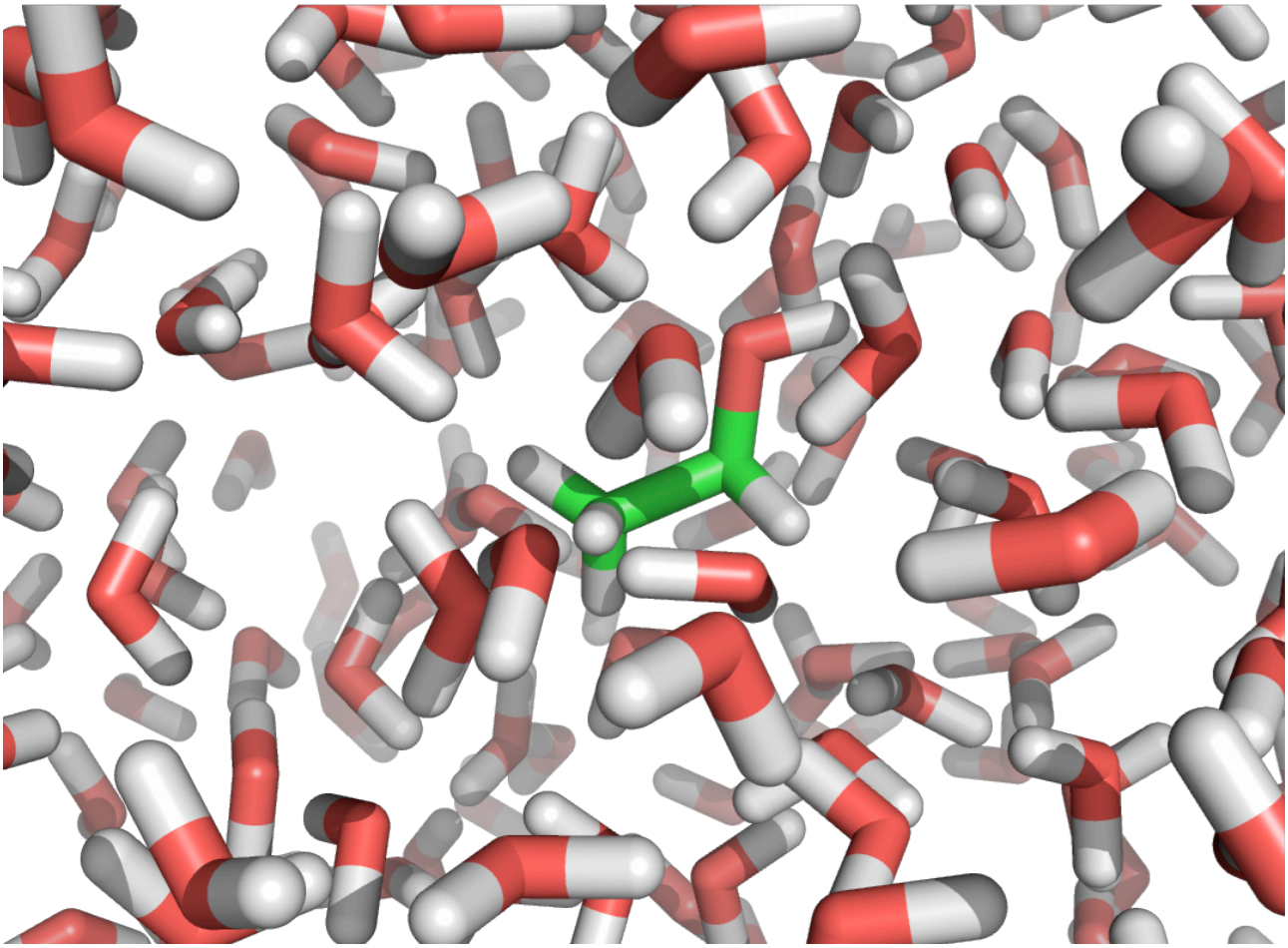
Free Energy Calculation with GROMACS



Hands-On Tutorial

Solvation free energy of ethanol

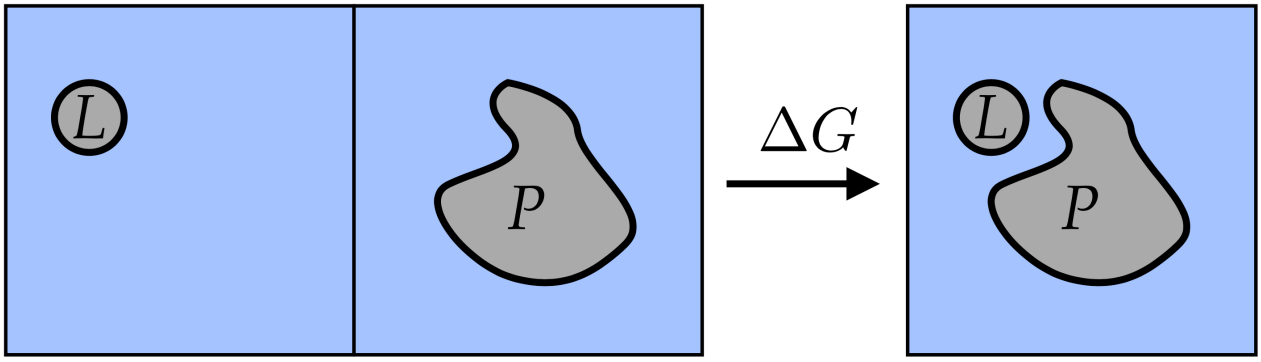
Background

In this tutorial, we'll calculate the free energy of solvation of a small molecule: ethanol. This type of calculation can either be done on its own, or can be part of a binding free energy calculation. Such calculations can be important, because the free energy is the most important static quantity in a thermal system: its sign determines the whether of a molecule will be soluble, or whether it will bind to another molecule.

We will start this tutorial with some background on how to calculate free energies, and how a free energy of solvation relates to a free energy of binding calculation. Then, we will focus on the practicalities of doing such a calculation in GROMACS. You should likely use GROMACS 5.1 (or later) for this tutorial.

Calculating a binding free energy

Calculating free energies can usually only be done using small steps and a full path between one end state and the other. For example, to calculate the binding free energy of the ligand to a protein, we ultimately need to compare the situation of the ligand being bound to the protein, to the situation where both the ligand and the protein are separately in solution:



