

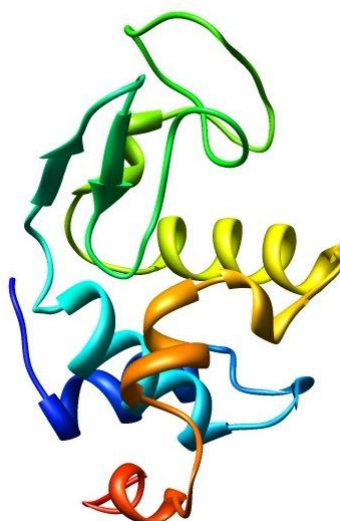
# GROMACS Tutorial

## Lysozyme in water

Based on the [tutorial](#) created by Justin A. Lemkul, Ph.D.

*Department of Pharmaceutical Sciences  
University of Maryland, Baltimore*

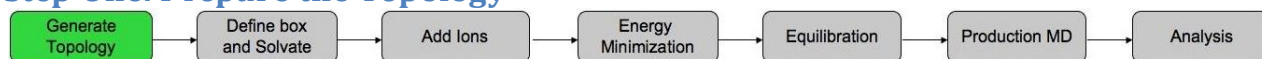
*Adapted by Atte Sillanpää, CSC - IT Center for Science Ltd.*



This example will guide a new user through the process of setting up a simulation system containing a protein (lysozyme) in a box of water, with ions. Each step will contain an explanation of input and output, using typical settings for general use.

This tutorial assumes you are using a GROMACS version in the 2018 series.

### Step One: Prepare the Topology



### Some GROMACS Basics

With the release of version 5.0 of GROMACS, all of the tools are essentially modules of a binary named "**gmx**". This is a departure from previous versions, wherein each of the tools was invoked as its own command. To get help information about any GROMACS module, you can invoke either of the following commands:

```
gmx help (module)
```

or

```
gmx (module) -h
```

where `(module)` is replaced by the actual name of the command you're trying to issue. Information will be printed to the terminal, including available algorithms, options, required file formats, known bugs and limitations, etc. For new users of GROMACS, invoking the help information for common commands is a great way to learn about what each command can do. Now, on to the fun stuff!

## Lysozyme Tutorial

We must download the protein structure file with which we will be working. For this tutorial, we will utilize hen egg white lysozyme (PDB code 1AKI). Go to the [RCSB](#) website and download the PDB text for the crystal structure.

Once you have downloaded the structure, you can visualize the structure using a viewing program such as **VMD**, **Chimera**, **PyMOL**, etc. (On Taito, load the VMD module: `module load vmd`, on the local workstations, it is already in the path):

```
vmd 1AKI.pdb (or laki.pdb check your filename)
```

or if you're running this tutorial in [Taito-GPU](#) always prepend `vmd` with `vglrun`:

```
module load vgl  
vglrun vmd 1AKI.pdb
```

Once you've had a look at the molecule, you are going to want to strip out the crystal waters. Use a plain text editor like **nano**, **vi**, **emacs** (Linux/Mac), or Notepad (Windows). Do not use word processing software! Delete the lines corresponding to these molecules (residue "HOH" in the PDB file). Note that such a procedure is not universally appropriate (*i.e.*, the case of a bound active site water molecule). For our intentions here, we do not need crystal water.

Always check your `.pdb` file for entries listed under the comment **MISSING**, as these entries indicate either atoms or whole residues that are not present in the crystal structure. Terminal regions may be absent, and may not present a problem for dynamics. Incomplete internal sequences or any amino acid residues that have missing atoms **will** cause `pdb2gmx` to fail. These missing atoms/residues must be modeled in using other software packages. Also note that `pdb2gmx` is not magic. It cannot generate topologies for arbitrary molecules, just the residues defined by the force field (in the `*.rtp` files - generally proteins, nucleic acids, and a **very** finite amount of cofactors, like NAD(H) and ATP).

Now that the crystal waters are gone and we have verified that all the necessary atoms are present, the PDB file should contain only protein atoms, and is ready to be input into the first GROMACS module, `pdb2gmx`. The purpose of `pdb2gmx` is to generate three files:

1. The topology for the molecule.
2. A position restraint file.
3. A post-processed structure file.

The topology (`topo1.top` by default) contains all the information necessary to define the molecule within a simulation. This information includes nonbonded parameters (atom types and charges) as well as bonded parameters (bonds, angles, and dihedrals). We will take a more detailed look at the topology once it has been generated.

Execute `pdb2gmx` by issuing the following command:

```
gmx pdb2gmx -f 1AKI.pdb -o 1AKI_processed.gro -water tip3p
```

The structure will be processed by `pdb2gmx`, and you will be prompted to choose a force field:

Select the Force Field:

From '/usr/local/gromacs/share/gromacs/top':

- 1: AMBER03 protein, nucleic AMBER94 (Duan et al., J. Comp. Chem. 24, 1999-2012, 2003)
- 2: AMBER94 force field (Cornell et al., JACS 117, 5179-5197, 1995)
- 3: AMBER96 protein, nucleic AMBER94 (Kollman et al., Acc. Chem. Res. 29, 461-469, 1996)
- 4: AMBER99 protein, nucleic AMBER94 (Wang et al., J. Comp. Chem. 21, 1049-1074, 2000)
- 5: AMBER99SB protein, nucleic AMBER94 (Hornak et al., Proteins 65, 712-725, 2006)
- 6: AMBER99SB-ILDN protein, nucleic AMBER94 (Lindorff-Larsen et al., Proteins 78, 1950-58, 2010)
- 7: AMBERGS force field (Garcia & Sanbonmatsu, PNAS 99, 2782-2787, 2002)
- 8: CHARMM27 all-atom force field (CHARM22 plus CMAP for proteins)
- 9: GROMOS96 43a1 force field
- 10: GROMOS96 43a2 force field (improved alkane dihedrals)
- 11: GROMOS96 45a3 force field (Schuler JCC 2001 22 1205)
- 12: GROMOS96 53a5 force field (JCC 2004 vol 25 pag 1656)
- 13: GROMOS96 53a6 force field (JCC 2004 vol 25 pag 1656)
- 14: GROMOS96 54a7 force field (Eur. Biophys. J. (2011), 40,, 843-856, DOI: 10.1007/s00249-011-0700-9)
- 15: OPLS-AA/L all-atom force field (2001 aminoacid dihedrals)

The force field will contain the information that will be written to the topology. This is a very important choice! You should always read thoroughly about each force field and decide which is most applicable to your situation. For this tutorial, we will use the all-atom OPLS force field, so type 15 at the command prompt, followed by 'Enter'.

