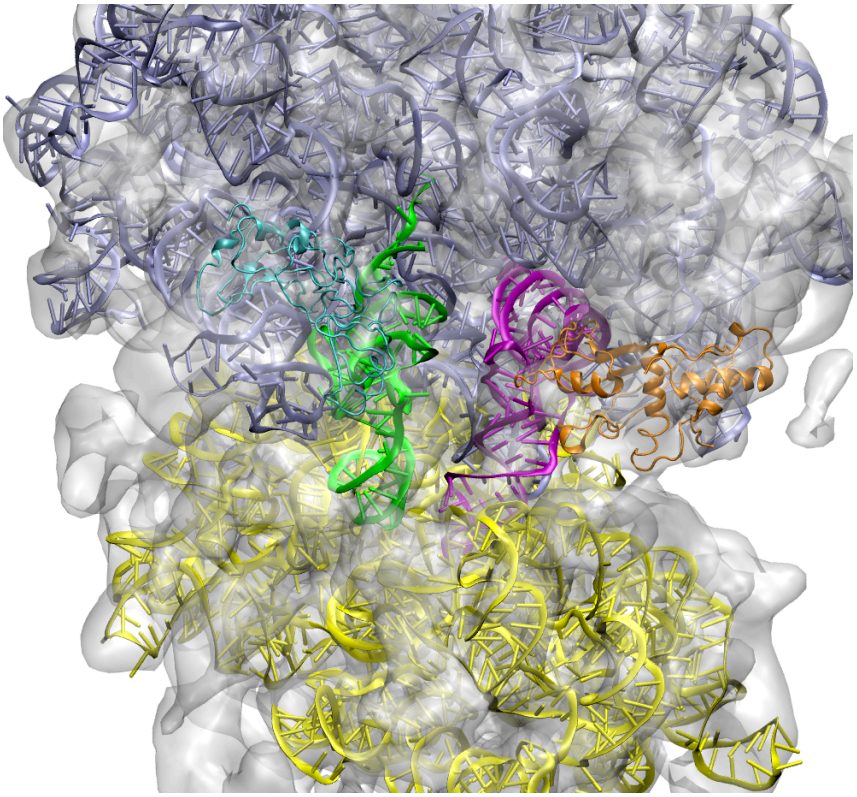
SLU

Swedish University of Agricultural Sciences

MMB 4.0



Tutorial

Updated June 17, 2023

Copyright and Permission Notice

Acknowledgments

Samuel Flores's early development of RNABuilder was funded by the Simbios National Center for Biomedical Computing through the National Institutes of Health Roadmap for Medical Research, Grant U54 GM072970. Information on the National Centers can be found at http://nihroadmap.nih.gov/bioinformatics. Past support has been received from eSSENCE and Uppsala University. The author is currently Dean of the Swedish National Graduate School in Medical Bioinformatics, at Stockholm University, funded by a Swedish Research Council grant.

Table of Contents

1 Overview

MMB constructs 3D structural and dynamical models of RNA and protein by applying user-specified base pairing interactions, interatomic forces, sterics, bond mobilities, and structural constraints. The forces, constraints, mobilities, parameters, and molecules can change from one simulation stage to another. It uses multi-resolution techniques, such as coarse-grained force fields and selective rigidification of groups of atoms, to decrease computation time.

MMB is run from the command line and requires a user-provided input parameter file that specifies the simulation, and optionally an input structural coordinate file in PDB format. It produces trajectory files, also in PDB format.

MMB was written in C++ code using Simbody and its molecular modeling extension, Molmodel. There are several ways to get MMB. If you have a simple nucleic acid folding problem, you can use our online service, webmmb.datmos.org. You can use docker to pull an image from dockerhub (you can pull a named distribution, or the very latest build). An MMB 3.4 executable is available for Windows (see separate mini-tutorial document which is included with that distribution). Older binaries are available for Intel-based Mac, and Linux. The source code is also freely available for download.

For a summary of what is new in this release, and for citation info, please see the Reference Guide.

2 Prerequisites and installation instructions

The examples in this tutorial generate results which should be interpreted with a molecular viewer such as Pymol, VMD, or Chimera. You will of course need the MMB executable and auxiliary files.

1. **VMD (or another software for viewing trajectory files):** We have experienced some problems installing VMD 1.8.7 on Windows. VMD 1.8.6 or Pymol can also be used.

   To install VMD, go to http://www.ks.uiuc.edu/Research/vmd. Click on "Download VMD" and select the installation for your platform. Follow the on-line instructions for installing.

2. **MMB:** MMB is a tool for building 3D RNA and protein models using a variety of knowledge the user has about the structure. It runs from the command line (don't expect a graphical interface!) and requires a user-generated input file and an MMB parameter file (more details below). The main output of MMB is the trajectory it generates from the provided parameters, in PDB format. The last frame of the trajectory is also saved as a PDB file.

   Previously we supported binary downloads, hosted at http://simtk.org/home/rnatoolbox. But you will see that we are now mostly relying on Docker images for newer releases. Before long on Debian flavors of Linux you will be able to install MMB with the package manager. The multiple ways of installing might make the instructions a bit confusing. In this chapter I explain some of the ways

to install, and during the rest of the tutorial I will just tell you to issue "MMB" and you will be expected to adjust that according to your installation method.

Windows:

Michal Maly has created a wonderful MMB 3.4 binary release, see the separate mini-tutorial in that distribution. You can download the much older `Installer.2_14.Windows.zip` from SimTK.org. Find it in Windows Explorer (a Windows Explorer window probably opened automatically when you downloaded the package). Right-click on `Installer.2_14.Windows.zip` and click on "`Extract to the specified folder`" in the mouse menu. In the "`Destination path`," type "`C:\Users\Installer.2_14.Windows`", or some other path of your choice. Don't extract it in "Program Files," because on some Windows flavors this directory does not allow writing output files.

I actually recommend using Docker for Windows instead. See the instructions for Linux below.

Mac OSX:

Here again I would recommend docker. However there are a few older Mac binaries on SimTK.org. If you want to use those, download the one you want, e.g. `Installer.2_18.OSX.tgz`. Move this file to another location – we suggest your home directory, in mac that would be `/Users/[your-user-name]` , in Linux that would be `/home/[your-user-name]`, or you can just use the Linux/Unix shortcut "`~`"; that's what we will do in this tutorial.

Now, open a Terminal window. You will find the Terminal application in:
Macintosh HD -> Applications -> Utilities -> Terminal
You will now decompress the above file. In Terminal, issue:

```
cd ~
tar -zxvf Installer.2_18.OSX.tgz
```

The files will be decompressed into the `~/Installer.2_14.[OSX | Ubuntu]` directory.

Now, you just need to tell OSX where to find the library files. That's easy, all the MMB files are in the same directory. You can specify this in your `~/.bash_profile` or wherever you put your configuration file, or just do it manually every time you run MMB. In bash the command is:

```
export     DYLD_LIBRARY_PATH=/Users/[your-user-
name]/Installer.2_18.OSX/lib
```

```
export     LD_LIBRARY_PATH=/home/[your-user-
name]/Installer.2_18.Ubuntu/lib
```

Note that you will have to adjust the above depending on where you installed MMB. Note also that you can't use the "~" shortcut in your `.bash_profile`.

Make sure you issue `source ~/.bash_profile` if you went that route. Now you're ready to go!

Some people have reported trouble with Docker on Mac, so I can't be sure that this would work for you. However if you can make it work that would be ideal. In that case install Docker and see the Linux instructions below.

Linux (64 Bit):

MMB is being packaged for Debian (including Ubuntu) so on those systems you will soon (perhaps by the time you read this) be able to install MMB with apt-get.

In the meantime, I would strongly recommend Docker if you are on Linux. Here you could issue the executable as:

```
docker run -v $(pwd):/work -it samuelflores/mmb-ubuntu3.2 MMB
```

If that seems awkward you can make a little bash script, e.g. in /usr/local/bin/MMB, containing:

```
#!/bin/bash
docker run -v   $(pwd):/work  -it samuelflores/mmb-ubuntu3.2 MMB
$@
```

You would thenceforth just issue "`MMB`" to execute MMB. The –v flag is telling docker to mount the current directory inside your docker image, so you will be able to read and write in your current directory just as with any normal command. Do not try to reference any input files above your current directory though. Just copy everything you need to your current directory.

There are also older binaries still available on simtk.org. The Mac instructions above will also work for you if you want to go that route.

3 Exercise 0: Your first MMB run

3.1 Objectives

This first exercise is intended for you to:

- Learn how to invoke MMB
- Verify that your installation is working properly
- Learn how to visualize the M-generated trajectory within VMD

3.2 Verify you have the required files

MMB requires two files in order to run:

- *parameters.csv*: This is a parameter file, analogous to those used by molecular dynamics programs to set bond, stretch, bend, torsion, etc. parameters. One of the main differences is that the parameters.csv specifies the rotation and translation relating the glycosidic nitrogen atoms in interacting pairs of bases. Casual users are unlikely to modify this file.

