CECAM Tutorial Coarse-Grained Biomolecular Modeling

Lausanne, Switzerland

October 17-21, 2011

In this tutorial, you will use the program package gromacs. To make sure that all gromacs commands are known, source the code:

```
> source /usr/local/gromacs/bin/GMXRC
```

You can find more on the MARTINI force-field plus some additional topology files at the MARTINI homepage:

http://www.cgmartini.nl/

as well as in the main published papers.[1] The tutorial is available also from this website, including links to downloadable material. You can also find downloadable material on the CECAM website for this tutorial (after logging in), and the files are also available on the CECAM system under:

```
/nfs_home/tutoadmin/CGBM/MARTINITUTORIAL
```

[1]**S.J. Marrink, A.H. de Vries, and A.E. Mark** Coarse grained model for semiquantitative lipid simulations *J. Phys. Chem. B 108, 750-760 (2004)* and **S.J. Marrink, H.J. Risselada, S. Yefimov, D.P. Tieleman, and A.H. de Vries** The MARTINI force field: Coarse grained model for biomolecular simulations *J. Phys. Chem. B, 111, 7812-7824 (2007)*

MARTINI Model for Lipids

In this tutorial, the aim is to create and study properties of coarse-grained models of lipid bilayer membranes. First, we will attempt to spontaneously self-assemble a DSPC bilayer and check various properties that characterize lipid bilayers, such as the area per lipid, bilayer thickness, diffusion, and P_2 order parameters. Next, we will change the nature of the lipid head groups and tails, and study how that influences the properties. Finally, there is a slightly more advanced exercise on how to parameterize a MARTINI CG model of diC18:2-PC, a PC lipid with a doubly unsaturated tail, based on an all-atom simulation.

1 Spontaneous assembly: topology files and system setup

Copy and unpack the lipid-tutorial.tar.gz archive (it expands to a directory called lipid-tutorial). We will begin with self-assembling a DSPC (distearoyl-phosphatidylcholine) bilayer from a random configuration of lipids and water in a simulation box.

1. Enter the spontaneous-assembly subdirectory; first, create this random configuration of 128 DSPC starting from a single DSPC molecule:

   ```
   > genbox -ci dspc_single.gro -nmol 128 -box 7.5 7.5 7.5 -try 500
   -o 128_noW.gro
   ```

2. Minimize the system:

   ```
   > grompp -f em.mdp -c 128_noW.gro -p dspc.top
   > mdrun -v -c 128_minimised.gro
   ```

3. Add 6 CG waters (*i.e.*, 24 all-atom waters) per lipid:

   ```
   > genbox -cp 128_minimised.gro -cs water.gro -o waterbox.gro
   -vdwd 0.21 -maxsol 768
   ```

   The value of the flag -vdwd (default van der Waals radii) has to be increased from its default atomistic length (0.105 ns) to a value reflecting the size of Martini coarse-grained beads.

4. Minimize again (adapt dspc.top to reflect the water beads added to the system):

   ```
   > grompp -f em.mdp -c waterbox.gro -p dspc.top
   > mdrun -v -c minimised.gro
   ```

2 Spontaneous assembly: simulation

Now, you are ready to run the self-assembly MD simulation. About 25 ns should suffice.

```
> grompp -f md.mdp -c minimised.gro -p dspc.top
```

```
> mdrun -rdd 1.6
```

This might take ∼50 min on a machine at CECAM. You may want to check the progress of the simulation to see whether the bilayer has already formed before the end of the simulation. You may do this most easily by starting the inbuilt gromacs viewer:

```
> ngmx
```

In the meantime, have a close look at the martini parameter files martini_v2.1.itp and martini_v2.0_lipids.itp.
In particular, the interactions and bead types. What are the differences between PC and PE head groups as well as between saturated and unsaturated tails?

Check whether you got a bilayer. If yes, check if the formed membrane is normal to the z-axis (*i.e.*, membrane in the xy-plane). If the latter is not the case, you can decide to rotate the system accordingly (with editconf). In case you did not get a bilayer at all, or you do not want to wait until the self-assembly is complete, continue with the pre-formed one from the dspc_bilayer.gro file. (You may need to adapt the number of water in your .top file!)

Continue the simulation for another 15 ns at zero surface tension (switch to semi-isotropic pressure coupling). You will find how to change pressure coupling in the gromacs manual:

http://manual.gromacs.org/current/

Again, you may not want to wait for this simulation to finish. In that case, you may skip ahead and use a trajectory provided in the subdirectory bilayer-analysis/dspc .