This could be calculated directly, for example by dragging the ligand away from the protein and integrating the potential of mean force (averaging the force, and integrating it). Forces have very large fluctuations, however, and this turns out to be much more expensive than using *free energy perturbation* methods such as the Bennett Acceptance Ratio (BAR) that we'll use in the tutorial.

Remember that a free energy difference between two states A and B determines their relative probability p_A and p_B,

$$\frac{p_A}{p_B} = \exp \frac{F_B - F_A}{k_B T}$$

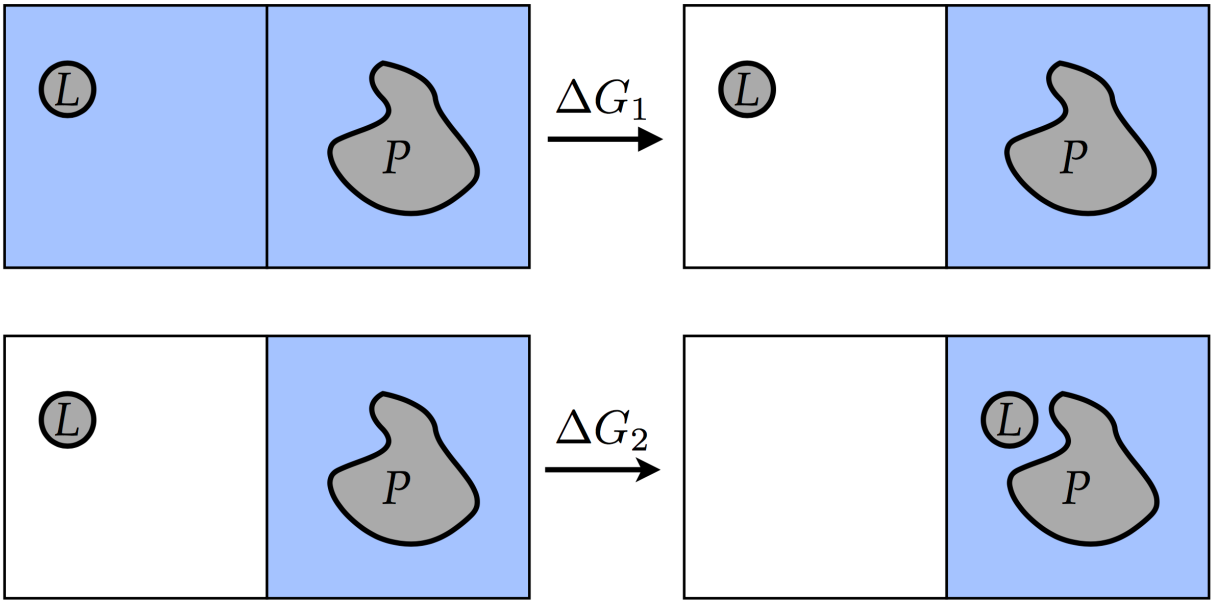
where k_B is Boltzmann's constant relating thermal energy to the temperature ($1.38 \cdot 10^{-23}$ J/K), and T is the temperature. We could, in principle, calculate a free energy difference by waiting long enough, and measuring how often the system is in which state. The free energy differences, however, are often of the order of tens of kJ/mol: for example, the free energy of solvation of ethanol at 298K is -20.1 kJ/mol, which is equivalent to -8.1 $k_B T$: a relative probability of $3 \cdot 10^{-4}$. We would need to wait a long time for that transition to occur spontaneously, and even longer to get good statistics on it.