  On Windows, this file should have been copied from the latest examples folder into your MMB folder.

  If you are using the docker image, a parameters.csv will automatically be copied into your current directory.

- An input file: This text file specifies the sequence, base pairs, and any other constraints, forces, and options that should be applied. In this exercise, we will use *commands.singlebasepair.dat*.

Verify that you have these two files in your MMB folder. If you are using binaries, the default/recommended locations for your MMB folder are:

(Mac OSX) ~/Installer.2_18.OSX
(Linux) ~/Installer.2_18.Ubuntu

3.3 Open a command prompt/terminal window

MMB is run from the command prompt/terminal/console. If you haven't already done so, to launch a command prompt/terminal window, select:

(Windows) If you are using docker, see the docker literature and follow the Linux instructions once you are in your docker session. For 3.4, see the windows mini-tutorial. For older binaries you can open a command prompt. Start -> All Programs -> Accessories -> Command Prompt
(Mac OS) Macintosh HD -> Applications -> Utilities -> Terminal
(Linux) Open the Console that comes with your distribution.

3.4 Navigate to your MMB folder

For these exercises, we will be running MMB from the MMB folder. It is possible to run MMB from other directories, but the directory must contain the *parameters.csv* parameter file.

Within the command prompt/terminal window/console, navigate to the MMB folder. If you installed in the default locations, you would type:

(Mac OSX) `cd ~/Installer.2_18.OSX`
(Linux) `cd ~/Installer.2_18.Ubuntu`

Note: Quotation marks are required in specifying directory paths within the Windows command prompt window if the directory path includes spaces.

3.5 Run MMB

To run MMB, type:

(Mac OS) `./MMB -C commands.singlebasepair.dat`
(Linux) `./MMB -C commands.singlebasepair.dat`

If you are using Docker or installed with the package manager, adjust accordingly.

For OSX, you have a choice of executables depending on what version you're using. The `-C` option specifies the input file name, in this case *commands.singlebasepair.dat*.

Note: If you do not specify the `-C` option, MMB uses a default input file name of *commands.dat*.

You should see output that looks something like this:

```
[TwoTransformForces.cpp] Satisfied contacts   : 0 out of : 1
Writing structure for reporting interval # 1
[TwoTransformForces.cpp] Satisfied contacts   : 0 out of : 1
Writing structure for reporting interval # 2
[TwoTransformForces.cpp] Satisfied contacts   : 0 out of : 1
Writing structure for reporting interval # 3
...
```

If instead you see messages like the following:

```
/Users/Sam/svn/RNAToolbox/trunk/src/ParameterReader.cpp:2312 Unable
to open command file: commands.singlebasepair.dat
```

MMB could not find the input file you specified (in this case *commands.singlebasepair.dat*). Make sure you spelled the file name correctly and that it exists in the directory from which you are calling MMB.

3.6 Visualize MMB results

MMB generates a number of files that by default are saved to the directory from which you ran MMB. You should see a *last.1.pdb* file and a *trajectory.1.pdb* file. *last.1.pdb* is the PDB file for the last frame in the trajectory, which is typically the most interesting for structure prediction. The entire trajectory is saved in NMR format in the file *trajectory.1.pdb*.

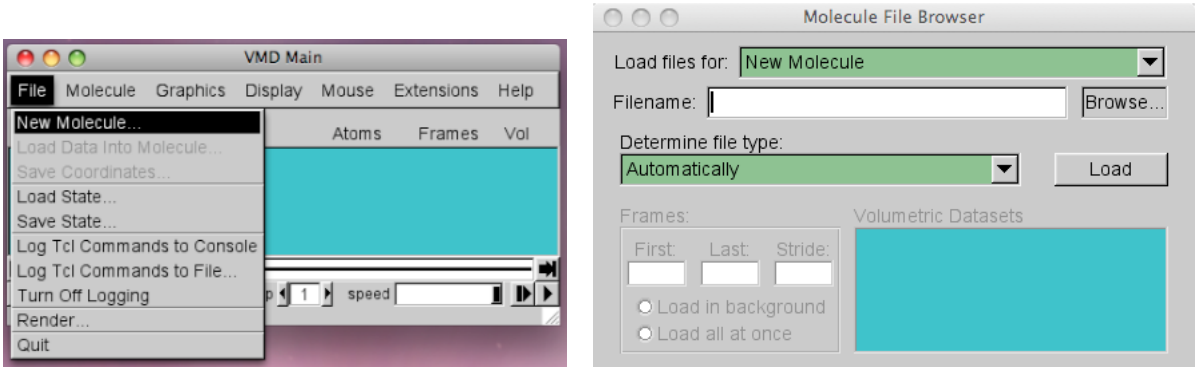
We can visualize the resulting trajectory within VMD:

1. Launch VMD. If you installed VMD in typical locations, you would select:

(Windows)	Start -> All Programs -> University of Illinois -> VMD -> VMD 1.8.7
(Mac OS)	Macintosh HD -> Applications -> VMD
(Linux)	The location may depend on your distribution.

2. The "VMD Main" window will appear. Select:

   File -> New Molecule...



3. In the "Molecule File Browser" that appears, click on "Browse" and select the *trajectory.1.pdb* created by MMB.

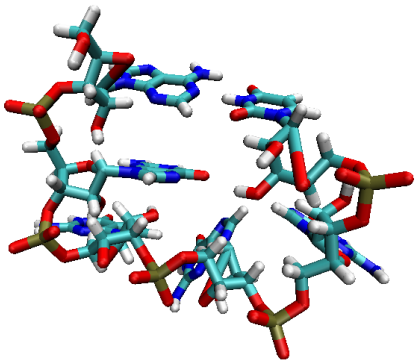
4. Change the molecule representation by going to the "VMD Main" window and selecting:

   Graphics -> Representations

   From the drop-down menu for "Drawing Method," select "Licorice." (In VMD 1.8.7, you might want to try the "New Cartoon" method, which provides a nice visualization of the molecule).

5. You should see a structure like that shown below by the end of the trajectory (the "Licorice" drawing method was used). In this simple example, a single base pair was specified pulling the two ends together.

   To rotate the structure, click and drag the structure. To translate the structure, type `t` and then click and drag the structure. To return to rotating the structure, type `r`.



4 Exercise 1: Generating your first 3D model

4.1 Objectives

In Exercise 1, you will:

- Learn about some of the key parameters that need to be specified within the command file
- Use your new knowledge about MMB to build a GNRA tetraloop from a starting sequence and published geometric constraints

4.2 Examining and editing the input parameters file

To edit or create and input parameters files, you must use a text editor (NOT a program like Microsoft Word, which will add many hidden characters for formatting, etc.) For Windows, we recommend using WordPad (Start -> All Programs -> Accessories -> WordPad). On Mac, some options include vi, emacs, and TextEdit (Macintosh HD -> Applications -> TextEdit.app).

Start your text editor and open up the file *commands.hairpin-short.dat*, located in your MMB "examples" folder. The default locations for your MMB folder are:

(Windows) My Computer -> C: -> Users -> Installer.2_14.Windows
(Mac OSX) ~/Installer.2_18.OSX
(Linux) ~/Installer.2_18.Ubuntu

If you are using Docker, you can start the container and fetch it from there:

```
docker run -v   $(pwd):/work -it samuelflores/mmb-ubuntu
cp /github/MMB/examples/commands.hairpin-short.dat .
```

```
exit
```

After you exit the container you will see the "`commands.hairpin-short.dat`" is in your current directory.

The syntax of the input parameters file is that each row contains information about one particular parameter. The first word in the row is the name of the parameter, followed by one or more values needed to specify that parameter.

4.2.1 RNA and protein sequence commands

The first section of the *commands.hairpin-short.dat* file contains the sequence parameters, described below. The `baseInteraction` records (discussed later) must appear sometime *after* the `firstResidueNumber` of each interacting chain in the base pair has been specified. `firstResidueNumber` must appear sometime *after* the corresponding `sequence` has been specified. Other than that, the order in which parameters are listed usually does not matter, except in some advanced usages not covered in this tutorial.

```
RNA A 2656 UACGUAAGUA
```

To instantiate a biopolymer, use `RNA`, `DNA` or `Protein` command. This takes the following parameters: chain ID (string, single character long), first residue number (integer), and sequence (string, single letter code).
This example instantiates an RNA chain with chain identifier "A", first residue number 2656, and the sequence shown in single-letter code. The chain identifier should be a single character in compliance with the PDB format. The sequence can be quite long, dependent mostly on your available memory. If you are supplying an input PDB structure file, the coordinates will be matched according to the chain ID and residue number.

4.2.2 Stage parameters

MMB can divide up the simulation into stages, each with its own set of simulation parameters. This allows flexibility in how the simulation is performed. Stages are explained

in more detail in Exercise 2. In this exercise, we will not be dividing up the simulation into stages, so the first stage and the last stage are the same:

```
firstStage 1
lastStage 1
```

This starts the simulation at stage 1, and ends when stage 1 is over.