3 Spontaneous assembly: analysis

From the latter simulation, we can calculate properties such as

- area per lipid
- bilayer thickness
- lateral diffusion of the lipids
- P_2 order parameters of the bonds

In general, for the analysis, you might want to discard the first few ns of your simulation (equilibration time).

3.1 Area per lipid

To get the *projected* area per lipid, you can simply divide the area of your simulation box ("Box-X" times "Box-Y" from g_energy) by half the number of lipids in your system[2].

3.2 Bilayer thickness

To get the bilayer thickness, use g_density. You can get the density for a number of different functional groups in the lipid (*e.g.*, phosphate and ammonium headgroup beads, carbon tail beads, *etc.*) by feeding an appropriate index file to g_density (make one with make_ndx; you can select, *e.g.*, the phosphate beads by typing "a P*"). The bilayer thickness you can obtain from the distance between the headgroup peaks in the density profile.

A more appropriate way to compare to experiment is to calculate the electron density profile. The g_density tool also provides this option. However, you need to supply the program with a data-file containing the number of electrons associated with each bead (option -ei electrons.dat). The format is described in the gromacs manual.

Compare your results to those from small-angle neutron scattering experiments:[3]

- area per lipid $= 0.65 \pm 0.05$ nm^2
- bilayer thickness $= 4.98 \pm 0.15$ nm

[2]Note that this might not be strictly OK, because the self-assembled bilayer might be slightly asymmetric in terms of number of lipids per monolayer, *i.e.*, the areas per lipid are different for the two monolayers. However, to a first approximation, we will ignore this here.

[3]**P. Balgavy, M. Dubnickova, N. Kucerka, M.A. Kiselev, S.P. Yaradaikin, and D. Uhrikova** Bilayer thickness and lipid interface area in unilamellar extruded 1,2-diacylphosphatidylcholine liposomes: a small-angle neutron scattering study *Biochim. et Biophys. Acta 1512, 40-52 (2001)*

3.3 Lateral diffusion

Before calculating the lateral diffusion, remove jumps over the box boundaries (`trjconv -pbc nojump`). Then, calculate the lateral diffusion using `g_msd`. Take care to remove the overall center of mass motion (`-rmcomm`), and to fit the line only to the linear regime of the mean-square-displacement curve (`-beginfit` and `-endfit` options of `g_msd`). To get the lateral diffusion, choose the `-lateral z` option.

To compare the diffusion coefficient obtained from a MARTINI simulation to a measured one, a conversion factor of about 4 has to be applied to account for the faster diffusion at the CG level due to the smoothened free energy landscape.

3.4 Order parameters

Now, we will calculate the (second rank) order parameter, which is defined as

$$P_2 = \frac{1}{2}\left(3\left\langle\cos^2\theta\right\rangle - 1\right)$$

where θ is the angle between the bond and the bilayer normal. $P_2 = 1$ means perfect alignment with the bilayer normal, $P_2 = -0.5$ anti-alignment, and $P_2 = 0$ random orientation.

The script to calculated these order parameters is located in the subdirectory `bilayer-analysis/scripts`. Copy the `.xtc` and `.tpr` files in the `bilayer-analysis/dspc` subdirectory (should be named `traj.xtc` and `topol.tpr`, respectively; you may want to use the 30 ns simulations already present there instead). The script `do-order.py` will calculate P_2 for you. As it explains when you invoke it, it needs a number of arguments. The command:

```
> ./do-order.py 0 10000 20 0 0 1 64 DSPC
```

will for example read in a 10 ns trajectory of 64 DSPC lipids and average over 20 equally-spaced snapshots. P_2 is calculated relative to the z-axis.

ADVANCED

4 Different bilayers: unsaturated tails, different head groups

Lipids can be thought of as modular molecules. In this section, we investigate the effect of changes in the lipid tails and in the headgroups on the properties of lipid bilayers using the MARTINI model. We will **i)** introduce a double bond in the lipid tails, and **ii)** change the lipid head groups from PC to PE.

4.1 Unsaturated tails

To introduce a double bond in the tail, we will replace the DSPC lipids by DOPC. Compare the MARTINI topologies of these two lipids. Replace DSPC by DOPC in your .top and .mdp files, and grompp will do the rest for you (in this case, you can ignore the "atom name does not match" warnings of grompp).

