This tutorial series aims to teach you how to create, simulate, and analyze cyclic peptides using Chimera and GROMACS. It is meant to teach you how and why we complete each step of the process. Good luck!

**Building Structure using Chimera**

Before starting this tutorial, you will want to have some understanding of Chimera. Although you can somewhat google-as-you-go when you forget syntax, it is a good idea to build an understanding of what Chimera is able to do and how to use it. Because Chimera can be used for more complex purposes than what we’ll be using it for, it is possible to get overwhelmed by the online resources. We recommend starting off with the following:

Basic Functions: (Be sure to check out points 1–7, 17, 18, and 19 in the link below). The command line instructions in particular come in handy.

<https://www.cgl.ucsf.edu/chimera/current/docs/UsersGuide/framecore.html>

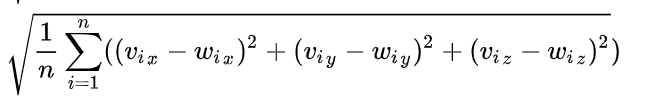
This tip sheet is a good place to find functions that might help you with a given task; you can then look at that function’s documentation for more information.

<https://www.cgl.ucsf.edu/chimera/docs/UsersGuide/quickref.pdf>

1. To begin, use Tools → Structure Editing → Build Structure to create an all-glycine linear peptide. The size of the peptide will depend on how big you want your eventual cyclic peptide to be. A common lab tutorial-sequence (as it is a cyclic peptide our group has found to be well-structured) is cyclo-(GNSRV), so let’s pretend we’re using that sequence for now. In that case, we want to create a sequence of GGGGG with random *φ*/*ψ* angles. We’ll eventually add in the actual amino acids that we want in place of the glycines, but Chimera sometimes gets confused when it is asked to cyclize chiral residues and may change chirality in the process.
2. Once you see your linear peptide, to form a cyclic peptide, delete the OXT atom of the C-terminus and add a bond between the C and N termini (the 5th and 1st residues in this example) using the command line in Chimera.
3. Energy-minimize your structure in Chimera (you can use the default options that Chimera provides for charges, etc.). At this point, you should get something that looks somewhat like a cyclic peptide. Typically, all amide bonds in our cyclic peptides are assumed to be in the *trans* conformation. Unfortunately, during this minimization step in Chimera, **a peptide bond might change into the *cis* configuration**. Carefully check the structure after energy minimization. If you see a *cis* bond in your peptide you’ll need to start over and provide new random *φ*/*ψ* angles.
4. Now you’ll want to swap in the actual residues that you want to use in place of the glycines that currently form your cyclic peptide using the command line in Chimera.
5. Lastly, GROMACS expects a given PDB structure not to have any hydrogens. For this reason, we have found that it is best to delete all hydrogens **except** the one connected to the N-terminus of the first residue. The reason for keeping this one hydrogen will be seen later - for now, it suffices to say that it gets around GROMACS’/force fields’ lack of native support for cyclic peptides.
6. Lastly, save your structure as a .pdb file.
7. For a given sequence, it is necessary to run two simulations, starting from two *different* structures. By comparing the similarity of these simulations over time, you can tell whether you have converged on a solution. So although you’ve now created one starting structure, you’ll need to do steps 1–6 again but using different randomized dihedrals to create your second structure.
8. Once you know how to do things by hand, you may use the provided GenInit script to automate the process. The script uses Chimera, MDAnalysis, and Python to automatically generate cyclic peptide structures. To use this script, clone the entire GenInit directory from this repository in a location where you want to run it. Ensure you have the correct dependencies (the README includes Tufts Research Cluster-specific information, but you really just need Python 3.7+, the MDAnalysis package, and Chimera – setting up a conda environment is recommended). An example usage of this script is:

python main.py -seq GNSRV -n 2 -thresh 1.2

**Alignment of the 2 configurations:**

After creating two (or more) initial confirmations (which you may see referred to as S1 and S2), you should make sure they are substantially different by calculating the backbone root-mean-squared-displacement using a tool like VMD. The backbone root-mean-squared-displacement (usually simplified to backbone RMSD) is a metric which indicates the separation between two structures’ backbones. Add up the squared displacement between all atoms that make up the protein backbones of interest in our initial structures:

*v* and *w* are the two structures, (*vix* *viy* *viz*) are the coordinates of atom *i* in structure *v,* (*wix* *wiy* *wiz*) are the coordinates of atom *i* in structure *w*. The summation is over all *n* backbone atoms in the structures.