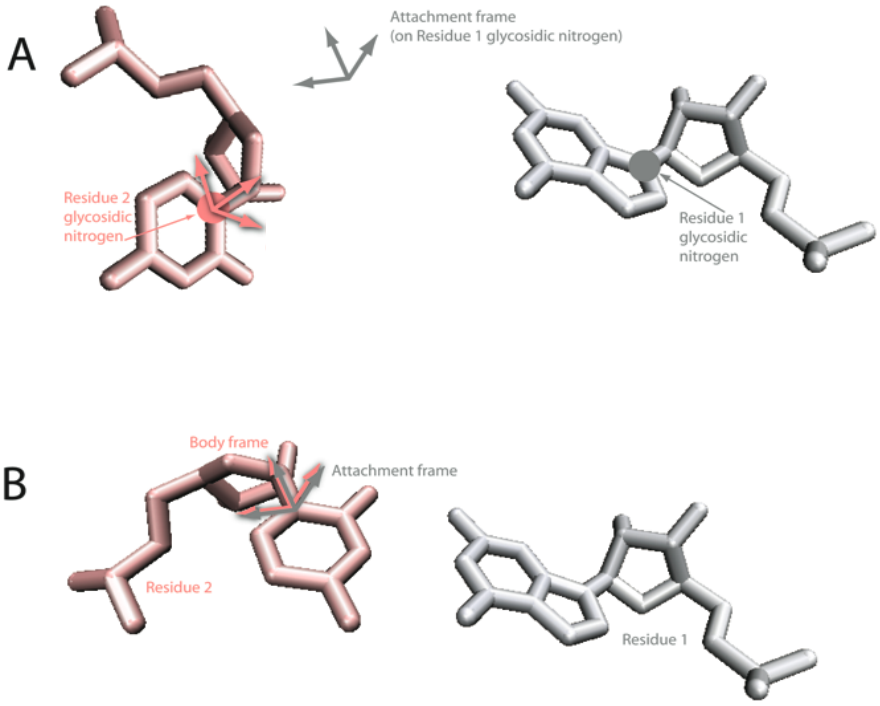
Run parameters

The next section of the *commands.hairpin-short.dat* file specifies the run parameters, which control the bookkeeping aspects of the MMB simulation.

4.2.2.1 baseInteractionScaleFactor

The `baseInteractionScaleFactor` (alias `forceMultiplier`, `twoTranformForceMultiplier`) is a scaling factor applied to the `baseInteraction` forces and energies. The base pairing forces themselves are applied using the following scheme.

First, an *attachment frame* is generated which is part of the first residue's glycosidic nitrogen body, but located outside it. Then a *body frame* is generated which is located at the center of the second residue's glycosidic nitrogen. The *body frame*'s x-axis points along the glycosidic bond, and its z-axis is perpendicular to its base plane. The location and orientation of the *attachment frame* is such that when it is aligned with the second residue's *body frame* the desired base pairing geometry is attained. Thus the task of parameterizing the MMB force field is firstly that of determining the correct position and orientation of the *attachment frame*. We distribute a program to compute this given the coordinates of a base pair with the desired geometry, but will not cover its use in this tutorial. After this is done one must also determine the depth of the potential well and its range – again beyond the scope of this tutorial.



In this exercise, `baseInteractionScaleFactor` was set to `20`, to make the forces strong enough for convergence:

```
baseInteractionScaleFactor 200
```

Note that it is not good idea to make the force multiplier *too* strong, because this will make the system stiff, which means there will be fast oscillations which will in turn require the variable time step integrator to take small time steps. If the `baseInteractionScaleFactor` parameter is not specified, it defaults to 1.

4.2.2.1 reportingInterval

This parameter controls the frequency of trajectory frames (reporting intervals) written by MMB.

`reportingInterval 4.0`	This instructs MMB to output a trajectory frame for every 4.0 ps of simulation time, starting at time 0

4.2.2.2 numReportingIntervals

This parameter controls the number of such frames written by MMB at a single stage.

Clearly, simulation time = numReportingIntervals * reportingInterval .

`numReportingIntervals 10`	This instructs MMB to write 10 frames at the applicable stage.

4.2.3 Temperature

In the *commands.hairpin-short.dat* file, the `temperature` parameter is specified:

`temperature 10.0`	This sets the temperature of the simulation to 10.0

If set`Temperature` is set to `TRUE`, as it is by default, MMB uses one of several available thermostat algorithms (set by `thermostatType`, which defaults to `NoseHoover`) to hold the system temperature to this setpoint. Note that thermostats do not conserve system energy.

4.2.4 Base pairing and nucleic acid duplex force commands

To generate base pairing forces to form the stem you can use the command:

```
nucleicAcidDuplex     <chain identifier A>
                      <first residue on A>
                      <last residue on A>
                      <chain identifier B>
                      <first residue on B>
                      <last residue on B>
```

Recalling that the duplex is antiparallel, we require that:

`(first residue on A) < (last residue on A)`

and

`(first residue on B) > (last residue on B)`

In the *commands.hairpin-short.dat* file, you will see an example of this:

```
nucleicAcidDuplex A 2656 2658 A 2665 2663
```

You can also specify the base pairing forces explicitly and individually. The syntax is:

```
baseInteraction <chain identifier for first residue>
           <residue number for first residue>
           <interacting edge for first residue>
           <chain identifier for second residue>
           <residue number for second residue>
           <interacting edge for second residue>
           <glycosidic bond orientation>
```

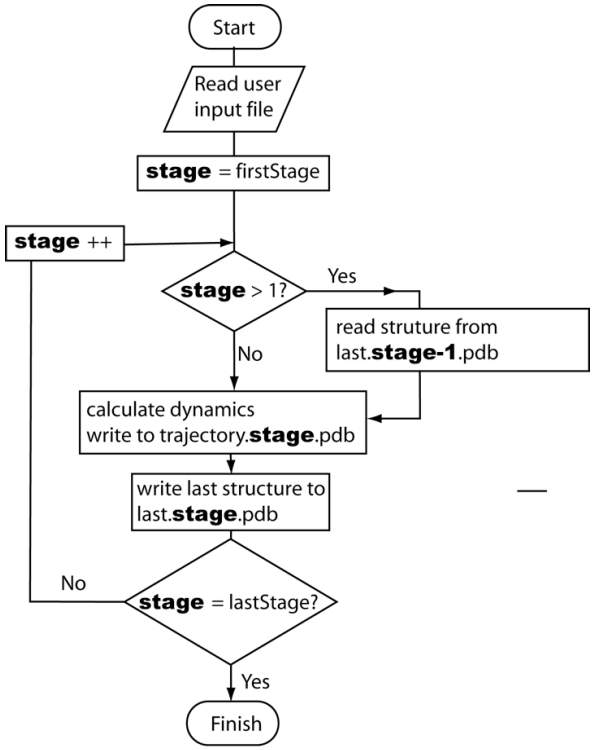
You can create the three base pairing forces above in this alternative way:,

```
baseInteraction A 2658 WatsonCrick  A 2663 WatsonCrick Cis
baseInteraction A 2657 WatsonCrick  A 2664 WatsonCrick Cis
baseInteraction A 2656 WatsonCrick  A 2665 WatsonCrick Cis
```

The first line specifies an interaction between the Watson-Crick edges of residues `2658` and `2663` of chain A, with the glycosidic bonds in the `Cis` orientation. See *Appendix: Forces* for an explanation of this type of interaction. See the same appendix for the other supported combinations of base pairing parameters.

When MMB sees three or more `WatsonCrick/WatsonCrick/Cis` interactions applied to three consecutive residues on each of two strands, it will automatically apply stacking interactions (`HelicalStackingA3/HelicalStackingA5/Cis`) to the consecutive residues (in this case `2656-2657, 2657-2658, 2663-2664,` and `2664-2665`). Thus the total number of `baseInteraction`'s in the system is 3+4 = 7. MMB monitors how many of these are approximately satisfied at each reporting interval, as you will see.

4.2.4.1 Using stages



MMB divides the simulation into stages, each with its own set of simulation parameters. The first stage is run using information solely from the input parameters file. Since there is no structure, all biopolymers are instantiated as extended chains. The last structure in this stage is written out to the file *last.1.pdb*. This *last.1.pdb* is the starting structure for stage 2 of the simulation. Similarly, at the end of stage 2, the file *last.2.pdb* is written out and used as the starting structure for stage 3. This process repeats for as many stages as specified.

Note that we can use this to start MMB using *any* PDB structure file. In the above explanation, `firstStage` was set to `1`, but there's nothing stopping us from setting it to a higher stage and reading in an arbitrary structure file, as follows:

1. Renaming the desired PDB file to *last.1.pdb*. Make sure the chain ID and residue numbering in the PDB file match that in the command file.
2. Setting the parameter `firstStage` to `2`
3. `lastStage` would also need to be greater than or equal to `2`

But let's not do that now! It will be part of a future exercise.