4.2 Changing the headgroups

Starting again from the final snapshot of the DSPC simulation, change the head groups from PC to PE.

For both new systems, run 15 ns MD simulations (or use the trajectories present in the bilayer-analysis subdirectory) and compare the above properties (area per lipid, thickness, diffusion, order parameter) between the three bilayers (DSPC, DOPC, DSPE). Do the observed changes match your expectations? Why/why not? Discuss.

ADVANCED

5 Refine CG parameters based on AA simulation

This section uses the fine-grained-to-coarse-grained (FG-to-CG) transformation tool, which is treated in section 8 of the MARTINI *tutorial. If you are unfamiliar with the tool, you might want to skip this section first and continue with the rest of the tutorial.*

In this part, we will try to obtain improved MARTINI parameters for diC18:2-PC, a PC lipid with a doubly unsaturated tail. We will try to optimise the MARTINI parameters such that the dihedral and angle distributions within the lipid tail match those from a 100 ns all-atom simulation (all-atom data in fg.xtc) as closely as possible. The files needed in this section are located in the refine-parameters subdirectory.

The task can be divided into the following five steps:

1. Transform the all-atom (fine-grained, FG) trajectory into its coarse-grained counterpart, based on a pre-defined mapping scheme[4].

[4]The mapping is already defined, look at the [mapping] section in dupc_fg.itp.

2. Calculate the angle and dihedral distributions. These will serve as the reference ("gold standard") later.
3. Do a CG simulation of the system.
4. Calculate the angle and dihedral distributions, and compare to the reference.
5. Revise the relevant CG parameters, repeat steps 3 and 4 until you are satisfied with the result.

For the transformation (or mapping) from FG to CG, we will use the program g_fg2cg, which is part of a modified version of gromacs that you can source from here:

```
> source /nfs_home/tutoadmin/CGBM/MARTINITUTORIALFILES/gromacs-3.3.1/bin/GMXRC
```

1. First, transform from FG to CG:

   ```
   > g_fg2cg -pfg fg.top -pcg cg.top -c fg.gro -n 1 -o cg.gro
   > g_fg2cg -pfg fg.top -pcg cg.top -c fg.xtc -n 1 -o fg2cg.xtc
   ```

2. Then, make an index file needed for the calculation of the angle and dihedral distributions within the lipid tail:

   ```
   > make_ndx -f cg.gro -o angle.ndx
   a GL1 | a C1A | a D2A
   a C1A | a D2A | a D3A
   a D2A | a D3A | a C4A
   a C1A | a D2A | a D3A | a C4A
   q
   ```

3. Now, calculate the distributions with g_angle:

   ```
   > g_angle -f fg2cg.xtc -type angle -n angle.ndx -od fg2cg_ang{1,2,3}.xvg
   > g_angle -f fg2cg.xtc -type dihedral -n angle.ndx -od fg2cg_dih1.xvg
   ```

These are the target distributions that we want to obtain with the CG model. As a starting point, we will use the MARTINI parameters as defined in dupc_cg.itp, *i.e.*, all angles at 180 degrees. Carry out a short CG simulation (starting from cg_mdstart.gro, you just have to add water to the cg.top; don't forget to source back the regular version of gromacs!). You will need to make an appropriate .mdp file. Note that the FG trajectory was obtained at a temperature of 300 K. After comparing the angle and dihedral distributions to the FG-to-CG reference, change the angle parameters in dupc_cg.itp and repeat until satisfied.

Martini Model for Proteins

In this tutorial you will learn to set up and run a Martini CG protein simulation starting from a PDB structure. The Martini model does not guarantee correct and stable folding of proteins and peptides. Two alternatives are presented here to add (selected) interactions to the model to ensure structural integrity of proteins with the Martini model. Additionally, advanced exercises will show you how to transform a CG model to an atomistic one using a simulated annealing approach. Furthermore, there is a tutorial on the recently added feature of polarizable Martini water. Finally, there is a tutorial on how to visualize Martini model proteins using vmd.

Keeping in line with the overall Martini philosophy, the coarse-grained protein model groups ∼4 heavy atoms together in one CG bead. Each residue has one backbone bead and zero or more side-chain beads depending on the amino acid type. The secondary structure of the protein influences both the selected bead types and bond/angle/dihedral parameters of each residue as explained in [5]. It is noteworthy that, even though the protein is free to change its tertiary arrangement, local secondary structure is pre-defined and thus imposed throughout a simulation. Conformational changes that involve changes in the secondary structure are therefore beyond the scope of Martini CG proteins.