Because of this probability issue, free energy methods rely on one basic idea: to force the system to where it doesn't want to be, and then measure by how much it doesn't want to be there. In free energy perturbation methods, we force the system by coupling the interaction strength between a molecule of interest and the rest of the system to a variable λ:

$$E_{total} = E_{ligand\text{-}ligand} + E_{rest\text{-}rest} + \lambda E_{ligand\text{-}rest}$$

and we slowly turn λ from 1 to 0. This means we can effectively turn off a molecule, and pretend that it is in vacuum (at λ=0): we force the system to where it doesn't want to be (either in the solvated or in the vacuum state, depending on what the sign of the free energy difference is). We'll then use the BAR method of calculating by how much it doesn't want to be there.

Coupling and de-coupling in this way helps us with calculating the free energy of binding, because we can now create a two-step path:



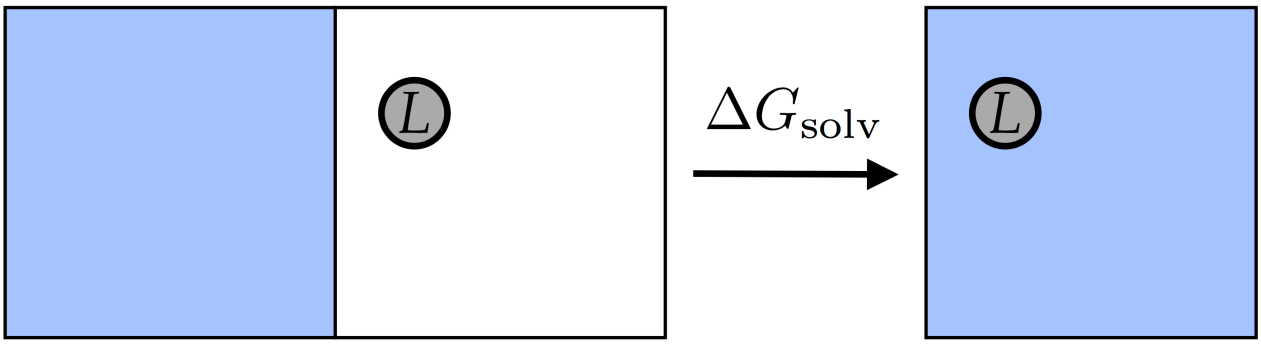
where we first de-couple the ligand from the solvent, and then re-solvate the ligand in the presence of the protein. The free energy of binding is thus

$$\Delta G_{binding} = \Delta G_1 + \Delta G_2$$

and the simulation is split into two parts: one calculating the de-solvation free energy, and one involving the free energy of coupling of a molecule into the system with a protein. That last simulation couples the ligand from λ=0 where it doesn't interact with the system, to the situation at λ=1, where the protein is bound to the ligand. The first simulation is the inverse of a free energy of solvation. This is the one we'll concentrate on in this tutorial - partly for computational performance reasons: because there is no protein involved, the simulation box size can be small and the simulations will be fast.