For now, we will use stages to change our simulation parameters, as we explain next.

4.2.4.2Turning any parameter into a staged parameter

Any parameters or commands can be enclosed in `readAtStage ... readBlockEnd` tags. This means that the enclosed parameters will be read only for the specified stage. So if you want to read certain values for parameter1, parameter2, etc only during stages 3, do this:

```
readAtStage 3
parameter1 value1
parameter2 value2
…
readBlockEnd
```

You can have as many of these blocks as you wish, and use them to change the parameters at many stages. There are some other block markers which behave differently (e.g. `readFromStage, readToStage, readUntilStage, readExceptAtStage`), see the *Reference guide*. Also, there are a few nuances to keep track of. The input file is read from top to bottom. Parameters encountered more than once in the input file are overwritten with the one closer to the bottom of the file prevailing. Commands (e.g. `baseInteraction`), on the other hand, are additive, rather overwriting each other. See also Appendix: Forces.

In this tutorial the first stage is very short – we are creating a hairpin without concern for steric clashes:

```
reportingInterval 4.0
numReportingIntervals 10
```

So we are specifying that at stage 1, we will run for 10 reporting intervals. Note that total simulation time = `numReportingIntervals*reportingInterval` .. so for stage 1 simulation time is 40 ps.

4.2.5 Global simulation parameters

The next section of the *commands.hairpin-short.dat* file specifies global simulation parameters, properties that apply to the overall simulation.

`numReportingIntervals 10`	This determines how many frames are generated. In this case, 10 intervals are requested, resulting in 11 frames (if we count last.1.pdb as the 11th) representing a 40-ps simulation.

4.2.6 Turning on the MD force field

You will see the following macro in your input file:

```
setDefaultMDParameters
```

This turns on all the PARM99 force field terms (except GBSA). It's equivalent to setting the following parameters:

```
globalAmberImproperTorsionScaleFactor     1
globalBondBendScaleFactor                 1
globalBondStretchScaleFactor              1
globalBondTorsionScaleFactor              1
globalCoulombScaleFactor                  1
globalVdwScaleFactor                      1
globalGbsaScaleFactor                     0
```

.

You can verify for yourself that these parameters and values appear in the stdout.

4.2.6.1 Run example

In your command prompt/terminal window (see Exercise 0), type:

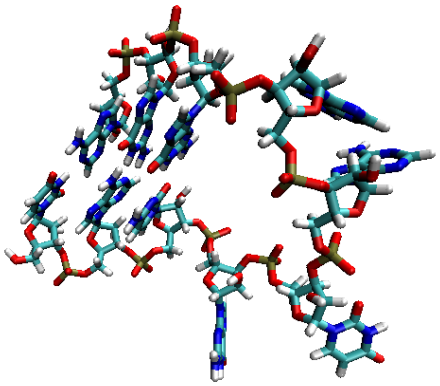
(Windows) `dir`
(Mac OS, Linux) `ls`

You will see a list of files in your current directory. Make sure you have *commands.hairpin-short.dat* and *parameters.csv*. If not, navigate to the directory with these files (see Exercise 0).

Now, run this example by typing:

(Mac OS) `./MMB -C commands.hairpin-short.dat`
(Linux) `./MMB -C commands.hairpin-short.dat`

The trajectory from this simulation run is in *trajectory.1.pdb*. The "1" in the output file name is the stage number. This trajectory can be loaded into and visualized with VMD (see Exercise 0. Make sure you first restart VMD or delete the molecule you loaded in that exercise). By the end of the trajectory, you should see a structure like that shown below. Notice how the 3 base pairs at the ends of the chain have been enforced to produce the hairpin structure.



5 Exercise 1B: Generating your first 3D model, modern way with NtC's

5.1 Objectives

This version of exercise 1 uses Nucleotide Conformers, which yield much better results, particularly with regard to helices. In this exercise, you will:

- Learn about some of the key parameters that need to be specified within the command file
- Learn about Nucleotide Conformers (NtC's)
- Use your new knowledge about MMB to build a GNRA tetraloop from a starting sequence and published geometric constraints

5.2 Examining and editing the input parameters file

To edit or create and input parameters files, you must use a text editor (NOT a program like Microsoft Word, which will add many hidden characters for formatting, etc.) For Windows, we recommend using WordPad (Start -> All Programs -> Accessories -> WordPad). On Mac, some options include vi, emacs, and TextEdit (Macintosh HD -> Applications -> TextEdit.app).

Start your text editor and open up the file `commands.GNRA-NtC.dat`, located in your MMB folder. The default locations for your MMB folder are:

(Windows) My Computer -> C: -> Users -> Installer.2_14.Windows
(Mac OSX) ~/Installer.2_18.OSX
(Linux) ~/Installer.2_18.Ubuntu

The syntax of the input parameters file is that each row contains information about one particular parameter. The first word in the row is the name of the parameter, followed by one or more values needed to specify that parameter.

5.2.1 RNA and protein sequence commands

The first section of the `commands.GNRA-NtC.dat` file contains the sequence parameters, described below. The `baseInteraction` records (discussed later) must appear sometime *after* the `firstResidueNumber` of each interacting chain in the base pair has been specified. `firstResidueNumber` must appear sometime *after* the corresponding `sequence` has been specified. Other than that, the order in which parameters are listed usually does not matter, except in some advanced usages not covered in this tutorial.

```
RNA A 1      UACGUAAGUA
```

To instantiate a biopolymer, use `RNA`, `DNA` or `Protein` command. This takes the following parameters: chain ID (string, single character long), first residue number (integer), and sequence (string, single letter code).
This example instantiates an RNA chain with chain identifier "A", first residue number 1, and the sequence shown in single-letter code. An experimental structure of this is PDB ID 5MRC, namely the stretch starting withresidue 2639. The chain identifier should be a single character in compliance with the PDB format. The sequence can be quite long, dependent mostly on your available memory. If you are supplying an input PDB structure file, the coordinates will be matched according to the chain ID and residue number.

5.2.2 Stage parameters

MMB can divide up the simulation into stages, each with its own set of simulation parameters. This allows flexibility in how the simulation is performed. Stages are explained in more detail in Exercise 2. In this exercise, we will not be dividing up the simulation into stages, so the first stage and the last stage are the same:

```
firstStage 1
```

```
lastStage 1
```

This starts the simulation at stage 1, and ends when stage 1 is over.

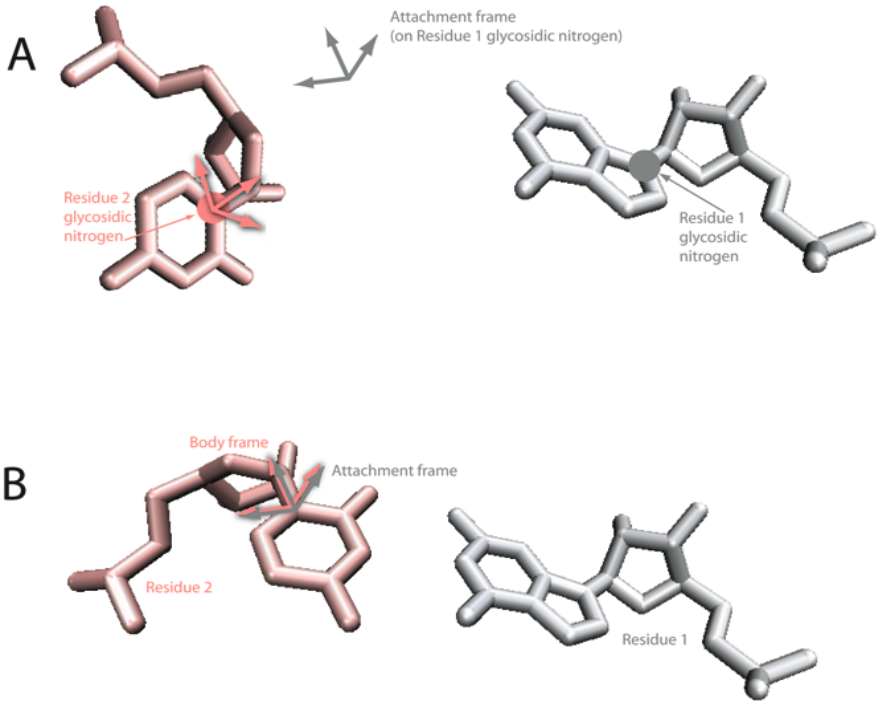
Run parameters

The next section of the *commands.GNRA-NtC.dat* file specifies the run parameters, which control the bookkeeping aspects of the MMB simulation.

5.2.2.1 forceMultiplier

The `forceMultiplier (`alias `twoTranformForceMultiplier, baseInteractionScaleFactor`) is a scaling factor applied to the `baseInteraction` forces and energies. The base pairing forces themselves are applied using the following scheme.