Setting up a CG protein simulation consists basically of two steps:

1. converting an atomistic protein structure into a CG model;
2. generating a suitable Martini topology.

Both steps are done using the publicly available martinize.py script, distributed via the Martini website and available also in the tutorial files.

6 Martini CG simulation of ubiquitin in water

The aim of this module is to set up a CG simulation of ubiquitin in a water box. The required files can be found in the archive `protein_tutorial.tar.gz`, expanding to `protein-tutorial`. In this part of the tutorial, some basic knowledge of gromacs commands is assumed and not all commands will be given explicitly. Please refer to the previous tutorials and/or the gromacs manual. After getting the atomistic structure of ubiquitin (1UBQ), you'll need to convert it into a CG structure and to prepare a Martini topology for it. Once this is done the CG structure can be minimized, solvated and simulated. The steps you need to take are roughly the following:

1. Download 1UBQ.pdb from the *Protein Data Bank* (or find it in the subdirectory `proteins`).

[5]**L. Monticelli, S.K. Kandasamy, X. Periole, R.G. Larson, D.P. Tieleman, and S.J. Marrink** The MartiniCoarse-Grained Force Field: Extension to Proteins *J Chem. Theory Comput., 819-834, 2008*

2. The pdb-structure can be used directly as input for the martinize.py script, to generate both a structure and a topology file. Have a look at the help function (*i.e.* run martinize.py -h) for the available options. Hint: During equilibration it might be useful to have (backbone) position restraints. The final command might look a bit like this:

   ```
   > ./martinize.py -f 1UBQ.pdb -o system.top -x cg_1UBQ.pdb
   -dssp /wherever/dsspcmbi -p backbone
   ```

   When using the -dssp option you'll need the dssp binary, which determines the secondary structure classification of the protein back-bone from the PDB-structure. It can be downloaded from a CMBI website:

   http://swift.cmbi.ru.nl/gv/dssp

   A working binary can also be found at
   /nfs_home/tutoadmin/CGBM/MARTINITUTORIALFILES/dsspcmbi.
   As an alternative, you may prepare a file with the required secondary structure yourself and feed it to the script:

   ```
   > ./martinize.py -f 1UBQ.pdb -o system.top -x cg_1UBQ.pdb
   -ss your_sec_struct_file -p backbone
   ```

   For examples see the .ssd-files in the subdirectory proteins.

3. If everything went well, the script generated three files: a coarse grain structure (.gro/.pdb), a master topology file (.top), and a protein topology file (.itp). In order to run a simulation you need two more: the Martini topology file (martini_v2.1.itp) and a run parameter file (.mdp). You can get examples from the Martini website or from the protein-tutorial package. Don't forget to adapt the settings where needed!

4. Do a short (∼10 steps!) minimization in vacuum. (Before you can generate the run-input files with grompp, you have to generate a box, for example using editconf).

5. Solvate the system with genbox (water-1bar-303K.gro is an equilibrated water box) and minimize. Make sure the box-size is large enough (i.e. there is enough water around the molecule) and remember to use a larger van der Waals distance when solvating to avoid clashes, *e.g.*:

   ```
   > genbox -cp confout.gro -cs water-1bar-303K.gro -vdwd 0.21 -o solvated.gro
   ```

6. Do a short energy minimization and position restrained simulation. Since the martinize.py script already generated position restraints, all you have to do is specify "define = -DPOSRES" in your .mdp file and add the appropriate number of water (W) beads to your .top file.

7. Start production run (without position restraints!).

8. ...

9. PROFIT! What sort of analysis can be done on this molecule? Start by having a look at the protein with vmd (section Visualizing MARTINI systems using vmd).

A mapped fine-grained simulation of ubiquitin is available in the archive mapubq.tar.gz at the MARTINI website and available in the `protein_tutorial` archive.

7 Elastic networks

The aim of this module is to see how application of elastic networks can be combined with the MARTINI model to conserve tertiary and quartenary structures more faithfully without sacrificing realistic dynamics of a protein. We offer two alternative routes. Please be advised that this is an active field of research and that there is as of yet no "gold standard". The first option is to generate a simple elastic network on the basis of a standard MARTINI topology. The second options is to use an ELNEDYN network. This second option constitutes quite some change to the MARTINI forcefield and thus is a different forcefield! The advantage is that the behavior of the method has been well described.[6] Both approaches can be set up using the martinize.py script and will be shortly described below.