There are many other options that can be passed to `pdb2gmx`. Some commonly used ones are listed here:

- `-ignh`: Ignore H atoms in the PDB file; especially useful for NMR structures. Otherwise, if H atoms are present, they must be in the file named exactly how the force fields in GROMACS expect them to be. Different conventions exist, so dealing with H atoms can

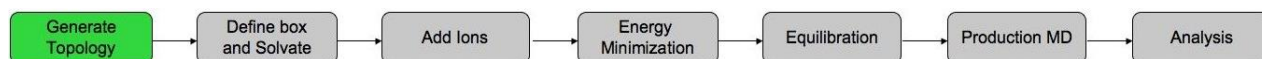
occasionally be a headache! If you need to preserve the initial H coordinates, but renaming is required, then the Linux `sed` command is your friend.

- `-ter`: Interactively assign charge states for N- and C-termini.
- `-inter`: Interactively assign charge states for Glu, Asp, Lys, Arg, and His; choose which Cys are involved in disulfide bonds.

You have now generated three new files: `1AKI_processed.gro`, `topol.top`, and `posre.itp`. `1AKI_processed.gro` is a GROMACS-formatted structure file that contains all the atoms defined within the force field (*i.e.*, H atoms have been added to the amino acids in the protein). The `topol.top` file is the system topology (more on this in a minute). The `posre.itp` file contains information used to restrain the positions of heavy atoms (more on this later).

One final note: many users assume that a `.gro` file is mandatory. **This is not true.** GROMACS can handle many different file formats, with `.gro` simply being the default for commands that write coordinate files. It is a very compact format, but it has limited precision. If you prefer to use, for instance, PDB format, all you need to do is to specify an appropriate file name with `.pdb` extension as your output. The purpose of `pdb2gmx` is to produce a force field-compliant topology; the output structure is largely a side effect of this purpose and is intended for user convenience. The format can be just about anything you like (see the GROMACS manual for different formats).

## Step Two: Examine the Topology



Let's look at what is in the output topology (`topol.top`). Again, using a plain text editor, inspect its contents. After several comment lines (preceded by `;`), you will find the following:

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

This line calls the parameters within the OPLS-AA force field. It is at the beginning of the file, indicating that all subsequent parameters are derived from this force field. The next important line is `[ moleculetype ]`, below which you will find

```
; Name          nrexcl
Protein_chain_A  3
```

The name `"Protein_chain_A"` defines the molecule name, based on the fact that the protein was labeled as chain A in the PDB file. There are 3 exclusions for bonded neighbors. More information on exclusions can be found in the GROMACS manual; a discussion of this information is beyond the scope of this tutorial.

The next section defines the `[ atoms ]` in the protein. The information is presented as columns:

```
[ atoms ]
;  nr type  resnr residue  atom  cgnr charge mass typeB  chargeB  massB
```

```

; residue 1 LYS rtp LYSH q +2.0
  1  opl_s_287  1  LYS  N  1  -0.3  14.0067  ; qtot -0.3
  2  opl_s_290  1  LYS  H1  1  0.33  1.008  ; qtot 0.03
  3  opl_s_290  1  LYS  H2  1  0.33  1.008  ; qtot 0.36
  4  opl_s_290  1  LYS  H3  1  0.33  1.008  ; qtot 0.69
  5  opl_s_293B 1  LYS  CA  1  0.25  12.011  ; qtot 0.94
  6  opl_s_140  1  LYS  HA  1  0.06  1.008  ; qtot 1

```

The interpretation of this information is as follows:

- **nr**: Atom number
- **type**: Atom type
- **resnr**: Amino acid residue number
- **residue**: The amino acid residue name

Note that this residue was "LYS" in the PDB file; the use of .rtp entry "LYSH" indicates that the residue is protonated (the predominant state at neutral pH).

- **atom**: Atom name
- **cgnr**: Charge group number

Charge groups define units of integer charge; they aid in speeding up calculations

- **charge**: Self-explanatory

The "qtot" descriptor is a running total of the charge on the molecule

- **mass**: Also self-explanatory
- **typeB**, **chargeB**, **massB**: Used for free energy perturbation (not discussed here)

Subsequent sections include [ **bonds** ], [ **pairs** ], [ **angles** ], and [ **dihedrals** ]. Some of these sections are self-explanatory (bonds, angles, and dihedrals). The parameters and function types associated with these sections are elaborated on in Chapter 5 of the GROMACS manual. Special 1-4 interactions are included under "**pairs**" (section 5.3.4 of the GROMACS manual).

The remainder of the file involves defining a few other useful/necessary topologies, starting with position restraints. The "**posre.itp**" file was generated by **pdb2gmx**; it defines a force constant used to keep atoms in place during equilibration (more on this later).

```

; Include Position restraint file
#ifdef POSRES
#include "posre.itp"
#endif

```

This ends the "**Protein\_chain\_A**" **moleculetype** definition. The remainder of the topology file is dedicated to defining other molecules and providing system-level descriptions. The next moleculetype (by default) is the solvent, in this case **TIP3P** water. Other typical choices for water include **SPC**, **SPC/E**, and **TIP4P**. We chose this by passing "**-water tip3p**" to **pdb2gmx**. For an

excellent summary of the many different water models, click [here](#), but be aware that not all of these models are present within GROMACS.

```
; Include water topology
#include "oplsaa.ff/tip3p.itp"

#ifdef POSRES_WATER
; Position restraint for each water oxygen
[ position_restraints ]
; i funct      fcx      fcy      fcz
  1    1        1000    1000    1000
#endif
```

As you can see, water can also be position-restrained, using a force constant ( $k_{pr}$ ) of 1000 kJ mol<sup>-1</sup> nm<sup>-2</sup>.

Ion parameters are included next:

```
; Include generic topology for ions
#include "oplsaa.ff/ions.itp"
```

Finally come system-level definitions. The [ **system** ] directive gives the name of the system that will be written to output files during the simulation. The [ **molecules** ] directive lists all of the molecules in the system.

```
[ system ]
; Name
LYSOZYME

[ molecules ]
; Compound          #mols
Protein_chain_A     1
```

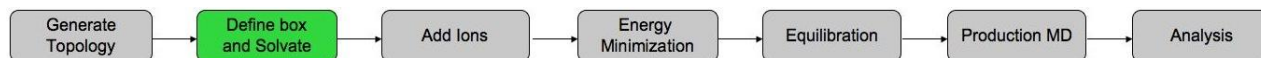
A few key notes about the [ **molecules** ] directive:

1. The order of the listed molecules must **exactly** match the order of the molecules in the coordinate (in this case, .gro) file.
2. The names listed must match the [ **moleculetype** ] name for each species, not residue names or anything else.