First, an *attachment frame* is generated which is part of the first residue's glycosidic nitrogen body, but located outside it. Then a *body frame* is generated which is located at the center of the second residue's glycosidic nitrogen. The *body frame*'s x-axis points along the glycosidic bond, and its z-axis is perpendicular to its base plane. The location and orientation of the *attachment frame* is such that when it is aligned with the second residue's *body frame* the desired base pairing geometry is attained. Thus the task of parameterizing the MMB force field is firstly that of determining the correct position and orientation of the *attachment frame*. We distribute a program to compute this given the coordinates of a base pair with the desired geometry, but will not cover its use in this tutorial. After this is done one must also determine the depth of the potential well and its range – again beyond the scope of this tutorial.



In this exercise, `baseInteractionScaleFactor` was set to `20`, to make the forces strong enough for convergence:

```
baseInteractionScaleFactor 200
```

Note that it is not good idea to make the force multiplier *too* strong, because this will make the system stiff, which means there will be fast oscillations which will in turn require the variable time step integrator to take small time steps. If the `baseInteractionScaleFactor` parameter is not specified, it defaults to 1.

5.2.2.2 reportingInterval

This parameter controls the frequency of trajectory frames (reporting intervals) written by MMB.

`reportingInterval 3.0`	This instructs MMB to output a trajectory frame for every 4.0 ps of simulation time, starting at time 0

5.2.2.3 numReportingIntervals

This parameter controls the number of such frames written by MMB at a single stage.

Clearly, simulation time = numReportingIntervals * reportingInterval .

`numReportingIntervals 6`	This instructs MMB to write 10 frames at the applicable stage.

5.2.3 Temperature

In the *commands.hairpin-short.dat* file, the `temperature` parameter is specified:

`temperature 10.0`	This sets the temperature of the simulation to 10.0

If set`Temperature` is set to `TRUE`, as it is by default, MMB uses one of several available thermostat algorithms (set by `thermostatType`, which defaults to `NoseHoover`) to hold the system temperature to this setpoint. Note that thermostats do not conserve system energy.

5.2.4 Base pairing and nucleic acid duplex force commands

To generate base pairing forces to form the stem you can use the command:

```
nucleicAcidDuplex     <chain identifier A>
                      <first residue on A>
                      <last residue on A>
                      <chain identifier B>
                      <first residue on B>
                      <last residue on B>
```

Recalling that the duplex is antiparallel, we require that:

`(first residue on A) < (last residue on A)`

and

`(first residue on B) > (last residue on B)`

In the *commands.GNRA-NtC.dat* file, you will see an example of this:

```
nucleicAcidDuplex A 1 3 A 10 8
```

You can also specify the base pairing forces explicitly and individually. The syntax is:

```
baseInteraction <chain identifier for first residue>
           <residue number for first residue>
           <interacting edge for first residue>
           <chain identifier for second residue>
           <residue number for second residue>
           <interacting edge for second residue>
           <glycosidic bond orientation>
```

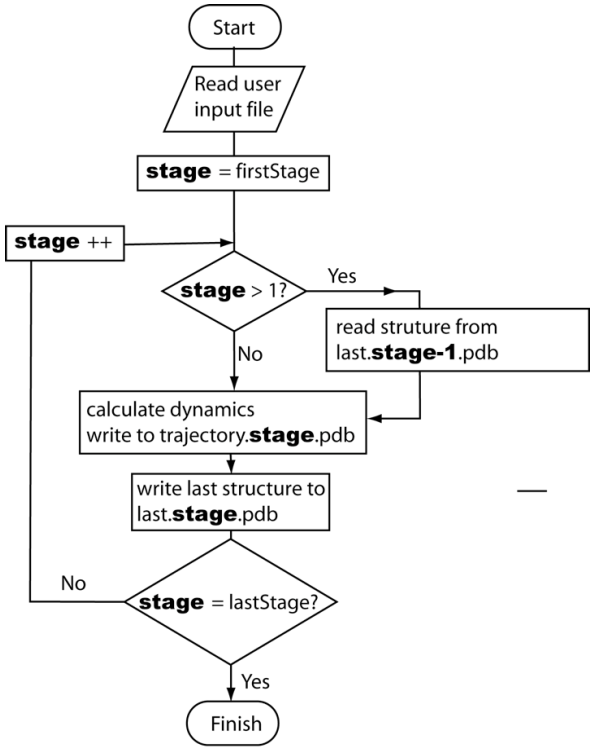
You can create the three base pairing forces above in this alternative way:,

```
baseInteraction A 1 WatsonCrick  A 10 WatsonCrick Cis
baseInteraction A 2 WatsonCrick  A 9  WatsonCrick Cis
baseInteraction A 3 WatsonCrick  A 8  WatsonCrick Cis
```

The first line specifies an interaction between the Watson-Crick edges of residues `2658` and `2663` of chain A, with the glycosidic bonds in the `Cis` orientation. See *Appendix: Forces* for an explanation of this type of interaction. See the same appendix for the other supported combinations of base pairing parameters.

When MMB sees three or more `WatsonCrick/WatsonCrick/Cis` interactions applied to three consecutive residues on each of two strands, it will automatically apply stacking interactions (`HelicalStackingA3/HelicalStackingA5/Cis`) to the consecutive residues (in this case `1-2, 2-3, 8-9,` and `9-10`). Thus the total number of `baseInteraction`'s in the system is 3+4 = 7. Except that in this exercise we are turning off the automated helical stacking. Anyway MMB monitors how many of these are approximately satisfied at each reporting interval, as you will see.

5.2.4.1 Using stages



MMB divides the simulation into stages, each with its own set of simulation parameters. The first stage is run using information solely from the input parameters file. Since there is no structure, all biopolymers are instantiated as extended chains. The last structure in this stage is written out to the file *last.1.pdb*. This *last.1.pdb* is the starting structure for stage 2 of the simulation. Similarly, at the end of stage 2, the file *last.2.pdb* is written out and used as the starting structure for stage 3. This process repeats for as many stages as specified.

Note that we can use this to start MMB using *any* PDB structure file. In the above explanation, `firstStage` was set to `1`, but there's nothing stopping us from setting it to a higher stage and reading in an arbitrary structure file, as follows:

4. Renaming the desired PDB file to *last.1.pdb*. Make sure the chain ID and residue numbering in the PDB file match that in the command file.
5. Setting the parameter `firstStage` to `2`
6. `lastStage` would also need to be greater than or equal to `2`

But let's not do that now! It will be part of a future exercise.

For now, we will use stages to change our simulation parameters, as we explain next.

5.2.4.2 Turning any parameter into a staged parameter

Any parameters or commands can be enclosed in `readAtStage ... readBlockEnd` tags. This means that the enclosed parameters will be read only for the specified stage. So if you want to read certain values for parameter1, parameter2, etc only during stages 3, do this:

```
readAtStage 3
parameter1 value1
parameter2 value2
…
readBlockEnd
```

You can have as many of these blocks as you wish, and use them to change the parameters at many stages. There are some other block markers which behave differently (e.g. `readFromStage, readToStage, readUntilStage, readExceptAtStage`), see the *Reference guide*. Also, there are a few nuances to keep track of. The input file is read from top to bottom. Parameters encountered more than once in the input file are overwritten with the one closer to the bottom of the file prevailing. Commands (e.g. `baseInteraction`), on the other hand, are additive, rather overwriting each other. See also Appendix: Forces.

In this tutorial the first stage is very short – we are creating a hairpin without concern for steric clashes:

```
reportingInterval 3.0
numReportingIntervals 6
```

So we are specifying that at stage 1, we will run for 10 reporting intervals. Note that total simulation time = `numReportingIntervals*reportingInterval` .. so for stage 1 simulation time is 18 ps.

5.2.5 Global simulation parameters

The next section of the *commands.hairpin-short.dat* file specifies global simulation parameters, properties that apply to the overall simulation.

`numReportingIntervals 6`	This determines how many frames are generated. In this case, 10 intervals are requested, resulting in 11 frames (if we count last.1.pdb as the 11th) representing a 40-ps simulation.

5.2.6 Turning on the MD force field

You will see the following macro in your input file:

```
setDefaultMDParameters
```

This turns on all the PARM99 force field terms (except GBSA) as explained before.

5.2.7 Turning OFF the old-style stacking parameters, and using NtCs

We used to use baseInteraction's to impose the correct stacking geometry in helices. These got applied automatically whenever three or more WatsonCrick base pairs were imposed in a row. Let's turn that off:

```
setHelicalStacking False
```

Now there is a more modern and effective way to do this, by restraining the backbone for consecutive pairs of nucleic acid residues. NtC class AA00 is the most populated class, corresponding to A-form helices. Let's impose those on the three base pairs in the helix, starting with the first stretch:

```
NtC A 1 2 AA00 .5
NtC A 2 3 AA00 .5
```

Actually for a continuous stretch you can just use a single command for the whole stretch, like this:

```
NtC A 1 3 AA00 .5
```

And also on the complementary stretch:

```
NtC A 8 10 AA00 .5
```

5.2.7.1 Making the GNRA tetraloop

trajectory.1.pdb should have a nicely folded stem. However you may note that the tetraloop "GUAA" meets the GNRA profile, so we can fold it into a GNRA tetraloop.