Free energy of solvation

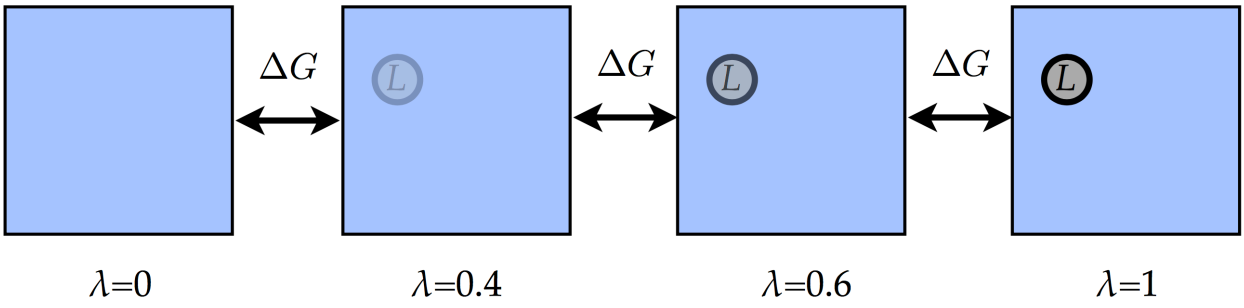
To calculate a free energy of solvation, we calculate -ΔG_1 in the picture above, or, equivalently, ΔG_{solv} in this picture:



We'll do this coupling our molecule to a variable λ (see equations above) and Bennett Acceptance Ratio (BAR) calculations, as built into GROMACS.

The BAR method relies on the output of pairs of simulations, say at state λ_A and λ_B. The free energy difference can be calculated directly if λ_A and λ_B are close enough (see Bennett's original article: Bennett, *J. Comp. Phys,* (1976) vol. 22 p. 245 for details), by calculating the Monte Carlo acceptance rates of transitions from λ_A to λ_B and vice versa, mapping states from λ_A to λ_B. The term 'close enough' here means that switching between the two states should be possible in both directions: some of the same configurations should be allowed in both end points (i.e. they should share some parts of phase space).

The most obvious points for λ_A and λ_B would be λ_A =0 and λ_B=1. These end points, however, usually have very few states in common: they share very little phase space. Because of this, the free energy would never converge to a usable value. That's why we'll split up the problem:



with as many λ points as are needed. We will therefore effectively 'slowly' turn on (or off) the interactions between our ligand and the solvent. This means that we need to run as many simulations as there are λ points, that we need to tell each simulation which neighboring λ points there are, and that we will post-process the results combining the results of many simulations (we

will use 7 λ points: 0, 0.2, 0.4, 0.6, 0.8, 0.9 and 1). As an example, we will run one simulation at λ=0.4, and that simulation will calculate the energy differences between its λ point and the neighboring points λ=0.2 and λ=0.6.

We will take one shortcut: we will turn off both the electrostatic (Coulomb) interactions and the Van der Waals (Lennard-Jones) interactions at the same time. For high-quality results, these stages are normally separated, but here we will do them both at the same time for expediency. Gromacs uses 'soft-core' interactions to make sure that while the normal (Lennard-Jones and Coulomb) interactions are being turned off, there will never be two point charges sitting on top of each other: this is achieved by turning on an interaction that effectively repels particles at intermediate λ points (in such a way that it cancels out from the free energy difference).

Preparing the system

We will start with a topology that can be downloaded from http://www.gromacs.org/Documentation/Tutorials/Free_energy_of_solvation_tutorial: get the archive file **gromacs-free-energy-tutorial.tar.gz** from the bottom of the page, or do

wget http://www.gromacs.org/@api/deki/files/261/=gromacs-free-energy-tutorial.tgz

Then extract the archive with
tar xzvf gromacs-free-energy-tutorial.tar.gz

and look for a file named **topol.top**, and a very basic coordinate file named **ethanol.gro**. This topology uses the OPLS force field and defines a methane molecule, and includes the definitions for SPC/E water.

Question: Take a look at the topology file **topol.top**. For the ethanol molecule definition, can you find which atoms are there, and how they are connected? We got the ethanol parameters by borrowing a threonine side chain…

We will first prepare the simulation box: the original configuration file has a dummy simulation box associated with it (you can see that by looking at the file **ethanol.gro**). We do this with:

```
gmx editconf -f ethanol.gro -o box.gro -bt dodecahedron -d 1
```

which sets up the simulation box. In this case, it will make the simulation box a rhombic dodecahedron with a minimum distance between the solute (the ethanol molecule) and the box edge of 1nm. The box is a rhombic dodecahedron because it provides a more effective packing of periodic images than rectangular boxes: we can use fewer waters for the same distance between periodic images of the ethanol molecule. See the Gromacs manual for illustrations of this box shape and how its periodic images are arranged.