If you fail to satisfy these concrete requirements at any time, you will get fatal errors from **grompp** (discussed later) about mismatched names, molecules not being found, or a number of others.

Now that we have examined the contents of a topology file, we can continue building our system.

## Step Three: Defining the Unit Cell & Adding Solvent



Now that you are familiar with the contents of the GROMACS topology, it is time to continue building our system. In this example, we are going to be simulating a simple aqueous system. It is possible to simulate proteins and other molecules in different solvents, provided that good parameters are available for all species involved.

There are two steps to defining the box and filling it with solvent:

1. Define the box dimensions using the `editconf` module.
2. Fill the box with water using the `solvate` module (formerly called `genbox`).

You are now presented with a choice as to how to treat the unit cell. For the purpose of this tutorial, we will use a simple cubic box as the unit cell. As you become more comfortable with periodic boundary conditions and box types, I highly recommend the rhombic dodecahedron, as its volume is ~71% of the cubic box of the same periodic distance, thus saving on the number of water molecules that need to be added to solvate the protein.

Let's define the box using `editconf`:

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

The above command centers the protein in the box (`-c`), and places it at least 1.0 nm from the box edge (`-d 1.0`). The box type is defined as a cube (`-bt cubic`). The distance to the edge of the box is an important parameter. Since we will be using periodic boundary conditions, we must satisfy the minimum image convention. That is, a protein should never see its periodic image, otherwise the forces calculated will be spurious. Specifying a solute-box distance of 1.0 nm will mean that there are at least 2.0 nm between any two periodic images of a protein. This distance will be sufficient for just about any cutoff scheme commonly used in simulations.

Now that we have defined a box, we can fill it with solvent (water). Solvation is accomplished using `solvate`:

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

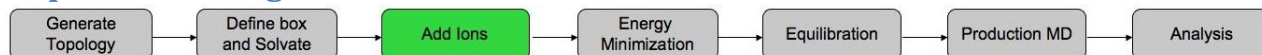
The configuration of the protein (`-cp`) is contained in the output of the previous `editconf` step, and the configuration of the solvent (`-cs`) is part of the standard GROMACS installation. We are using `spc216.gro`, which is a generic equilibrated 3-point solvent model. You can use `spc216.gro` as the solvent configuration for `SPC`, `SPC/E`, or `TIP3P` water, since they are all three-point water models. The output is called `1AKI_solv.gro`, and we tell `solvate` the name of the topology file (`topol.top`) so it can be modified. Note the changes to the `[ molecules ]` directive of `topol.top`:

```
[ molecules ]
; Compound          #mols
```

```
Protein_chain_A      1
SOL                  10644
```

What `solvate` has done is keep track of how many water molecules it has added, which it then writes to your topology to reflect the changes that have been made. Note that if you use any other (non-water) solvent, `solvate` will not make these changes to your topology! Its compatibility with updating water molecules is hard-coded.

### Step Four: Adding Ions



We now have a solvated system that contains a charged protein. The output of `pdb2gmx` told us that the protein has a net charge of  $+8e$  (based on its amino acid composition). If you missed this information in the `pdb2gmx` output, look at the last line of your `[ atoms ]` directive in `topol.top`; it should read (in part) "`qtot 8.`" Since life does not exist at a net charge, we must add ions to our system.

The tool for adding ions within GROMACS is called `genion`. What `genion` does is read through the topology and replace water molecules with the ions that the user specifies. The input is called a run input file, which has an extension of `.tpr`; this file is produced by the GROMACS `grompp` module (GROMACS pre-processor), which will also be used later when we run our first simulation. What `grompp` does is process the coordinate file and topology (which describes the molecules) to generate an atomic-level input (`.tpr`). The `.tpr` file contains all the parameters for all of the atoms in the system.

To produce a `.tpr` file with `grompp`, we will need an additional input file, with the extension `.mdp` (molecular dynamics parameter file); `grompp` will assemble the parameters specified in the `.mdp` file with the coordinates and topology information to generate a `.tpr` file.

An `.mdp` file is normally used to run energy minimization or an MD simulation, but in this case is simply used to generate an atomic description of the system. An example `.mdp` file (the one we will use) is named `ions.mdp`.

In reality, the `.mdp` file used at this step can contain any legitimate combination of parameters. I typically use an energy-minimization script, because they are very basic and do not involve any complicated parameter combinations. **Please note** that the files provided with this tutorial are intended **only** for use with the OPLS-AA force field. Settings, particularly nonbonded interaction settings, will be different for other force fields.

Assemble your `.tpr` file with the following:

```
gmx grompp -f ions.mdp -c lAKI_solv.gro -p topol.top -o ions.tpr
```

Did you get an error? Why? Since we're in the middle of preparing the system, we can accept the *warning*, but it needs to be done explicitly. Retry with:

```
gmx grompp -f ions.mdp -c lAKI_solv.gro -p topol.top -o ions.tpr -maxwarn 1
```

Now we have an atomic-level description of our system in the binary file `ions.tpr`. We will pass this file to `genion` (all in one line):

```
gmx genion -s ions.tpr -o lAKI_solv_ions.gro -p topol.top -pname NA \  
-nname CL -nn 8
```

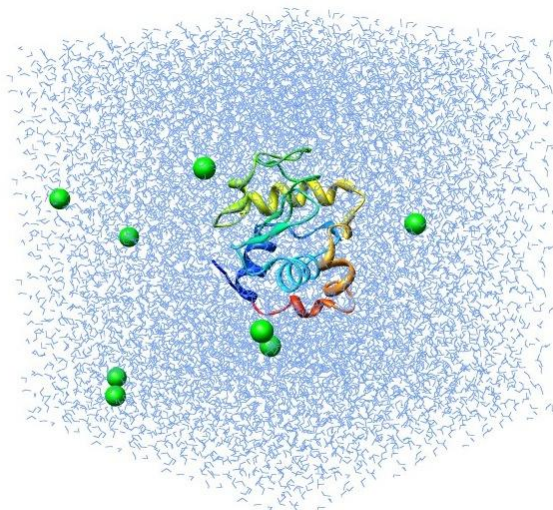
When prompted, choose group 13 "SOL" for embedding ions. You do not want to replace parts of your protein with ions.