We do that at stage 2, enclosing the relevant commands in a block:

```
readAtStage 2
```

First we create a "sheared" or Hoogsteen/Sugar Edge/Trans interaction to "staple" the ends of the tetraloop:

```
baseInteraction A 2645 Hoogsteen A 2642 SugarEdge Trans
```

Next we need to stack a few residues. The A-form helical stacking parameters work OK for this:

```
NtC A 1 3 AA00 0.5
NtC A 8 10 AA00 0.5
```

And then we close the block:

```
readBlockEnd
```

5.2.7.2 Run example

In your command prompt/terminal window (see Exercise 0), type:

(Windows) `dir`
(Mac OS, Linux) `ls`

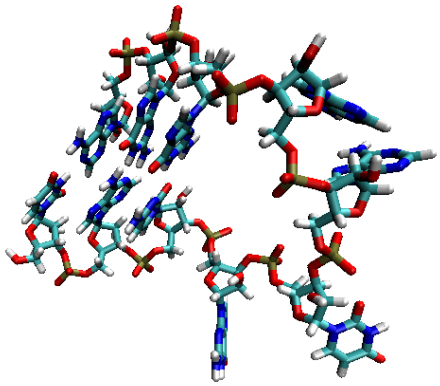
You will see a list of files in your current directory. Make sure you have *commands.hairpin-short.dat* and *parameters.csv*. If not, navigate to the directory with these files (see Exercise 0).

Now, run this example by issuing e.g.:

(Mac OS) `MMB -C commands.GNRA-NtC.dat`

(Linux) `MMB -C commands.GNRA-NtC.dat`

The trajectory from this simulation run is in *trajectory.1.pdb*. The "1" in the output file name refers to the fact that the results are from stage 1. This trajectory can be loaded into and visualized with VMD (see Exercise 0. Make sure you first restart VMD or delete the molecule you loaded in that exercise). By the end of the trajectory, you should see a structure like that shown below. Notice how the 3 base pairs at the ends of the chain have been enforced to produce the hairpin structure.



6 Exercise 2: Reading structures from a PDB file and rigidifying parts of your model

6.1 Objectives

In this exercise, you will:

- Learn how to use stages to read in and simulate a structure from a PDB file
- Learn about two new types of constraints that can be applied to your model: Weld and Rigid
- Experiment to see what happens when you release these constraints

In your MMB folder, you should see the following files: *1ARJ.short.pdb* and *commands.TAR.dat*. If you do not see them, make sure you are in your MMB folder (see Exercise 0). Also recall that you can fetch it from your docker image "examples" directory as explained in Exercise 0.

1ARJ.short.pdb is the file that we want MMB to read in, so let's copy it to a file named *last.1.pdb*. In your command prompt/terminal window, type:

(Windows) `copy 1ARJ.short.pdb last.1.pdb`
(Mac OS, Linux) `cp 1ARJ.short.pdb last.1.pdb`

Now, let's look at the input parameters file. Open *commands.TAR.dat* in your text editor. Notice that `firstStage` and `lastStage` are both set to `2`. Notice also that `sequence` and `firstResidueNumber` are set to match that of the TAR molecule. (You can compare the

values for these parameters with the PDB entry for 1ARJ at http://www.pdb.org/pdb/explore/remediatedSequence.do?structureId=1ARJ).

6.2 Rigid mobilizers and Weld constraints

MMB allows you to (1) fix a chain to ground, (2) weld two residues to each other, and (3) rigidify continuous stretches of residues.

(1) is useful for instances when you are not interested in the overall rotation and translation of a molecule, or when you expect that the 5' end would be fixed in an experimental situation. (2) is often useful when two strands of a helix have been made rigid and now need to be fixed with respect to each other, or to fix the ends of a flexible loop to each other. (3) can be used, for example, to rigidify regions of a molecule to focus resources on a small region of interest, or to model the motion of domains about a flexible hinge.

Note that while (3) almost always saves computer time, (1) and (2) may actually increase it. The reason for this is that rigidification involves Rigid *mobilizers,* but welding specifies Weld *constraints*. The latter create constraint equations which must then be satisfied, while the former simply prevent internal degrees of freedom from being created in the first place. See the *Simbody* literature (http://simtk.org/home/simtkcore and look under "Documents") for details on this. Also note that there are many more ways to control the bond mobilities in M, which we will not discuss in this tutorial.

The following parameter settings in the *commands.TAR.dat* show how to set up these different types of rigidification. Refer to the diagram on the next page for the residue numbering.

`removeRigidBodyMomentum False`	By default, MMB removes the rigid body momenta and keeps the system center of mass at the origin. While this is useful to prevent the system from spinning or drifting, it is not compatible with constraints to Ground, so we will turn it off for this simulation.
`constrainToGround N 17`	This command fixes the C3' atom of the specified residue (here chain N, residue 17) to the ground frame. See *Appendix: Forces*

for an alternative command.

```
mobilizer Rigid N 17 21
mobilizer Rigid N 41 45
```

These two lines rigidify helix I except for the base pair adjacent to the bulge (residues 17 to 21 and residues 41 to 45).

```
mobilizer Rigid N 27 38
```

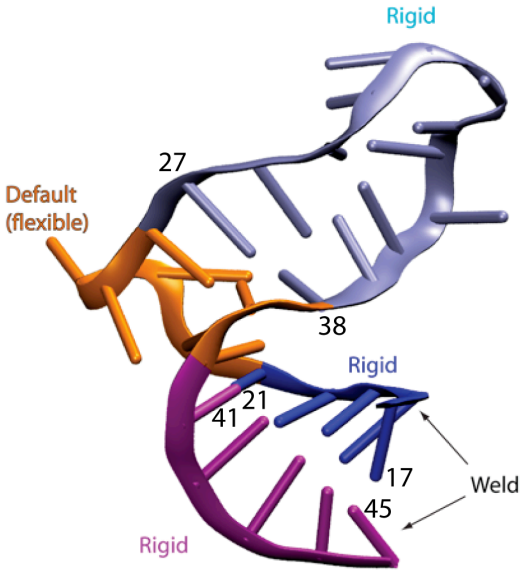
The stretch of residues from 27 to 38 (most of helix II plus the loop) are rigidified.

```
constraint N 17 Weld  N 45
```

This line welds the two ends (residues 17 and 45) together.

```
constraintTolerance .001
```

This line controls the fidelity with which Rigid and Weld constraints are enforced. A value of .001 means that all internal coordinates must be fixed within .001 nanometers or radians, depending on whether they are distances or angles.



6.3 SelectedAtoms

We use a syntactical variation of the `contact` command introduced in Exercise 1. Here we use the `SelectedAtoms` scheme, and also specify the residue range explicitly:

```
contact SelectedAtoms N FirstResidue LastResidue
```

Where `FirstResidue` and `LastResidue` are self explanatory – but we could just as easily have given residue numbers (including any insertion codes) explicitly – see the *Reference guide*.

These parameters you've encountered before:

```
numReportingIntervals  200
reportingInterval 2.0
firstStage 2
lastStage 2
temperature   10.0
```

6.4 Run example

Make sure you are still in the directory with the *commands.TAR.dat* and the *parameters.csv* files. To do this, in your command prompt/terminal window (see Exercise 0), type:

(Windows) `dir`
(Mac OS, Linux) `ls`

You will see a list of files in your current directory. Make sure you have *commands.TAR.dat* and *parameters.csv*. If not, navigate to the directory with these files (see Exercise 0).

Now, run this example by typing:

(Windows) `MMB.2_14.exe -C commands.TAR.dat`
(Mac OS) `./MMB -C commands.TAR.dat`
(Linux) `./MMB -C commands.TAR.dat`

The trajectory from this simulation run is in *trajectory.2.pdb* file. Note the "2" in the file name. Since the first stage in this run was "2," the corresponding output has a tag of "2" in its file name. Load this trajectory into VMD (see Exercise 0). During the trajectory, you should notice that one part of the structure is rigid and the other part is flexible.

6.5 On your own: Determine the effects of the Weld constraints and rigidification

Try holding the helices together with just base pairing forces rather than constraints. This is a matter of removing the lines specifying the Weld constraints and rigidification. Does the domain structure change much?

6.6 On your own: Turn the RNA into a different 3D structure

Change the sequence and/or constraints and turn the RNA into a different 3D structure, e.g., a hairpin or a pseudoknot.

7 Exercise 3: Homology modeling Azoarcus to Tetrahymena Ribozyme P6ab

7.1 Objectives

In this exercise, you will:

- Learn how to use MMB to construct a model using a known RNA structure as a template (this process is known as homology modeling)
- Practice using the `AllHeavyAtomSterics` collision detecting spheres (optional)
- Turn on MD forces using `setDefaultMDParameters`
- Learn how to use `alignmentForces`

7.2 Specifying the template

The template is the known RNA structure.