Next, we solvate the system in water

```
gmx solvate -cp box.gro -cs -o solvated.gro -p topol.top
```

This should generate a system with ~300 water molecules taken from the default file name of the **-cs** option: a box of equilibrated water molecules.

To make the configuration suitable for simulation, we will first minimize its energy, twice: once with flexible bonds, and once with constrained bonds. For the minimization we will use the following settings (see the included file **em.mdp**)

```
------em.mdp------
integrator                = steep
nsteps                    = 500
coulombtype               = pme
vdw-type                  = pme
------------
```

Run the equilibration by preprocessing the input files into a run file with

```
gmx grompp -f em.mdp -c solvated.gro -o em.tpr
```

which generates the run file **em.tpr**. Run this file with

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

Since this is a very small system, you might need to play around with the number of MPI ranks (used for domain decomposition) or the number of OpenMP threads. You can for instance add **-nt 4** to limit the number of threads (processor cores) to four, because the simulated system with ~1000 atoms is too small to support highly parallel runs. How many cores do you manage to run it on? If you find that a nuisance and have resources to spare, you can also create a larger water box initially.

Global equilibration

We are now ready to equilibrate the system thermally. For this we will turn on pressure and temperature coupling: we're trying to calculate the difference in Gibbs free energy, and for that, the system must maintain temperature, but also pressure, while the ethanol molecule is de-coupled. The global equilibration (i.e. the equilibration done before we impose several different values) is done with **equil.mdp**:

```
------equil.mdp------
integrator                = md
nsteps                    = 20000
dt                        = 0.002
nstenergy                 = 100
rlist                     = 1.0
nstlist                   = 10
vdw-type                  = pme
rvdw                      = 1.0
coulombtype               = pme
rcoulomb                  = 1.0
fourierspacing            = 0.12
constraints               = all-bonds
tcoupl                    = v-rescale
tc-grps                   = system
tau-t                     = 0.2
```

```
ref-t                    = 300
pcoupl                   = berendsen
ref-p                    = 1
compressibility          = 4.5e-5
tau-p                    = 5
gen-vel                  = yes
gen-temp                 = 300
------------
```

We'll be using the v-rescale thermostat, and the Berendsen barostat. We run with

```
gmx grompp -f equil.mdp -c em.gro -o equil.tpr
gmx mdrun -deffnm equil -v
```

As before, you might need to limit the number of cores used. After this we should be ready with a – hopefully – equilibrated configuration of ethanol in water in a minute. The name of the output configuration is **equil.gro**.

Question: You should check whether the system has been equilibrated. How could you do this?

Creating the λ points

After the equilibration is done, we are ready to split the system into different λ points. To make this easier, We've prepared a script named **mklambdas.sh**, that can be found in the archive file mentioned before. For the final run, we'll use the run settings **run.mdp**:

```
------run.mdp------
; we'll use the sd integrator with 100000 time steps (200ps)
integrator               = sd
nsteps                   = 100000
dt                       = 0.002
nstenergy                = 1000
nstlog                   = 5000
; cut-offs at 1.0nm
rlist                    = 1.0
dispcorr                 = EnerPres
vdw-type                 = pme
rvdw                     = 1.0
; Coulomb interactions
coulombtype              = pme
rcoulomb                 = 1.0
fourierspacing           = 0.12
; Constraints
constraints              = all-bonds
; set temperature to 300K
tcoupl                   = v-rescale
tc-grps                  = system
tau-t                    = 0.2
ref-t                    = 300
; set pressure to 1 bar with a thermostat that gives a correct
; thermodynamic ensemble
pcoupl                   = parrinello-rahman
```

```
ref-p                    = 1
compressibility          = 4.5e-5
tau-p                    = 5

; and set the free energy parameters
free-energy              = yes
couple-moltype           = ethanol
; these 'soft-core' parameters make sure we never get overlapping
; charges as lambda goes to 0
sc-power                 = 1
sc-sigma                 = 0.3
sc-alpha                 = 1.0
; we still want the molecule to interact with itself at lambda=0
couple-intramol          = no
couple-lambda1           = vdwq
couple-lambda0           = none
init-lambda-state        = $LAMBDA$
; These are the lambda states at which we simulate
; for separate LJ and Coulomb decoupling, use
fep-lambdas              = 0.0 0.2 0.4 0.6 0.8 0.9 1.0
------------
```