In the `genion` command, we provide the structure/state file (`-s`) as input, generate a `.gro` file as output (`-o`), process the topology (`-p`) to reflect the removal of water molecules and addition of ions, define positive and negative ion names (`-pname` and `-nname`, respectively), and tell `genion` to add only the ions necessary to neutralize the net charge on the protein by adding the correct number of negative ions (`-nn 8`). You could also use `genion` to add a specified concentration of ions in addition to simply neutralizing the system by specifying the `-neutral` and `-conc` options in conjunction. Refer to the `genion` man page for information on how to use these options.

The names of the ions specified with `-pname` and `-nname` were force field-specific in previous versions of GROMACS, but have been standardized as of version 4.5. The specified ion names are always the elemental symbol in all capital letters, which is the [ `moleculetype` ] name that is then written to the topology. Residue or atom names may or may not append the sign of the charge (+/-), depending on the force field. **Do not use atom or residue names in the `genion` command, or you will encounter errors in subsequent steps.**

Your [ `molecules` ] directive should now look like:

```
[ molecules ]  
; Compound          #mols  
Protein_chain_A     1  
SOL                  10636  
CL                   8
```



## Step Five: Energy Minimization



The solvated, electroneutral system is now assembled. Before we can begin dynamics, we must ensure that the system has no steric clashes or inappropriate geometry. The structure is relaxed through a process called energy minimization (EM).

The process for EM is much like the addition of ions. We are once again going to use **grompp** to assemble the structure, topology, and simulation parameters into a binary input file (.tpr), but this time, instead of passing the .tpr to **genion**, we will run the energy minimization through the GROMACS MD engine, **mdrun**.

Assemble the binary input using **grompp** using a parameter file (**minim.mdp**):

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

Make sure you have been updating your **topol.top** file when running **genbox** and **genion**, or else you will get lots of nasty error messages ("number of coordinates in coordinate file does not match topology," etc).

We are now ready to invoke **mdrun** to carry out the EM either as:

```
gmx mdrun -nt 4 -v -deffnm em
```

in a node, where you have access to 4 cores (or 6, like a Taito-GPU node), or

```
gmx mdrun -v -deffnm em
```

if you can use all existing cores. The **-v** flag is for the impatient: it makes **mdrun** verbose, such that it prints its progress to the screen at every step (**don't** use this in production runs. It generates unnecessary overhead). The **-deffnm** flag will define the file names of the input and output. So, if you did not name your **grompp** output "**em.tpr**," you will have to explicitly specify its name with the **mdrun -s** flag. In our case, we will get the following files:

- **em.log**: ASCII-text log file of the EM process
- **em.edr**: Binary energy file
- **em.trr**: Binary full-precision trajectory
- **em.gro**: Energy-minimized structure

There are two very important factors to evaluate to determine if EM was successful. The first is the potential energy (printed at the end of the EM process, even without **-v**).  $E_{\text{pot}}$  should be negative, and (for a simple protein in water) on the order of  $10^5$ - $10^6$ , depending on the system size and number of water molecules. The second important feature is the maximum force,  $F_{\text{max}}$ , the target for which was set in **minim.mdp** - "**emtol = 1000.0**" - indicating a target  $F_{\text{max}}$  of no greater than  $1000 \text{ kJ mol}^{-1} \text{ nm}^{-1}$ . It is possible to arrive at a reasonable  $E_{\text{pot}}$  with  $F_{\text{max}} > \text{emtol}$ . If

this happens, your system may not be stable enough for simulation. Evaluate why it may be happening, and perhaps change your minimization parameters (`integrator`, `emstep`, etc).

Let's do a bit of analysis. The `em.edr` file contains all of the energy terms that GROMACS collects during EM. You can analyze any `.edr` file using the GROMACS energy module:

```
gmx energy -f em.edr -o potential.xvg
```

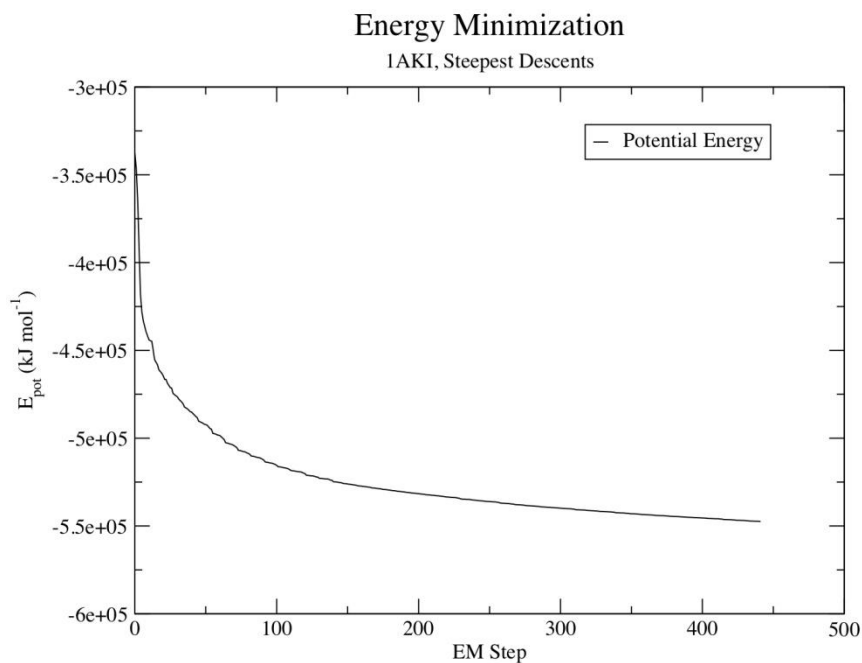
At the prompt, type "10 0" to select Potential (10); zero (0) terminates input. You will be shown the average of  $E_{\text{pot}}$ , and a file called "potential.xvg" will be written. To plot this data, you will need the [Xmgrace](#) (`xmgrace potential.xvg`) plotting tool.

Or if you don't have `xmgrace` available, you can use e.g. `gnuplot`

```
gnuplot
gnuplot> plot 'potential.xvg' using 1:2 with line
```

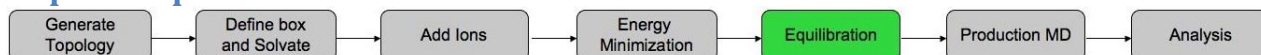
Exit `gnuplot` with `quit`.

The resulting plot should look something like this (the picture below has some additional formatting), demonstrating the nice, steady convergence of  $E_{\text{pot}}$ :



Now that our system is at an energy minimum, we can begin real dynamics.