7.2.1 Read in the PDB file for the template

You will need to read in the PDB file for this RNA (see Exercise 2). In this exercise, you will be using the *1GID.shifted.pdb* file.

In your examples folder, you should see the following files: *1GID.shifted.pdb* and *commands.P6ab-threading.dat* (see Exercise 0).

Copy *1GID.shifted.pdb* to *last.1.pdb* by typing the following in your command prompt/terminal window:

(Windows) `copy 1GID.shifted.pdb last.1.pdb`
(Mac OS, Linux) `cp 1GID.shifted.pdb last.1.pdb`

Open up *commands.P6ab-threading.dat* in a text editor. Verify that `firstStage` is set to `2` so that the provided PDB file is read in and used by MMB.

7.2.2 Specify template sequence to match information in the PDB file

In the input parameters file, you will also need to specify a template sequence with a chain ID and residue numbering that matches that of the PDB file. If you open the file *1GID.shifted.pdb* in a text editor, you will see that the first residue is numbered "220" and has a chain ID of "Q." So, in the command file, you would include the following line:

```
RNA Q 220 GUCCUAAGUCAACAGAUCUUCUGUUGAUAUGGAU
```

7.2.3 Rigidify the template

Lastly, you need to rigidify your template molecule so that it does not move. The threaded chain is the one that will morph so that it matches the template. In this example, the following line would rigidify the template (Tetrahymena ribozyme P6ab):

```
mobilizer Rigid Q 220 253
```

7.3 Specify the sequence of the target chain

The target chain is the one being mapped onto a known structure. For the Azoarcus fragment, this is done with the following line:

```
RNA C 146 CCUAAGGCAAACGCUAUGG
```

P6AB

7.3.1 Account for sterics using "Physics where you want it"

We can use the PARM99 potential to prevent steric clashes and spread out the loop nicely. This turns on the Lennard-Jones and electrostatic terms, in addition to the bonded terms (which are on by default):

```
setDefaultMDParameters
```

Then we limit the MD forces to the *target* chain only:

```
includeResidues C FirstResidue LastResidue
```

Without this line, the template would also have non-bonded forces active, and would repel the threaded chain. In the original (Flores et al., RNA 2010) article, we used the `contact` command, you will see a note on this in the input file:

```
#contact AllHeavyAtomSterics C 146 164
```

This is faster, but can be a bit limited in preventing steric clashes, and won't have the long-range electrostatic repulsion that we find useful in this exercise.

7.4 Apply forces to pull the corresponding residues together

The `alignmentForces` keyword is explained in the Reference Guide, in our chapter on "Forces." Also see our chapter on homology modeling of proteins, in this Tutorial.

First, specify that all subsequent alignmentForces commands will be performed with a prohibitive gap penalty, effectively restricting us to ungapped alignments:

```
alignmentForces gapPenalty  -10000
```

Now we set the force constant of all the atomSpring's:

```
alignmentForces forceConstant 300.0
```

Lastly, let's issue the actual alignment commands:

```
alignmentForces C 146 151 Q 222 227
alignmentForces C 160 164 Q 247 251
```

In the first line above we are asking for residues 146-150 of the model ("C") to be aligned with residues 222 to 300 of the template ("Q"). For each pair of corresponding residues, this command looks for all (non-hydrogen) atoms in the first residue which have atoms with the same name in the corresponding second residue. It then applies a spring to pull those two atoms together. The spring has an adjustable force constant, which we earlier set to 300.

7.5 Run example

Make sure you are still in the directory with the *commands.P6ab-threading.dat* and the *parameters.csv* files. To do this, in your command prompt/terminal window (see Exercise 0), type:

(Windows) `dir`
(Mac OS, Linux) `ls`

You will see a list of files in your current directory. Make sure you have *commands.P6ab-threading.dat* and *parameters.csv*. If not, navigate to the directory with these files (see Exercise 0).

Now, run this example by typing:

(Linux) `MMB -C commands.P6ab-threading.dat`

P6AB

The trajectory from this simulation run is in *trajectory.2.pdb* file. Load this trajectory into VMD (see Exercise 0). At the beginning of the trajectory, you should see two distinct structures. Eventually, you should see one end of the Azoarcus fragment appear to attach itself to the Tetrahymena fragment and then gradually "thread" the rest of itself onto Tetrahymena. At the end of the trajectory, you will get a structure like that shown below, where the Tetrahymena template structure is in green and the Azoarcus target fragment is in blue. Notice that we did the homology modeling even though there are portions of Azoarcus that do not match to any parts of Tetrahymena; we dealt with this by only applying forces to corresponding bases, and leaving the rest alone.

8 Exercise 4: Protein homology modeling

8.1 Objectives

In this exercise, you will:

- Learn how to create protein chains
- Practice using the `AllHeavyAtomSterics` contact force
- Use the `threading` forces for protein chains

8.2 Specifying the template

The template is the known protein structure.

8.2.1 Provide the PDB file for the template

You will need to read in the PDB file for this RNA (see Exercise 2). In this exercise, you will be using the *protein-template.pdb* file.

In your Installation folder, you should see the following files: *protein-template.pdb* and *commands.protein-homology-modeling.dat*. If you do not see them, make sure you are in your installation folder (see Exercise 0).

Copy *protein-template.pdb* to *last.1.pdb* by typing the following in your command prompt/terminal window:

(Windows) `copy protein-template.pdb last.1.pdb`
(Mac OS, Linux) `cp protein-template.pdb last.1.pdb`

Open up *commands.protein-homology-modeling.dat* in a text editor. Verify that `firstStage` is set to `2` so that the provided PDB file is read in and used by MMB. There are a some reporting and simulation parameters which you're by now familiar with, and we'll skip the explanation of those.

8.2.2 Start the run

```
MMB -c commands.protein-homology-modeling.dat
```

Note that the last Windows release was 2.14. Windows users will therefore find that the tutorial does not exactly follow the contents of their command files. I actually hate Windows. There, I've said it! Anyway try the Docker image.

8.2.3 Specify template sequence to match information in the PDB file

In the command file, you will need to specify a template sequence with a chain ID and residue numbering that matches that of the PDB file. If you open the file *protein-template.pdb* in a text editor, you will see that the first residue is numbered "94" and has a chain ID of "E." So, in the command file, we have the following line:

```
protein E 94 CYDYDAIPWLQNVEPNLRPKLLLKHNLFLLDNIVKPIIAFYYKPIKTLNGHEIKFIRKEEYIS
```

8.2.4 Specify model sequence, for which no structural information is available

You will also need to specify a model sequence with a chain ID (here we use "H" as a mnemonic for "human") and residue numbering which should probably follow some biological convention. We got our sequence from the telomerase database (telomerase.asu.edu):

```
protein H 522 RSPGVGCVPAAEHRLREEILAKFLHWLMSVYVVELLRSFFYVTETTFQKNRLFFYRKSVE
```

8.2.5 Rigidify the template and constrain it to ground

You will need to rigidify the template, but not leave the model flexible. You might also want to constrain the template to ground, though that's a matter of taste. Anyway, you know how to do this:

```
mobilizer Rigid E 94 156
constrainToGround E 94
```

8.2.6 Globally align the model and template (with gaps)

Lastly, we will pull the model backbone into structural alignment with the template backbone based on sequence identity.

The syntax of the alignmentForces is in our chapter on "Forces" in the Reference Guide. In a nutshell, this is a utility that aligns sequences, and applies atomSpring forces between likenamed atoms in corresponding residues under that alignment. The alignmentForces keyword admits parameters or commands. Parameters apply to commands that are below that parameter in the input file, but not to any commands that are above it. So let's set the parameters for the alignment first. Start with the force constant for the atomSpring's:

```
 alignmentForces forceConstant 300
```

We want to allow gaps, so we leave the `alignmentForces gapPenalty` at the default (do nothing about this).