To calculate the backbone RMSD, we first need to load our structures into VMD. To do so, run vmd -m, followed by the names of your two structures:

vmd -m s1.pdb s2.pdb

Then, in VMD, go to **Extension → Analysis → RMSD Trajectory Tool.** This will give you the ability to calculate the RMSD. Specifically, however, we want to calculate the backbone-aligned backbone RMSD. This means that we want to first align the protein backbones.

Check the **Backbone** box

Click on **ALIGN**

Then, to calculate the backbone RMSD of the two structures,

Click on **RMSD**

The output number will be the RMSD in units of Å.

Typically, you want the structures you come up with to be significantly distinct from each other. If they are not, you’ll need to create another structure (or possibly multiple other structures if they are all too similar). Because RMSD is protein-length dependent (longer proteins → larger RMSD), it is impossible to come up with a single threshold to use. Eyeballing a little and relying on visual differences is okay until you develop an intuition for what a high RMSD value is.

**Quiz!**

1. Briefly list the major steps you need to build an initial cyclic peptide in Chimera.
2. Why do you begin with all glycines and swap in your actual residues later?
3. What situations may lead to having to generate a new starting structure?
4. What atoms do you consider when aligning and calculating RMSD?
5. What does the RMSD signify and how is it calculated between two protein backbones?

**Generating Initial GROMACS Files**

**Altering the Amber99SB Force Field**

Now that we have our initial PDB structures, we want to convert them into files that GROMACS can understand. Unfortunately, GROMACS is designed for linear peptides, so we will need to go into the GROMACS library to create residues that can be cyclized.

Eventually, we want to be using the RSFF2 force field to simulate our cyclic peptides, since RSFF2 is optimized with a protein coil library. This is different from other force fields which aim to predict alpha and beta sheets from (primarily) linear protein libraries. Because cyclic peptides can form a wider range of conformations, it is good to optimize on a coil library - one without specific backbone structure. This optimization is performed with the goal of capturing the inherent preference of each amino acid. You should read more in depth about the RSFF2 force field in its paper: “Residue-Specific Force Field Based on Protein Coil Library. RSFF2: Modification of AMBER ff99SB'' by Chen-Yang Zhou, Fan Jiang, and Yun-Dong Wu. Because RSFF2 is derived from Amber99SB, we initially want to create residues in the Amber99SB force field that can be cyclized. Then, the Amber99SB topologies we create can be converted to RSFF2 topologies.

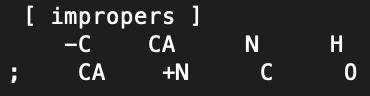
**To create cyclizable residues**,

1. Copy the GROMACS library for your current version into a local directory, naming it something like local\_gmx\_library. Then, use the command:

export GMXLIB=(your directory path)

The command is to make sure that GROMACS recognizes that you are using this library rather than the default.

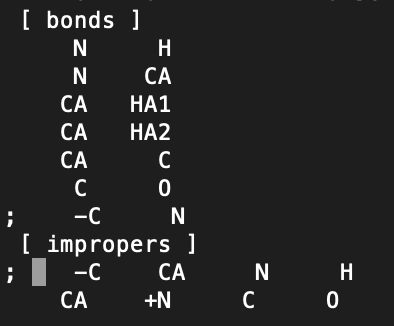
1. Make a copy of the Amber99SB force field within your GROMACS force field library and rename it to something descriptive. cd into your copied force field and edit the first line of forcefield.doc to distinguish this copied force field from the original.
2. **IMPORTANT:** Now open forcefield.itp and **change the fudgeQQ parameter to 1.0** instead of 0.833 (0.833 is the default value in the Amber99SB forcefield.itp file).
3. There are then **3 files** that must be altered to allow GROMACS to recognize your cyclized residues. GROMACS assumes that any given structure is linear. Assuming that we are, again, trying to simulate cyclo-(GNSRV), since there appears to be no preceding carbon atom for the N-terminal GLY and following nitrogen atom for the C-terminal VAL, GROMACS will use NGLY and CVAL for the first and last residues, respectively, and you’ll end up with a topology file for linear GNSRV. (The NGLY and CVAL residue names indicate N-terminal GLY and C-terminal VAL. They can be found in the aminoacid.rtp file in the forcefield directory). To work around this, we can make custom residues CNG and CCV for “cyclic N-terminal glycine” and “cyclic C-terminal valine” to trick GROMACS into connecting these two residues and giving you a topology file for cyclo-(GNSRV).
   1. **Make the C-terminal residue cyclizable:** To make the last residue in your sequence cyclizable, first copy the corresponding linear amino acid entry from **aminoacids.rtp** and paste it in another section in the same aminoacids.rtp file. So, for example, to make a cyclizable valine, we want to start with the information in aminoacids.rtp for VAL (**not CVAL**). We do this because the cyclized residue will eventually be the same as its linear counterpart – it should not be interpreted as a C terminal residue. Then, change the name of the residue to, for example, CCV so GROMACS can distinguish it. Finally, ***for your new residue, delete the line for the improper dihedral connecting your residue to the N of the next residue (symbolized in GROMACS as +N). In the below figure this line is simply commented out for the sake of clarity.*** This makes it so that GROMACS will not try to connect the 1st residue in your sequence with its preceding residue – this is the task that GROMACS doesn’t know how to do. Instead, this dihedral will have to be manually added in later.

