of a single node with 28 cores and one K80 GPU (with two logical GPUs). The second is a small cluster called "nccise" (Next-generation Cluster for Computationally Intensive Science and Engineering — pronounced "incise", like incisive, cutting-edge ...). The nccise cluster consists of 9 nodes, each with 32 cores and one K80 GPU.

Closet and nccise are only accessible from the local LAN. When off campus, the local LAN can be accessed by logging onto a VPN using Cisco AnyConnect Secure Mobility Client, available at `https://downloads.its.psu.edu/#` under "Connecting to Penn State". Once installed, connect to `vpn.its.psu.edu` and join the group "ISPtoPSU". Then, to logon to closet or nccise from your computer, use

```
ssh -Y <userid>@closet.dc.psu.edu
ssh -Y <userid>@nccise.dc.psu.edu
```

Closet and nccise both have a single batch queue called "batch", which works like ACI except that instead of the PBS directive `#PBS -A <account>`, use `PBS -q batch`. Please be

courteous in using our cluster — no one actively monitors it to make sure jobs are correctly sized, killed if they get stuck, etc.

To run with 8 cores and 1 GPU on closet or nccise, use the PBS directive

```
#PBS -l nodes=1:ppn=8:gpus=1
```

with the command

```
gmx mdrun -nt 8 -gpu_0 -deffnm <jobname>
```

Because nccise nodes have 32 cores and the closet workstation has 28 cores, it is good practice to size jobs in multiples of 4 cores (e.g., on nccise, jobs of 4, 8, 16, 32 cores; on closet, 4, 8, 16, 24 cores) so that nodes have a chance to completely fill.

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 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`

6.2 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 for bonds involving hydrogen atoms, timesteps can often be boosted from the usual 1 fs to more like 2 fs. (We constrain these bonds especially, because hydrogen atoms are light, and so have high "thermal velocity" compared to all other atoms.) 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 = h-bonds
constraint_algorithm = lincs
```

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

In general, it is not a good idea to constrain the lengths of bonds along the backbone of a polymer. As the polymer explores multiple configurations, satisfying all the length constraints is a numerically challenging problem best avoided.

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 2 fs with only bond constraints, to 4 fs 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 are 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 are simulating a glassy fluid, things will be slow near the glass transition; maybe you need to raise the temperature. If you are 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. (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.

6.3 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. The wall density in the `.mdp` file must match the wall density that you choose.

6.4 Building polymer .pdb files

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 a different species, requiring its own topology file. Another example are atactic polymers, in which sidegroups on each monomer are randomly on one side or the other of the all-trans chain. A single molecule .itp file works for all such chains, but we must still generate many different .pdb files to represent the system. Either way, we have an unpleasantly large number of .pdb files to generate and clean up.

In such cases, we can use scripts to help us build .pdb files for long chains, which can then be processed by `pdb2gmx` and used to construct initial configurations. The basic idea is to prepare .pdb files for the constituent monomers, in which the atoms have been properly ordered and labeled to match our residue definitions. The script (see below) then uses `insert-molecules` to assemble the monomers together into a single chain.

To generate .pdb files for the monomers, proceed as follows. First, use Avogadro to draw the monomer in the all-trans state, including methyl groups at the attachment points for

the "previous" and "next" monomers. Energy minimize the structure, then align the chain so that it points along the z axis, with the zig-zag of the all-trans bonds in the x-z plane.

Aligning the zig-zag in the x-z plane can be tricky. For a chain like polyethylene, this can be done by aligning two H atoms in a CH_2 group along the y axis. Alignment along z can be then performed by selecting as the atoms to align 1) the monomer first carbon C1, and 2) the terminal methyl carbon C*, which is structurally equivalent to C1. This will place C1 at the origin, and place C* along z. Measure and record the distance from C1 to C*.

Once the alignment has been performed, delete the initial and terminal methyl groups, and save the monomer .pdb file. Now edit the file, in much the same way as for decane; remove the CONECT statements, reorder and renumber the atoms to correspond to the residue definition, and replace the atom and residue names.

Finally, add a comment line at the top of the .pdb file, which we use to record the monomer "offset vector":

```
REMARK 999  0.   0.000   z.zzz
```

where `z.zzz` is replaced by the distance along z from C1 to C*.

Now we use a script `buildChain.sh` to automate the process of using `gmx insert-molecules` to add monomer .pdb files to a configuration, which will build up a chain. `buildChain.sh` is written in the "language of the shell", that we use whenever we type at the command line. Shell scripting is convenient, because much of the script is ordinary commands.