### Step Six: Equilibration



EM ensured that we have a reasonable starting structure, in terms of geometry and solvent orientation. To begin real dynamics, we must equilibrate the solvent and ions around the protein. If we were to attempt unrestrained dynamics at this point, the system might collapse. The reason is that the solvent is mostly optimized within itself, and not necessarily with the solute. It needs to be brought to the temperature we wish to simulate and establish the proper orientation about the solute (the protein). After we arrive at the correct temperature (based on kinetic energies), we will apply pressure to the system until it reaches the proper density.

Remember that `posre.itp` file that `pdb2gmx` generated a long time ago? We're going to use it now. The purpose of `posre.itp` is to apply a position restraining force on the heavy atoms of the protein (anything that is not a hydrogen). Movement is permitted, but only after overcoming a substantial energy penalty. The utility of position restraints is that they allow us to equilibrate our solvent around our protein, without the added variable of structural changes in the protein.

Equilibration is often conducted in two phases. The first phase is conducted under an *NVT* ensemble (constant Number of particles, Volume, and Temperature). This ensemble is also referred to as "isothermal-isochoric" or "canonical." The timeframe for such a procedure is dependent upon the contents of the system, but in *NVT*, the temperature of the system should reach a plateau at the desired value. If the temperature has not yet stabilized, additional time will be required. Typically, 50-100 ps should suffice, and we will conduct a 100-ps *NVT* equilibration for this exercise. Depending on your machine, this may take a while (just over an hour on a dual-core MacBook). The parameters are in a file called `nvt.mdp`:

We will call `grompp` and `mdrun` just as we did at the EM step:

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

The example below is for an interactive command. For a bigger system, this step should be run as a batch job. The input files include a template SLURM script (`batch.bash`). If you're running this exercise on Taito, use that instead. Just put the `gmx mdrun . . .` command there (without the `-v flag!`) and match the number of `--cpus-per-task=N` requested in the `#SBATCH` line):

```
gmx mdrun -nt 4 -v -deffnm nvt (or just gmx mdrun -v -deffnm nvt to let gmx to choose the number of cores to use, remember NOT to use -v in production calculations)
```

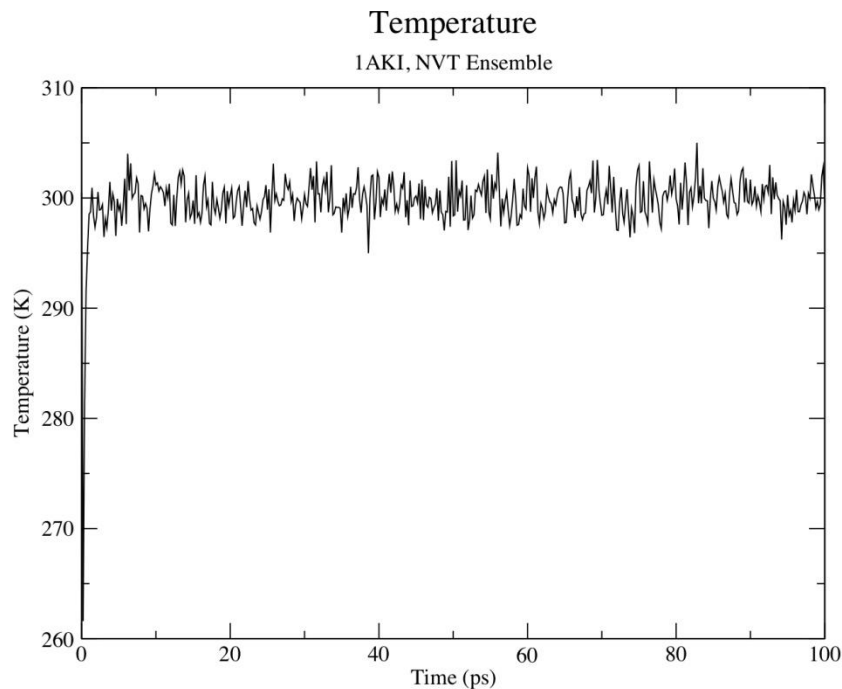
A full explanation of the parameters used can be found in the GROMACS manual, in addition to the comments provided. Take note of a few parameters in the `.mdp` file:

- `gen_vel = yes`: Initiates velocity generation. Using different random seeds (`gen_seed`) gives different initial velocities, and thus multiple (different) simulations can be conducted from the same starting structure.
- `tcoupl = v-rescale`: The velocity rescaling thermostat is an improvement upon the Berendsen weak coupling method, which did not reproduce a correct kinetic ensemble.
- `pcoupl = no`: Pressure coupling is not applied.

Let's analyze the temperature progression, again using energy:

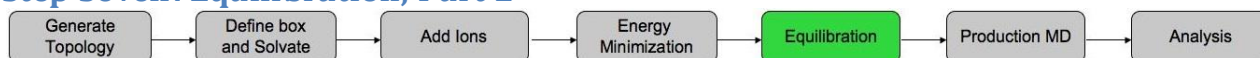
```
gmx energy -f nvt.edr -o temperature.xvg
```

Type "15 0" at the prompt to select the temperature of the system and exit. The resulting plot should look something like the following:



From the plot, it is clear that the temperature of the system quickly reaches the target value (300 K), and remains stable over the remainder of the equilibration. For this system, a shorter equilibration period (on the order of 50 ps) might have been adequate.

## Step Seven: Equilibration, Part 2



The previous step, *NVT* equilibration, stabilized the temperature of the system. Prior to data collection, we must also stabilize the pressure (and thus also the density) of the system. Equilibration of pressure is conducted under an *NPT* ensemble, wherein the Number of particles, Pressure, and Temperature are all constant. The ensemble is also called the "isothermal-isobaric" ensemble, and most closely resembles experimental conditions.

The `.mdp` file used for a 100-ps *NPT* equilibration is `npt.mdp`. It is not drastically different from the parameter file used for *NVT* equilibration. Note the addition of the pressure coupling section, using the Parrinello-Rahman barostat.

A few other changes:

- `continuation = yes`: We are continuing the simulation from the *NVT* equilibration phase

- `gen_vel = no`: Velocities are read from the trajectory (see below)

We will call `grompp` and `mdrun` just as we did for *NVT* equilibration. Note that we are now including the `-t` flag to include the checkpoint file from the *NVT* equilibration; this file contains all the necessary state variables to continue our simulation. To conserve the velocities produced during *NVT*, we must include this file. The coordinate file (`-c`) is the final output of the *NVT* simulation.