****

* 1. **Make the N-terminal residue cyclizable:** We want to do a near-identical process for the 1st amino acid in your sequence. The main difference will be that instead of deleting the line that connects your residue’s C-terminal to the next residue, we want to get rid of the lines connecting your residue’s N-terminal to the next residue. To do so,

***i. we delete the improper dihedral with the -C atom, AND***

***ii. delete the bond between -C and N:*** (check below)



This is necessary because the “-C” symbol stands for the carboxyl carbon of the previous residue. For our N-terminal, there will be no -C. Instead, we will have to manually connect the “N-terminal” and “C-terminal” amino acids, as will be seen later.

* 1. Instead, to manually add a bond between the first and last residues in your sequence, you want to update **specbond.dat** so that GROMACS knows how to bond your newly created residues. For this process, type the following at the bottom of the file:Also, make sure to increment the number at the top of the **specbond.dat** to account for this new bond.

Now, when the command pdb2gmx is run, GROMACS will ask whether you want to connect GLY and VAL (assuming the distance between the atoms indicated above is < 0.14 nm), which is indeed what we want.

* 1. The last file you need to update is your **aminoacids.hdb** file. This will tell GROMACS how to add hydrogens to the first and last residues in your sequence, which are now called CNG and CCV. For this, you want to copy and paste the current .hdb entries for GLY and VAL, but rename them for the CNG and CCV residues that you created. **For the last residue in your sequence (CCV), this will be sufficient - GROMACS should add hydrogens to it just like it would in a linear peptide**. For the **first residue** in your sequence (CNG), however, you will need to:
     1. ***decrease the number next to the residue name (this number tells GROMACS how many lines to expect for that residue) by 1*** **and**
     2. ***delete the line which adds a hydrogen to its N-terminal (the line containing the -C atom).*** The hydrogen you left in the .pdb of your dehydrogenated peptide will take care of the hydrogen that you are removing from the .hdb file (recall the Chimera tutorial).

**Generating gro and topology files**

1. To make sure your GMXLIB path is correctly linked, echo $GMXLIB. You should see your modified force field in this directory.
2. At this point, you should be able to run the pdb2gmx command and create your desired cyclic peptide’s topology and .gro files in GROMACS (substitute your own file names for the examples below).

gmx\_mpi pdb2gmx -f GNSRV\_int\_noh.pdb

-o prot.gro

-p cGNSRV\_amber99sb.mod4CPs\_tip3p.top

-ter -inter

-chainsep id -merge interactive

Here, -ter and -inter will generate prompts for you to assign the desired charges to your various residues. “-chainsep id” along with “-merge interactive” will make GROMACS recognize that you want a single chain and, after answering relevant prompts that come up, that you want this chain to be cyclized.

* 1. The first prompt will ask you to select your force field. Make sure to select your modified force field. Your force field will be defined by the first line of forcefield.doc (step 1). We want to be able to cyclize our peptide correctly with our altered residues.
  2. The next prompt will ask for the desired representation of water. Choose TIP3P, as is recommended for Amber99SB.
  3. There will then be another prompt for you to indicate whether or not you want to bond GLY and VAL using the special bond that you wrote. Type “y” to link your defined starting and ending termini, and “n“ for any remaining such prompts.
  4. Finally, if your protein contains residues that can be charged, there will be a prompt asking for their protonation states. For residues such as aspartic acid, choose to create a -1 charge, and for residues such as arginine, choose to create a +1 charge.