First you should simulate a normal MARTINI CG protein without an elastic network and then see what changes when you use a CG topology with an elastic network.

1. Copy HIV-1 Protease (1A8G.pdb).
2. Repeat steps 2–7 from the previous exercise. (You'll need to simulate the protein for hundreds of nanoseconds to see major changes in the structure, a sample long simulation is provided in the archive `HIVP_std.tar.gz`).
3. Visualize the simulation; look especially at the binding pocket of the protein: does it stay closed, open up, or what? What happens to the overall protein structure?

7.1 Martini + elastic network

The first option to help preserve higher-order structure of proteins is to add to the standard MARTINI topology extra harmonic bonds between non-bonded beads based on a distance cut-off. Note that in standard MARTINI, long harmonic bonds are already used to impose the secondary structure of extended elements (sheets) of the protein. The martinize.py script will generate harmonic bonds between backbone beads if the options -elastic is set. It is possible to tune the elastic bonds (*e.g.*: make the force constant distance dependent, change

[6] **X. Periole, M. Cavalli, S.J. Marrink, M.A. Ceruso**, Combining an Elastic Network With a Coarse-Grained Molecular Force Field: Structure, Dynamics, and Intermolecular Recognition.,*J. Chem. Theory Comput., 5, 2531-2543 (2009)*

upper and lower distance cut-off, *etc.*) in order to make the protein behave properly. The only way to find the proper parameters is to try different options and compare the behavior of your protein to an atomistic simulation or experimental data (NMR, *etc.*). Here we will use basic parameters in order to show the principle.

1. Use the martinize.py script to generate the coarse grain structure and topology as above. For the elastic network options use:

   ```
   -elastic -ef 500 -el 0.5 -eu 0.9 -ea 0 -ep 0
   ```

   This turns on the elastic network (-elastic), sets the elastic bond force constant to 500 kJ.mol^{-1}.nm^{-2} (-ef 500), the lower and upper elastic bond cut-off to 0.5 and 0.9 nm, respectively, (-el 0.5 -eu 0.9) and makes the bond strengths independent of the bond length (elastic bond decay factor and decay power, -ea 0 -ep 0, respectively; these are default). The elastic network is defined in the .itp file by an "ifdef" statement (have a look!). The "#define RUBBER_BANDS" in the .top or .itp file switches it on. Note that martinize.py does not generate elastic bonds between i→i+1 and i→i+2 backbone bonds, as those are already connected by bonds and angles.
2. Proceed as before and start a production run.

7.2 ElNeDyn

The second option to use elastic networks in combination with Martini puts more emphasis on remaining close to the overall structure as deposited in the PDB than standard Martini does. The main difference from the standard way (used in the previous exercise) is the use of a *global* elastic network between the backbone beads to conserve the conformation instead of relying on the angle/dihedral potentials and/or local elastic bonds to do the job. The position of the backbone beads is also slightly different. In standard Martinithe center-of-mass of the peptide plane is used as the location of the backbone bead, but in the elastic network implementation the positions of the Cα-atoms are used for this.

The martinize.py script automatically sets these options and sets the correct parameters for the elastic network. As the elastic bondstrength and the upper cut-off have to be tuned in an ElNeDyn network, these options can be set manually (-ef -eu).

1. Use the martinize.py script to generate the coarse grain structure and topology as above. Set the following options:

   ```
   -elnedyn -ef 500 -eu 0.9
   ```

   This selects the ElNeDyn topology options, with a force constant of 500 kJ.mol^{-1}.nm^{-2} (-ef 500) and an upper bond length cut-off of 0.9 nm (-eu 0.9).

2. Proceed as before and start a production run.

Now you've got three simulations of the same protein with different elastic networks. One of them might fit your needs in terms of structural and dynamic behavior. If not, there are an almost infinite number of ways to further tweak the elastic network!