```
#!/bin/bash
# args are xBox, yBox, zBox

# insert first monomer (read from stdin)
read monomer
cp $monomer.pdb newChain.pdb
read a b dx dy dz < $monomer.pdb
zOff=‘echo "scale=4;$dz/10" | bc -l‘
echo "0. 0. $zOff" > pos.dat

# loop over monomers
while read monomer
do
        mv newChain.pdb oldChain.pdb
        gmx insert-molecules -f oldChain.pdb -ci $monomer.pdb -ip pos.dat \
           -rot none -scale 0. -o newChain.pdb -box $1 $2 $3 &> insert.log
        read a b dx dy dz < $monomer.pdb
        zOff=‘echo "scale=4;$zOff + $dz/10" | bc -l‘
        echo "0. 0. $zOff" > pos.dat
done
```

Shell scripting in general is beyond the scope of this document; here we present some comments on the script above.

`buildChain.sh` takes three inputs, which are the dimensions of the box containing the chain. Script arguments are accessed with `$1`, `$2` and so forth (for the 1st, 2nd,

... arguments). (In general, `$<variable>` means "the value of ¡variable¿".) This script reads a list of monomer names from "standard input" (stdin), which by default means typing from the command line, but can be supplied from any file by "input redirection":

```
buildChain.sh 1. 1. 20. < monFile.txt
```

This syntax avoids the need for an "open and save" file dialog box to specify the input file.

The script produces a file `newChain.pdb`, by copying the first monomer .pdb file to `newChain.pdb`, and then looping over the monomer names read from stdin, adding each one to the growing chain with `insert-molecules`. Before each monomer addition, the file `newChain.pdb` is renamed (with `mv`) to `oldChain.pdb`, and the new monomer added to `oldChain.pdb` to give `newChain.pdb`.

The top line of each monomer .pdb file is read to extract the "offset" value, which is added to the total z offset `zOff`. The total offset is written (using `echo`) to a file `pos.dat`, and used by `insert-molecules` to place the next monomer.

One shortcoming of shell scripting is that floating-point arithmetic is awkward. `zOff` is updated by adding the next offset increment `dz`, but we cannot simply write "zOff=$zOff$+dz". Instead, we use the "basic calculator" `bc` (with option `-l`, which loads a "math library"). To use `bc`, use `echo` to "type in the commands", much as we would do on an actual calculator. The commands are "piped into" `bc` using `|`. Finally, the output of the entire operation is saved into the variable `zOff` by enclosing the command string in "backquotes" (`` `...` ``).

6.5 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.

6.6 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.

6.7 Pulling

You can use Gromacs to "pull" on certain groups of atoms. This can be useful for many purposes; one common application is "umbrella sampling", used to map out the potential of mean force of some coordinate. In umbrella sampling, a system is biased into into unlikely configurations, by applying a harmonic "umbrella" potential to some coordinate. The tendency of the coordinate to move away from the bottom of the umbrella is a measure of the mean force acting on it. Another common setup is to constrain some coordinate, and measure the force required to hold the coordinate at the specified value. Still another option is to apply a fixed force to a coordinate, and measure the resulting average displacement.

The groups of atoms to be pulled on are specified using an index (`.ndx`) file, which are built using the interactive utility `make_ndx`. The options for what kind of pulling to do, what groups to pull on and how to pull on them, are given in the `.mdp` file. Numerous options are available, and are documented in the Gromacs manual. The options changed completely from Gromacs 4 to 5, and again from 5.0.7 to 5.1.4, so be sure to look at the correct help file for the version you are using. For 5.1.4, the full set of mdp options is well documented online at `\http://manual.gromacs.org/documentation/5.1/user-guide/mdp-options.html`.

Here is an example `.mdp` file for 1D constant force pulling.

```
; pull code
pull          = yes
pull-ngroups = 2
pull-ncoords = 1
pull-group1-name   = Cleft
pull-group2-name   = Cright
```

```
pull-coord1-type = constant-force
pull-coord1-geometry = distance
pull-coord1-dim = N N Y
pull-coord1-k = -1500 ; -force in kJ/mol/nm
```