where there is an extensive section on the free energy settings. Some values look like **$LAMBDA$**: these will be substituted by the script **mklambdas.sh**. The free energy settings state the following: take the molecule ethanol, and couple it to our variable λ. This is done so that λ =0 means that the molecule is de-coupled, and λ =1 means that the molecule is fully coupled (vdwq means Lennard-Jones + Coulomb). The **sc-power**, **sc-sigma** and **sc-alpha** settings control the 'soft-core' interactions that prevent the system from having overlapping particles as it is de-coupled.

The only things that still need to be set are the actual λ value **init-lambda** and the **foreign-lambda** value: that field determines for which other λ values the simulation should calculate energy differences. For our purposes, we will just calculate energy differences to all other λ values and keep this the same for all simulations (the performance impact of this is negligible).

The script **mklambdas.sh** will create a directory for each lambda point, and fill it with the configuration, topology and simulation settings, while substituting the **$LAMBDAS$** and **$ALL_LAMBDAS$** placeholders in the mdp file with actual values. We can run it as:

```
./mklambdas.sh run.mdp topol.top equil.gro
```

which should generate a number of directories:

lambda_00
lambda_01
lambda_02
lambda_03
lambda_04
lambda_05
lambda_06

which each have contents:

conf.gro
grompp.mdp
topol.top
Verify that the substitution worked correctly with

```
grep init-lambda lambda_*/grompp.mdp
```

which should show something like

lambda_00/grompp.mdp:init-lambda-state = 0
lambda_01/grompp.mdp:init-lambda-state = 1

etc. for each λ point. We now need to pre-process each run with

```
cd lambda_00
gmx grompp
cd ../lambda_01
gmx grompp
....
```

Check the output for whether these are successful, just to be sure. At this point we are ready to run. The total run time will be mere minutes on a computer for each λ point. This means that we can run the jobs sequentially, but then we'll have to wait a bit and we're wasting a big opportunity for parallelization.

Instead, we're going to try to run them in parallel and assume that we have some kind of batch system that we can submit jobs to. Because the system only has 1000 particles, scaling can be difficult. Typically, a modern compute cluster might have 32-64 cores or more per node. We will therefore trick the queue system to run multiple jobs.

If you have GROMACS compiles with a proper MPI library (i.e., not the built-in thread-MPI), you can use the -multidir option to gmx mdrun and specify all the different directories. Play around a bit and see what happens! Can you mix/match using multiple MPI ranks (one per simulation), and many cores for each such simulation?

Another alternative (in particular if you don't have any MPI version of GROMACS) is to simply start multiple jobs inside a single batch script. The exact settings to use in the batch system settings (usually **#PBS** or **#SBATCH** depending on the batch system in use) fields depends strongly on the system you're running on, so the example here probably won't work (modify it for your system).

#!/bin/sh
#SBATCH -J fe
#SBATCH -e run1.stderr
#SBATCH -o run1.stdout
#SBATCH --mem-per-cpu=1000
#SBATCH -t 00:10:00
 # One node of 12 cores per job:

```
#SBATCH -N 1
#SBATCH -n 12
module load gromacs  # add suitable modules such as cuda, fftw, etc.
( cd lambda_0; gmx mdrun -nt 4 >& run.log ) &
( cd lambda_01; gmx mdrun -nt 4 >& run.log ) &
( cd lambda_02; gmx mdrun -nt 4 >& run.log ) &
wait  # wait for all background tasks to finish
```

which will run three 4-threaded (set with **-nt 4**) versions of **mdrun**, each in their own directory, for a maximum of 10 minutes. Make as many of these scripts as needed for all the different λ-values (i.e. 3), and submit them to the queue.