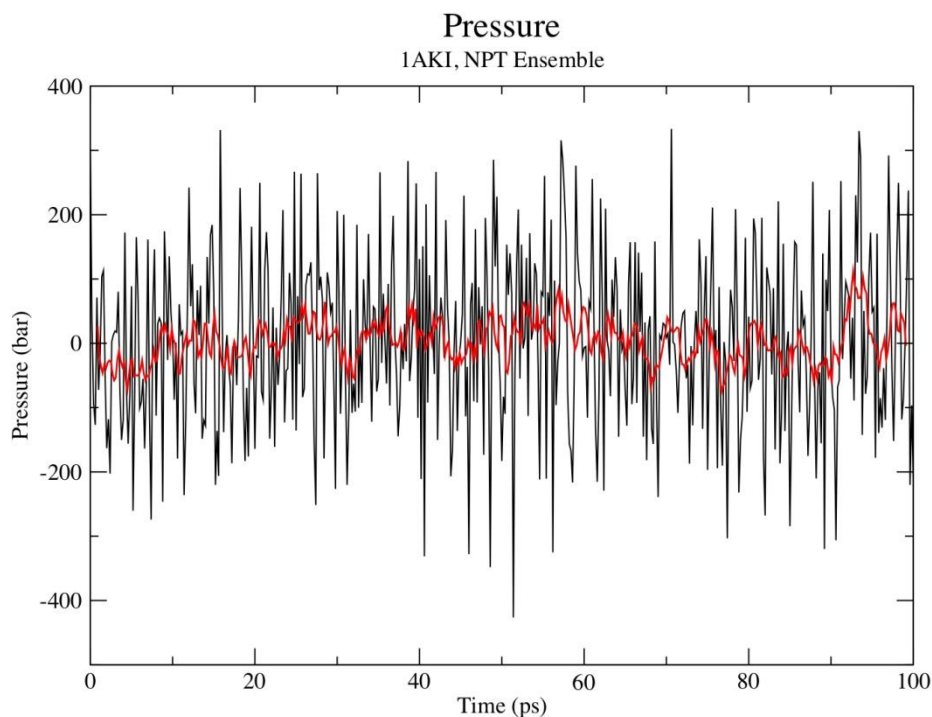
```
gmx grompp -f npt.mdp -c nvt.gro -r nvt.gro \
          -t nvt.cpt -p topol.top -o npt.tpr
```

```
gmx mdrun -nt 4 -deffnm npt
```

Let's analyze the pressure progression, again using `energy`:

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

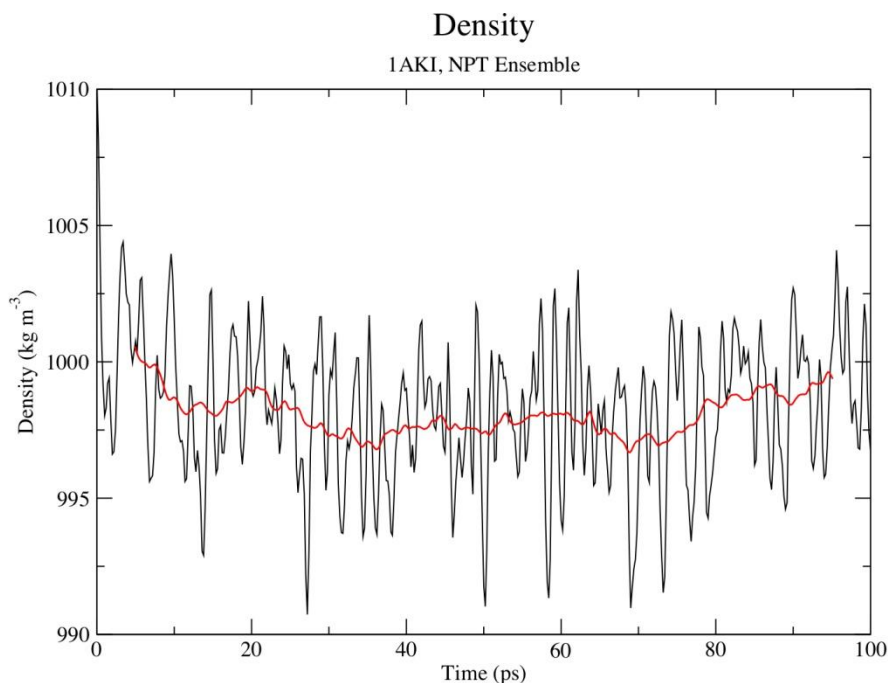
Type "17 0" at the prompt to select the pressure of the system and exit. The resulting plot should look something like the following (without the running average):



The pressure value fluctuates widely over the course of the 100-ps equilibration phase, but this behavior is not unexpected. The running average of these data are plotted as the red line in the plot. Over the course of the equilibration, the average value of the pressure is 1.05 bar.

Let's take a look at density as well, this time using `energy` and entering "22 0" at the prompt.

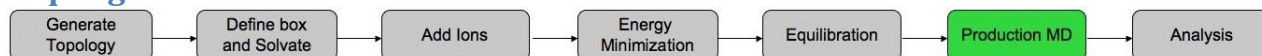
```
gmx energy -f npt.edr -o density.xvg
```



As with the pressure, the running average of the density is here also plotted in red. The average value over the course of 100 ps is  $1006 \text{ kg m}^{-3}$ , close to the experimental value of  $997.1 \text{ kg m}^{-3}$  and the expected density of the TIP3P model of  $1002 \text{ kg m}^{-3}$  (298K). The parameters for the TIP3P water model closely replicate experimental values for water. The density values are rather stable over time, indicating that the system is well-equilibrated now with respect to pressure and density.

**Please note:** there are frequently questions about why density values obtained do not match the ones shown here. Pressure-related terms are slow to converge, and thus you may have to run *NPT* equilibration slightly longer than is specified here.