An easy way to compare the slightly different behaviors of the proteins in the previous three cases is to follow deviation/fluctuation of the backbone during simulation (and compare it to an all-atom simulation if possible). vmd *provides a set of friendly tools to compute RMSD/RMSF, but it needs some tricks to be adapted to CG systems (*"protein"*,* "water"*,* etc. *keywords are not understood by* vmd *on CG structures). In this specific case, to visualize/select only the backbone you need to create a representation containing only the corresponding beads:* "name BB"*. See section 10 for more detailed howtos and advice.*

8 Reverse transformation

In this module we explore the use of the reverse transformation tool, to convert between CG and fine grain (FG) representations of a system.

Simulations at CG level allow the exploration of longer time-scales and larger length-scales than at the usual atomistic level. However, the loss of detail can seriously limit the questions that can be asked to the system. Methods that re-introduce atomic details in a CG structure are therefore of considerable interest. Such structures provide starting points in phase space for simulations at the more detailed level that may otherwise take too long to reach. As a demonstration of such a method, we will use restrained simulated annealing (SA) to increase the resolution of a system containing a WALP peptide spanning a DPPC bilayer. We will investigate how the number of steps allowed for the reconstruction influences the quality of the generated FG structure.

8.1 Reverse transformation: transformation run

In this part of exercise we will use a modified version of gromacs that allows one to generate a FG structure from CG beads[7]. The same version is also used to transform a FG structure to CG beads, section 5. Thus, the program allows one to switch between FG and CG representations. To achieve this, additional information is put in the topology file at the FG level in a section called `[ mapping ]`. The tool pdb2gmx can generate this mapping for proteins automatically. Note that this will usually require the option -missing to be specified! Two new functions in mdrun are responsible for restrained simulated annealing (SA) to converge from a random FG structure to a FG structure compatible with the CG structure, i.e. the FG positions are optimized in such a way that the CG structure is preserved in the FG-to-CG mapping (each CG

[7]**A.J. Rzepiela, L.V. Schäfer, N. Goga, H.J. Risselada, A.H. de Vries, S.J. Marrink**, Reconstruction of atomistic detail from coarse grained structures, *J. Comput. Chem., 31, 1333-1343 (2010)*

bead is at the center of mass of the FG atoms that map to it). There is also small tool, called g_fg2cg, to generate CG structures from FG ones defined by the mapping and random FG starting structures based on CG input file, that are afterwards optimized during a SA run. The software package is supplied in the archive gmx_rev.tar.gz and is also available from the MARTINI website.

1. Unpack the rev_trans.tar.gz in the MARTINITUTORIAL directory that contains all necessary gromacs files for this exercise.

2. Compile and/or source the modified version of gromacs (remember this tool is based upon gromacs version 3.3.1 and needs the corresponding tricks and threats to be compiled.) On the CECAM cluster it can be sourced directly:
   ```
   > source /nfs_home/tutoadmin/CGBM/MARTINITUTORIALFILES/
   /gromacs-3.3.1/bin/GMXRC
   > export GMXLIB=/nfs_home/tutoadmin/CGBM/
   /MARTINITUTORIALFILES/gromacs-3.3.1/share/gromacs/top
   ```

3. Modify the FG fg.top file in such a way that the number of water and lipid molecules is the same as in the coarse-grained model. One FG_W corresponds to four normal water molecules.

4. Use g_fg2cg to construct an input atomistic structure for a simulated annealing run. The coarse grain structure is already prepared for you and is called cg.gro. Check the output with ngmx or VMD.
   ```
   > g_fg2cg -pfg fg.top -pcg dppc_cg.top -n 0 -c cg.gro -o fg.gro
   ```

5. Modify the fg.mdp file. Set the last non-water CG bead (see comments for additional mdp options at the bottom of fg.mdp file) to the one from your CG *.gro file. Leave the rest of options default.

6. Use grompp to create a topol.tpr file.

7. Perform a SA run by typing
   ```
   > mdrun -coarse cg.gro -v
   ```

8. Change the number of simulation steps to 1000 and SA time parameters to 0 1.5 0 1.5 0 1.5. Perform another SA run with the altered parameters.

9. Change the number of simulation steps to 5000 and SA time parameters to 0 7.5 0 7.5 0 7.5. Perform a SA run again.

10. If you have enough time, change the number of simulation steps to 60000 and SA time parameters to 0 100 0 100 0 100. Repeat the SA run.