Post-processing: extracting the free energy

After the simulations are done, we can extract the full free energy difference from the output data. Check your directories **lambda_00** to **lambda_06** for files called **dhdl.xvg**. These contain the energy differences that are going to be used to calculate the free energy difference. Combine them into a free energy with the Gromacs BAR tool **gmx bar**:

```
gmx bar -b 100 -f lambda_*/dhdl.xvg
```

This is pure magic. As part of the free energy code in mdrun, we have already calculated the offset enthalpies to the adjacent λ points, so this is already available in the dhdl.xvg file (together with all information about the point itself, what simulation it was, etc) - no need to rerun any simulations or store the entire trajectories. Second, the gmx bar command will do all the complicated processing needed for Bennett Acceptance Ratio free energies and just give you the results.

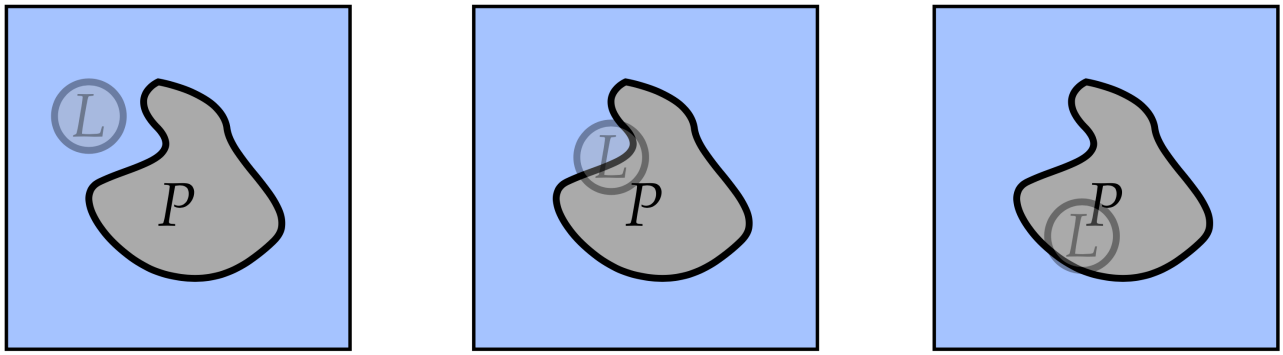
Where the **-b 100** means that the first 100 ps should be disregarded: they serve as another equilibration, this time at the conditions of the simulation. You should get a free energy difference of approximately -20.6 +/- 2.4 kJ/mol (this may be different if you run on different hardware: this answer is from a standard x86_64 cluster). This should be compared to an experimental value of -20.9 kJ/mol.

Question: Longer runs will change the free energy value a bit as the standard error estimate shrinks (try it). Exactly what value you get will depend on a number of settings, such as whether you use LJ-PME. Why can there sometimes be a significant (i.e. bigger than the estimated error) difference between the experimental result and the simulation result? How could this be improved?

Question: Look at the error bars for the individual λ points: they vary a lot between individual point pairs. What does this mean for the efficiency for the overall calculation? How could it be improved?

Where to go from here

After calculating the free energy of solvation, we've solved the first part of the free energy of binding of the earlier equations. The second part involves coupling a molecule into (or out of) a situation where it is bound to a protein. This introduces one additional complexity: we end up with a situation where a weakly coupled ligand wanders through our system:



which is bad because this is a poorly reversible situation: there are suddenly very few states that map from a weakly coupled to a more strongly coupled molecule, which will drastically reduce the accuracy of the free energy calculation.

This situation can be remedied by forcing the ligand to stay at a specific position relative to the protein. This can be done with the Gromacs ‘pull code’, which allows the specification of arbitrary forces or constraints onto with respect to centers of mass of any chosen set of atoms onto any other group of atoms. With a pull type of ‘umbrella’, we can specify that we want a quadratic potential to this specified location, forcing the ligand to stay at its native position even when it has been fully de-coupled.

One way find out where to put the center of the force is by choosing a group of atoms in the protein close to the ligand, and doing a simulation with full ligand coupling, where the pull code is enabled, but with zero force. The pull code will then frequently output the coordinates of the ligand, from which an average position and an expected deviation can be calculated. This can then serve as a reference point for the location of the center of force for the pull code during the production runs, and the force constant of the pull code.

Once the free energy has been calculated, care must be taken to correct for the fact that we have trapped our molecule. This can easily be done analytically.

Optional Question: Given a measured standard deviation in the location of the center of mass of our ligand, how do we choose the force constant for the pull code?

Optional Question: How do we correct for using the pull code: what is the contribution to the free energy of applying a quadratic potential to a molecule?