### Step Eight: Production MD



Upon completion of the two equilibration phases, the system is now well-equilibrated at the desired temperature and pressure. We are now ready to release the position restraints and run production MD for data collection. The process is just like we have seen before, as we will make use of the checkpoint file (which in this case now contains preserve pressure coupling information) to **grompp**. We will run a 1-ns MD simulation, with parameters defined in **md.mdp**. With 4 cores this simulation will take more than 1 hour. Normally, you would run these as batch jobs. There is an example batch script among the files (**batch.bash**). If you want you can use that instead, but it needs some editing.

```

gmX grompp -f md.mdp -c npt.gro -r npt.gro -t npt.cpt \
          -p topol.top -o md_0_1.tpr

```

```

gmX mdrun -nt 4 -deffnm md_0_1

```

...  
**Estimate for the relative computational load of the PME mesh part: 0.25**

The estimate for PME load will dictate how many processors should be dedicated to the PME calculation, and how many for the PP calculations. Refer to the GROMACS 4 [publication](#) and the manual for details. For a cubic box, the optimal setup will have a PME load of 0.25 (3:1 PP:PME - we're in luck!); for a dodecahedral box, the optimal PME load is 0.33 (2:1 PP:PME). When executing `mdrun`, the program should automatically determine the best number of processors to assign for the PP and PME calculations. Thus, make sure you indicate an appropriate number of cores for your calculation (the value of `-nt x` or `-np x`, and check your queuing system guide!), so that you can get the best performance.

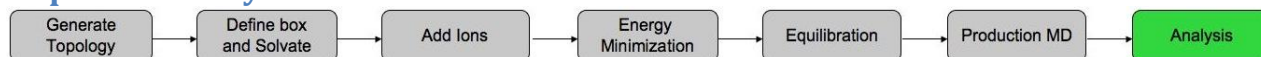
### Running GROMACS on GPU

As of version 4.6, GROMACS supports the use of GPU accelerators for running MD simulations. The nonbonded interactions are calculated on the GPU, and bonded and PME interactions are calculated on standard CPU hardware. When building GROMACS (see [www.gromacs.org](http://www.gromacs.org) for installation instructions), GPU hardware will automatically be detected, if present. The minimum requirements for using GPU acceleration are the CUDA libraries and SDK, and a GPU card with a compute capability of 2.0. A nice list of some of the more common cards and their specifications can be found [here](#). To use a GPU, the only change to the `.mdp` file posted above would be to add the following line to make use of the Verlet cutoff scheme (the old group scheme is not supported on GPU):

```
cutoff-scheme = Verlet
```

Assuming you have one GPU available, the `mdrun` uses it by default. If you have more than one card available, or require customization of how the work is divided up via the hybrid parallelization scheme available in GROMACS, please consult the GROMACS manual and webpage. Such technical details are beyond the scope of this tutorial.

### Step Nine: Analysis



Now that we have simulated our protein, we should run some analysis on the system. What types of data are important? This is an important question to ask before running the simulation, so you should have some ideas about the types of data you will want to collect in your own systems. For this tutorial, a few basic tools will be introduced.

The first is `trjconv`, which is used as a post-processing tool to strip out coordinates, correct for periodicity, or manually alter the trajectory (time units, frame frequency, etc). First we'll just have a look at the trajectory visually. The protein will diffuse through the unit cell, and may appear to "jump" across to the other side of the box. For visualization we'll first print out only the Protein and center the trajectory with its center of mass. To that end, we will use `trjconv` which can also be used to account for any periodicity in the system. To prepare the trajectory for visualization issue the following:

```
gmx trjconv -s md_0_1.tpr -f md_0_1.xtc -o md_p.xtc -pbc mol \  
-ur compact -center
```

Select 1 ("Protein") for both centering and output. For VMD we'll also need a gro-file including only the protein:

```
gmx trjconv -s md_0_1.tpr -f md_0_1.xtc -o md_p.gro -pbc mol \  
-ur compact -center -dump 1000
```

Select also now the protein for both centering and output. You can now use the `view.vmd` script to display the trajectory with VMD (Note: If you're running this example on Taito-GPU prepend `vmd: vglrun vmd -e view.vmd`):

```
vmd -e view.vmd
```

If you get an error, change the xtc-file referred to in the `view.vmd` script to the one you want to visualize. If the simulation looks ok by visual inspection, we can proceed to proper analysis. For that end, we'll rerun the trajectory tool but print out also the solvent and ions (select "System"):

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

We will conduct all our analyses on this "corrected" trajectory. Let's look at structural stability first. GROMACS has a built-in utility for RMSD calculations called `rms`. To use `rms`, issue this command:

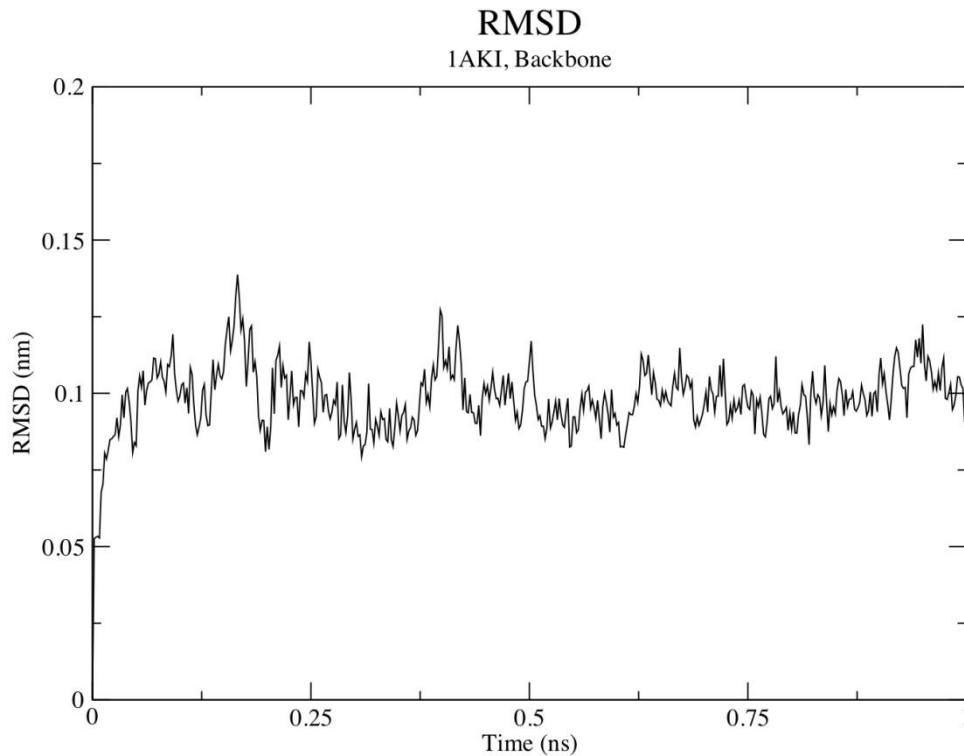
```
gmx rms -s md_0_1.tpr -f md_0_1_noPBC.xtc -o rmsd.xvg -tu ns
```