11. Plot the potential energies for all runs and compare them.

12. Compare FG WALP models from all runs. Check secondary structure and ϕ and ψ angles.

13. Compare dihedral distribution for one of dihedral angles from the tail of the DOPC lipids.

9 Polarizable Water

In this part of the tutorial we will convert water in an existing Martini-system to polarizable water. The necessary files may be found in the archive martini_Pwater.tar.gz. In the polarizable Martini paper[8] the model is described as follows:

"*The polarizable CG water consists of three particles instead of one in the standard* Martini *force field. The central particle W is neutral and interacts with other particles in the system by means of the Lennard-Jones interactions, just like the standard water particle. The additional particles WP and WM are bound to the central particle and carry a positive and negative charge of +0.46 and -0.46, respectively. They interact with other particles via a Coulomb function only, and lack any LJ interactions. The bonds W-WP and W-WM are constrained to a distance of* $0.14nm$*. The interactions between WP and WM particles inside the same CG water bead are excluded, thus these particles are "transparent" toward each other. As a result the charged particles can rotate around the W particle. A harmonic angle potential with equilibrium angle* $\theta = 0rad$ *and force constant* $K_h = 4.2kJ.mol^{-1}.rad^{-2}$ *is furthermore added to control the rotation of WP and WM particles and thus to adjust the distribution of the dipole momentum.*"

If you want to solvate a new system with polarizable water, follow the steps in the protein part of the Martini tutorial, section 6. Instead of a normal waterbox and .mdp/.itp files use the waterbox containing polarizable water and polarizable Martini .mdp and .itp files (available from the Martini website: http://cgmartini.nl). Note that minimizing a polarizable Martini system requires some tweaking, as described below.

If you have an existing system with normal Martini water and want to change to polarizable water you may use the python script triple-w.py.

1. Create a new gro file:

   ```
   > triple-w.py dppc_bilayer.gro
   ```

   The python-script triple-w.py adds positive and negative sites at a small distance to every central water bead in a .gro file.

2. Adapt the .top-file. Make sure the polarizable version of the particle definition martini_v2.P.itp-file is included. In the .mdp file the value of epsilon_rc has to be adapted and the index group W renamed into PW. The .itp files for the lipids and possibly other molecules do not have to be changed.

3. If polarizable water is used in combination with proteins or peptides, all `AC1` and `AC2` beads have to be replaced by normal `C1` and `C2` beads. `AC1` and `AC2` are obsolete in polarizable Martini.

4. Minimize the system. For polarizable water to minimize without problems, the constraints have to be changed to stiff bonds. Using the `ifdef`-

[8]**S.O. Yesylevskyy, L.V. Schäfer, D. Sengupta, S.J. Marrink**, Polarizable Water Model for the Coarse-Grained MARTINI Force Field, *PLoS Comput. Biol. 6, e1000810, 2010*

statement in the martini_v2.P.itp file, this can be set using a define value in the .mdp file (have look at the bottom of martini_v2.P.itp to see how it works).

5. Set the correct options in martini_v2.P_example.mdp (e.g. `integrator = steep`, `nsteps = 50`, `costraints = none`, `epsilon_r = 2.5`, *etc.*) and the 'define' option to -DEM and change the W index group name.

6. Generate the input files for the minimization run:

```
> grompp -f martini_v2.P_example.mdp -c dppc-polW.gro
-p dppc_bilayer.top -o em -maxwarn 1
```

7. Proceed with equilibration and production runs.

10 Visualizing Martini systems using vmd

vmd *is a molecular visualization program for displaying, animating, and analyzing large biomolecular systems using 3-D graphics and built-in scripting.* (http://www.ks.uiuc.edu/Research/vmd/).

In this module, we explain some of the vmd commands that can be used to visualize the CG systems. Additionally, Tcl scripts are presented that assist with the visualization of CG systems; these scripts, as well as extended help, are available on the Martini website:

http://md.chem.rug.nl/cgmartini/index.php/downloads/tools/

10.1 Some basics

vmd adopted a representation-philosophy: for any set of atoms/molecules/protein chains we want to display or analyze, we need to select this set through a "representation" defined by keywords related to this set (somewhat similar to make_ndx). vmd comes with implemented keywords for all-atom systems ("protein", "chain", "hydrogen", "solvent", *etc.*). More general keywords are implemented to be able to display non-classical-systems (CG systems are part of this second category); you can find them in vmd manual:

http://www.ks.uiuc.edu/Research/vmd/current/ug/

Below are listed a few examples:

- To select only lipids: resname POPC, or only part of each lipids: resname POPC and name "C.*.A" "C.*.*.B" "D.*.A" "D.*.B" for lipid tails (C1A, C1B, *etc.*) and resname POPC and name NC3 PO4 "GL.*" for heads.
- To select only backbone beads of a protein: name BB (use "BB.*" or BAS for old versions of FG-to-CG scripts).
- To get rid of solvent (water/ion beads): not resname W WF ION or WP in case of polarizable water.
- To show only charged positively charged residues (in Martini): resname LYS ARG.
- To display the water shell around specific residues: within 7.0 of (index 531 to 538).
- To display all lipids (excluding DPPC) whose heads are interacting with the same specific residues: same resid as ((within 7.0 of (index 531 to 538)) and name NC3 PO4 "GL.*") and not resname DPPC.

As you can see in the examples, all these keywords can be mixed with the logical links: and, or, not, *etc.* to produce any representation. Try it yourself!

vmd is really demanding in term of memory; an easy trick to decrease the amount needed by vmd is to load structure/trajectories containing only beads needed by your analysis; this can easily be done by preprocessing the trajectory using trjconv. And as we are simulating bigger and bigger systems in which more and more beads are involved, the previous trick can be adapted to increase the displaying/seeking speed of trajectories by writing representations showing only beads needed for the visualization (head groups of a bilayer for instance).

Keep in mind that you can save the visualizing state of your system whenever you want by saving a state.vmd file. This file contains the Tcl commands needed to obtain the current display; lists of bonds and drawings (cylinders) are not saved, but you can open this .vmd file and manually add the lines you wrote to generate them at the bottom.

10.2 CG bonds/constraints and elastic networks

When a CG structure/trajectory is opened with vmd, the program builds a network of bonds using a distance criteria and an atomistic library of possible bond lengths (defined by the namd forcefield, developed by the same group); CG beads, linked with bonds with an average length of ∼0.35 nm, are not defined through this automatic algorithm. vmd inevitably ends up displaying a cloud of dots, which are hard (impossible?) to properly visualize with non-bionic human eyes. A Tcl script which reads the bonds and constraints from the CG .itp/.tpr files, and rewrites the CG bond network is available on the Martini website. First, make sure your system knows where to find the script:

```
> source /wherever/cg_bonds.tcl
```

This script can now be used from the command line window of vmd as follows:

```
> cg_bonds -top system.top -topoltype "elnedyn"
```

in case you have a .top available, but no gromacs. Alternatively, if gromacs is installed on the machine you are using, you may use a .tpr instead:

```
> cg_bonds -gmx my_gromacs/bin/gmxdump -tpr dyn.tpr
-net "elnedyn" -cutoff 12.0 -color "orange" -mat "AOChalky"
-res 12 -rad 0.1
```

The last line will draw the ElNeDyn network with the options (cutoff, color, material, resolution and radius) specified by extracting the bonds from the dyn.tpr file (that's where the gmxdump comes in). Note that you must specify a my_gromacs version compatible with the dyn.tpr-file. On the CECAM cluster, there is a gromacs version (4.0.7) installed under /usr/local/gromacs, but there is no gromacs installed on the workstations on which you run vmd. There you must use the .top (copy all the necessary files over).

10.3 Visualization of secondary structure

After being able to draw bonds and constraints defined by the CG forcefield, the next step is to see protein secondary structure. We are currently developing a graphical script drawing vmd cartoon-like representation. This set of routines is still under development, and needs to be improved... by your feedback?

Two main routines are provided by the cg_secondary_structure.tcl script: cg_helix and cg_sheet. Use these two commands in the same fashion:

```
> cg_whatever {list of termini} [-graphical options]
```

Or, in an example:

```
> cg_helix {{5 48} {120 146}} -hlxmethod "cylinder" -hlxcolor "red"
-hlxrad 2.5
```

which will draw two helices, from residue labeled as 5 to residue labeled as 48 and from residue labeled as 120 to residue labeled as 146, as red cylinder of radius 0.25 nm. Check the help shown when the script is sourced or the website for an exhaustive list of options and default values.

To define the list of termini, two options are implemented: **i)** providing the list by yourself (like in the example shown above), **ii)** reading/parsing a file generated by do_dssp. In the second case, you don't need to provide any termini, but the list of termini still needs to be written in the command line as an empty list: {}.

Please keep in mind that, due to the restricted amount of structural information carried on by a CG structure, the beauty and exactitude of these graphical representations are limited…