This command will also create posre.itp, a file which defines position restraints for heavy atoms in your protein (non-hydrogens). This can be included in your .top file by using the -DPOSRES flag in the relevant .mdp file.

1. With our specbonds.dat file, we have added a bond between the first and last residues in our sequence. However, if we think back to our alteration of aminoacids.rtp, we also removed two improper dihedrals. We now need to add two improper dihedrals to manually connect the first and last residues. To do this, we want to open the topology file that was created (.top) and add the improper dihedrals under the [ dihedrals ] header, among the dihedrals with function 4. In the case of cyclic GNSRV, these are the dihedrals we are adding back.



1. Make sure to check your topology file. Check that your topology file indeed now has (1) a bond between the first and the last residues, as well as all the associated angles and dihedrals, and (2) double-check there are two improper dihedrals related to the peptide bond between the first and the last residues.

**Converting to the RSFF2 Force Field**

We want to use the RSFF2 force field, as it captures the behavior of cyclic peptides better than Amber99SB. However, RSFF2 is not yet included with GROMACS. As a result, to use RSFF2, you’ll need to use a script to transform an Amber99SB topology file to an RSFF2 topology file. This script will swap some force field file paths in your topology (.top) from your modified Amber99SB force field directory to reference some .itp files in a directory called RSFF2.

If you do not have an RSFF2 folder, you can add one to your own GROMACS library. You may download the RSFF2 force field separately, or modify the .itp files yourself. If you choose to do the latter, navigate to your GROMACS library and create a new directory called “RSFF2”. Then copy the forcefield.itp file from your modified Amber99SB directory into the RSFF2 folder. Edit the name of the forcefield.itp file to be RSFF2.itp. Then edit RSFF2.itp and verify the following changes:

* The fudgeQQ parameter is set to be 1.0 instead of 0.833 (0.833 is from the original amber99sb.ff)
* The #include statements are set to have a path to the modified Amber directory
  + - e.g. #include "ffnonbonded.itp" is modified to

#include "[Path to modified Amber99SB]/ffnonbonded.itp"

The most recent version of the g\_mod\_top\_RSFF2.py script is available in this repository. Be sure to read the README and the entire code to understand how it functions, what use cases it is built for, and what the expected behavior is in use cases that the script is not prepared to handle. The script was tested under Python 2.7.6. **Pay attention to the file paths that the script is referencing and edit these locations as necessary; the default modified Amber99SB directory for the script is amber99sbHY.ff, so if your local directory is named differently, you will have to edit the g\_mod script.**

The most recent version of this topology conversion script can handle uncapped linear, cyclic, and capped linear peptides (using Ace and NHMe caps). It is important to understand what is different for each of these cases when converting from the Amber99SB to RSFF2 forcefield, and which amino acids have RSFF2 parameterizations.

Example command structure:

python g\_mod\_top\_RSFF2\_cyclicComment.py cGNSRV\_amber99sb\_tip3p.top

cGNSRV\_rsff2\_tip3p.top

where cGNSRV\_amber99sb\_tip3p.top is the input Amber99SB topology file and cGNSRV\_rsff2\_tip3p.top is the output RSFF2 topology file

Note: This script g\_mod\_top\_RSFF2\_cyclicComment.py was originally developed by the authors of the RSFF2 paper mentioned above and then modified by our group members to treat cyclic peptides. There are some discrepancies between the modifications made by this script and those that were described in the original RSFF2 force field paper. These discrepancies can be seen in the file in the above directory named “0. Discrepancies between script and SI for RSFF2\_v4.docx”. Unfortunately, the RSFF2 authors are somewhat unclear on the reason for this difference, but claim that the script only has further optimizations compared to what was published in the paper.

At this point, you will have a topology file that describes your cyclic peptide with an RSFF2 forcefield. Additionally, you will have your prot.gro file and posre.itp file. These files together describe to GROMACS what your cyclic peptide is.

**Quiz (pt. 2)**

1. What are the 3 files that must be changed in your GROMACS library to create a pair of cyclizable residues?
2. Briefly outline what must be altered in the 3 files you listed.
3. Name 3 key differences between RSFF2 and Amber99SB
4. What files does pdb2gmx output?