Choose 4 ("Backbone") for both the least squares fit and the group for RMSD calculation. The `-tu` flag will output the results in terms of ns, even though the trajectory was written in ps. This is done for clarity of the output (especially if you have a long simulation - 1e+05 ps does not look as nice as 100 ns). The output plot will show the RMSD relative to the structure present in the minimized, equilibrated system:

```
xmgrace rmsd.xvg
```

or

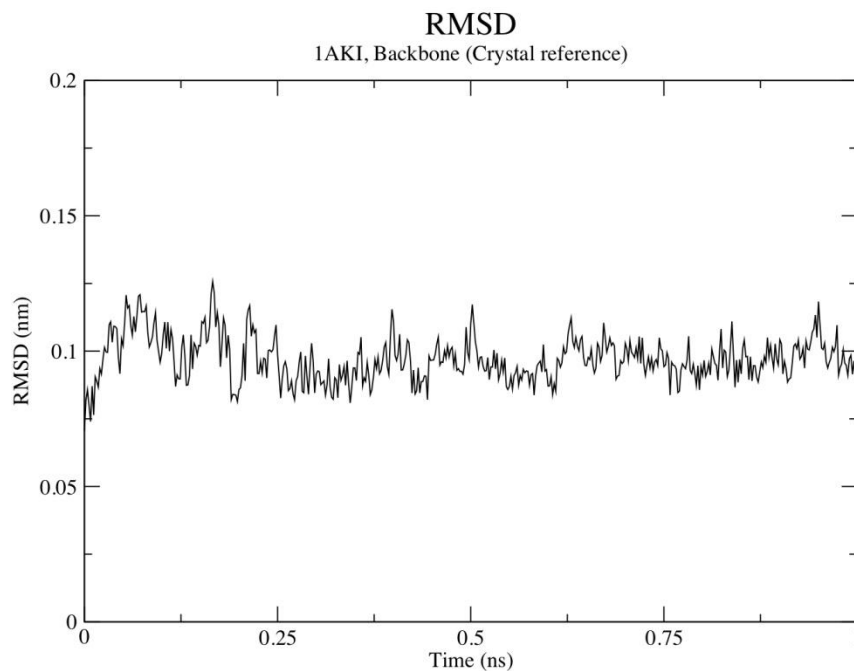
```
gnuplot  
gnuplot> 'rmsd.xvg' u 1:2 w l
```



If we wish to calculate RMSD relative to the crystal structure, we could issue the following:

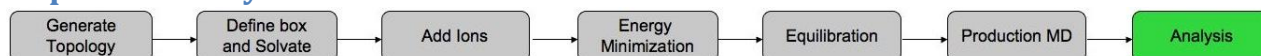
```
gmx rms -s em.tpr -f md_0_1_noPBC.xtc -o rmsd_xtal.xvg -tu ns
```

Choose again **Backbone** for fitting. The result will be something like:



Both plots show the RMSD levels off to  $\sim 0.1$  nm ( $1 \text{ \AA}$ ), indicating that the structure is very stable. Subtle differences between the plots indicate that the structure at  $t = 0$  ns is slightly different from this crystal structure. This is to be expected, since it has been energy-minimized, and because the position restraints are not 100% perfect, as discussed previously.

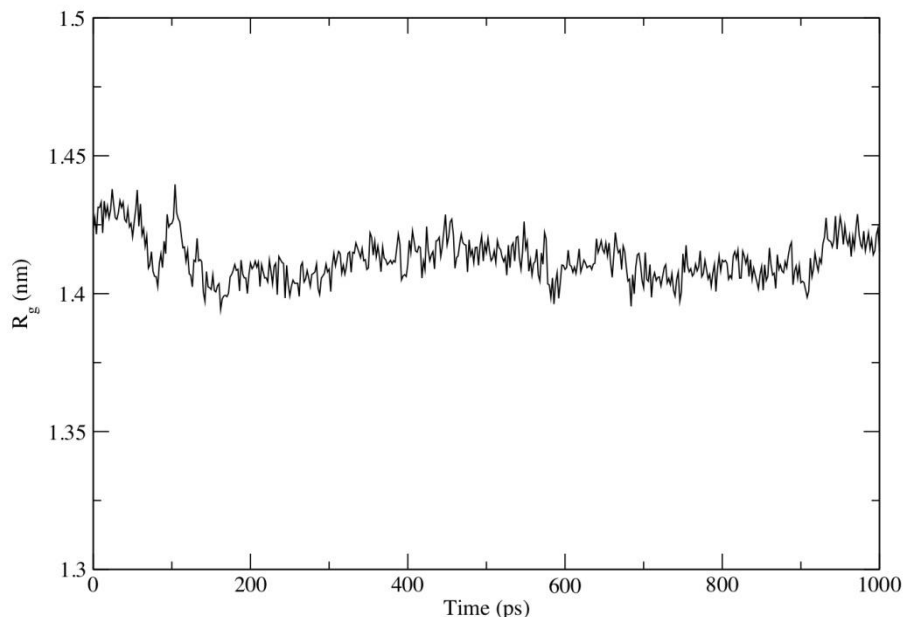
### Step Nine: Analysis continued



The radius of gyration of a protein is a measure of its compactness. If a protein is stably folded, it will likely maintain a relatively steady value of  $R_g$ . If a protein unfolds, its  $R_g$  will change over time. Let's analyze the radius of gyration for lysozyme in our simulation:

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

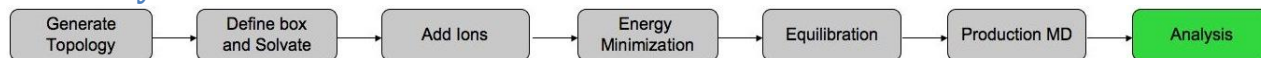
Radius of gyration



We can see from the reasonably invariant  $R_g$  values that the protein remains very stable, in its compact (folded) form over the course of 1 ns at 300 K. You can also try `xmgrace -nxy gyrate.xvg`. This result is not unexpected, but illustrates an advanced capacity of GROMACS analysis that comes built-in. With gnuplot you can replot the other columns e.g. with:

```
gnuplot> replot 'gyrate' u 1:3 w l
```

### Summary



You have now conducted a molecular dynamics simulation with GROMACS, and analyzed some of the results. This tutorial should not be viewed as comprehensive. There are many more

types of simulations that one can conduct with GROMACS (free energy calculations, non-equilibrium MD, and normal modes analysis, just to name a few). You should also review the literature and the GROMACS manual for adjustments to the `.mdp` files provided here for efficiency and accuracy purposes.

Happy simulating!