Next we tell it which chains need to be globally aligned:

```
alignmentForces H E
```

In the above we are using alignmentForces as a command, and passing two arguments – the chain IDs to be aligned. Global alignments have a high potential to be crappy locally, so make sure you check the alignment. This is explained in the Reference Guide but at the risk of being redundant -- search for "SeqAn sequence alignment follows:" in the (admittedly) verbose output. You may see something like this:

```
55 -RSPGVGCVPAAEHRLREEILAKFLHWLMSVYVVELLRSFFYVTETTFQ
               |  ||   | |    |     |     | |    |
  CYDYDAIPWLQNVEPNLRPKLLLKHNLFLLD-NIVKPIIAFYYKPIKTLN
50 . : KNRLFFYRKSV---
            | ||
       GHEIKFIRKEEYIS
```

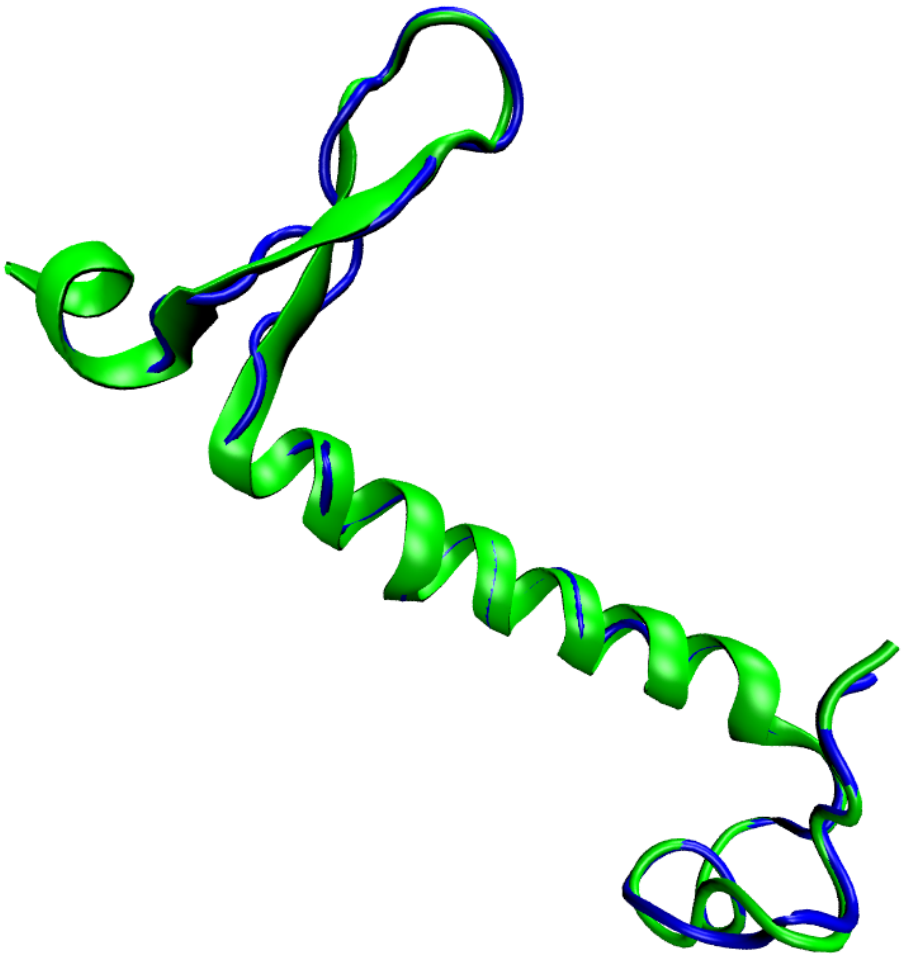
where "-" are deletions, and "|" are perfect matches.

8.2.7 Globally align (no gaps)

If you are not happy with this gapped alignment you can specify an ungapped alignment, by setting `alignmentForces gapPenalty` to a very negative value. Then we can also specify fragments (residue ranges) to be aligned:

#alignmentForces H 524 543 E 96 115
#alignmentForces H 544 580 E 117 153

In the first line above we are asking for residues 524-543 of the model ("H") to be aligned with residues 96 to 115 of the template ("E"). Similarly, chain H 544-580 vs. chain E 117-153. How do we know that these residue stretches should align? From the sequence alignment (again, telomerase.asu.edu). There is a single-residue insertion in chain E -- residue 116. Correspondingly, we do not align this residue with any on chain H. .. And we're done! The output should look something like this:



Where the template is in green and the model is in blue.

9 Exercise 5: Protein morphing

9.1 Objectives

Using what you learned earlier, you will learn to generate a putative trajectory between two known conformations of a macromolecule, a technique known as *morphing*. This will give you practice in:

- Using the `alignmentForce` command (for a slightly different purpose).
- Using the `loadSequencesFromPdb` command.
- Using the `mobilizer` command.
- Using the `readAtStage .. readBlockEnd` conditional blocks.
- Adjusting the `reportingInterval`

9.2 Introduction

We will morph Glutamine Binding Protein (GlnBP), a molecule somewhat larger than any we've worked with up to now. GlnBP is a domain hinge bending protein, meaning that it has stable structural domains connected by a flexible hinge. We will take advantage of this property by rigidifying the two domains of the model for time savings. This will get the model most of the way towards the target molecule. As an exercise, in a final stage you will leave the model flexible to complete the morph. You will see that this exercise is like the preceding homology modeling exercise, with one key difference: in morphing, not only the target's, but also the model's initial atomic coordinates are known.

Morphing is an old technique, and many good servers (e.g. molmovdb.org) and programs are available. You will see that selective rigidification offers the advantage of speed. In published work, we have morphed the entire ribosome, including all 50 protein subunits, in about 2.5 hours of computer time. Using MMB also gives you more control over precisely how the morph is done. The price of all this is that the process is bit more manual, but you will learn how to do it here.

9.3 Preparing the input structure file

In the previous exercise, we copied the coordinates into last.1.pdb. However that was a little kludgy. We can just specify the name of the file that has the structures we want. Recall that stage 1 does not take any input structures. So we start at stage 2:

```
readAtStage 2
# Template, 1WDN
loadSequencesFromPdb 1WDN.short.pdb
# Model, 1GGG
loadSequencesFromPdb 1GGG.short.pdb
readBlockEnd
```

MMB will find a chain "B" in 1WDN.short.pdb, and a chain "A" in 1GGG.short.pdb. There are actually ways to deal with chain naming conflicts, but for now just verify that we are not duplicating chain IDs.

9.4 Start the run

Start the job as usual:

```
MMB –c commands.protein-morphing.dat
```

9.5 Examine the input file

As in the previous exercise, we have two chains. However in contrast with the homology modeling example, here both chains are structured. The "model", chain A, has flexibility limited to the hinges and will be aligned with a fully rigid template chain B:

Chain B has some residues at the N- and C-termini which don't exist on chain A. However you can verify that residue A 5 corresponds to B 5, and so on all the way to residue 224. So we will pull those residues together like this:

```
alignmentForces gapPenalty -10000
alignmentForces  A 5 224 B 5 224
```

We will be doing this in two stages – one for rigid body alignment and another for semi-rigid morphing, as we will explain:

```
firstStage 2
lastStage 3
```

Initially we will use a `reportingInterval` of 10 ps, but you may reduce this later.

```
reportingInterval 10.0
numReportingIntervals 25
```

We will be constraining residues to ground later, so let's keep turn off the rigid body momentum removal:

```
removeRigidBodyMomentum false
```

The target molecule will be rigid throughout this exercise. The model will have some of the flexibility given back later. But we start by fully rigidifying both chains. The following syntax, with no chain IDs, tells MMB that it applies to all chains:

```
mobilizer Rigid
```

9.5.1.1 Turning any parameter into a staged parameter

Any parameters or commands can be enclosed in `readAtStage ... readBlockEnd` tags. This means that the enclosed parameters will be read only for the specified stage. So if you want to read certain values for parameter1, parameter2, etc only during stages 3, do this:

```
readAtStage 3
parameter1 value1
parameter2 value2
…
readBlockEnd
```

You can have as many of these blocks as you wish, and use them to change the parameters at many stages. There are some other block markers which behave differently (e.g. `readFromStage, readToStage, readUntilStage, readExceptAtStage`), see the *Reference guide*. Also, there are a few nuances to keep track of. The input file is read from top to bottom. Parameters encountered more than once in the input file are overwritten with the one closer to the bottom of the file prevailing. Commands (e.g. `baseInteraction`), on the other hand, are additive, rather overwriting each other. See also Appendix: Forces.

Our first task is to rigidly align the model and target, since they start out spatially quite separated. We will do this at stage 2, using the `readAtStage` command just introduced. At this stage, the model will be completely rigid:

```
# Rigid alignment stage
readAtStage 2
```

We turn off the electrostatics and Lennard-Jones forces for both chains. We have to turn them off in any case at least for the template, because otherwise we would not be able to superimpose the model on it:

```
deactivatePhysics A
deactivatePhysics B
```

We already specified alignmentForces, so not much else to do :

```
# This stage is short, will converge after 60 ps or so:
reportingInterval 10.0
numReportingIntervals 6
ReadBlockEnd
```

Then at stage 3 we will do the semiflexible morphing.

```
readAtStage 3
reportingInterval 1.0
numReportingIntervals 25
```

We know from published work that there is a hinge at residues 88-89 and 181-183. So we rigidify the thus-defined structural domains, and constrain one of them to the ground:

```
# Flexibilize just the hinges on your model:
mobilizer Default A 87+1 90-1
mobilizer Default A 180+1 184-1
```

Note that MMB can do +/- arithmetic.

Next we have to constrain the fragments that belong to the same structural domain. Otherwise the domain would fall apart:

```
# The N- and C-termini are in the same, discontinuous domain.
Constrain them to each other, otherwise the domain would fall apart:
constraint A FirstResidue Weld A LastResidue
```

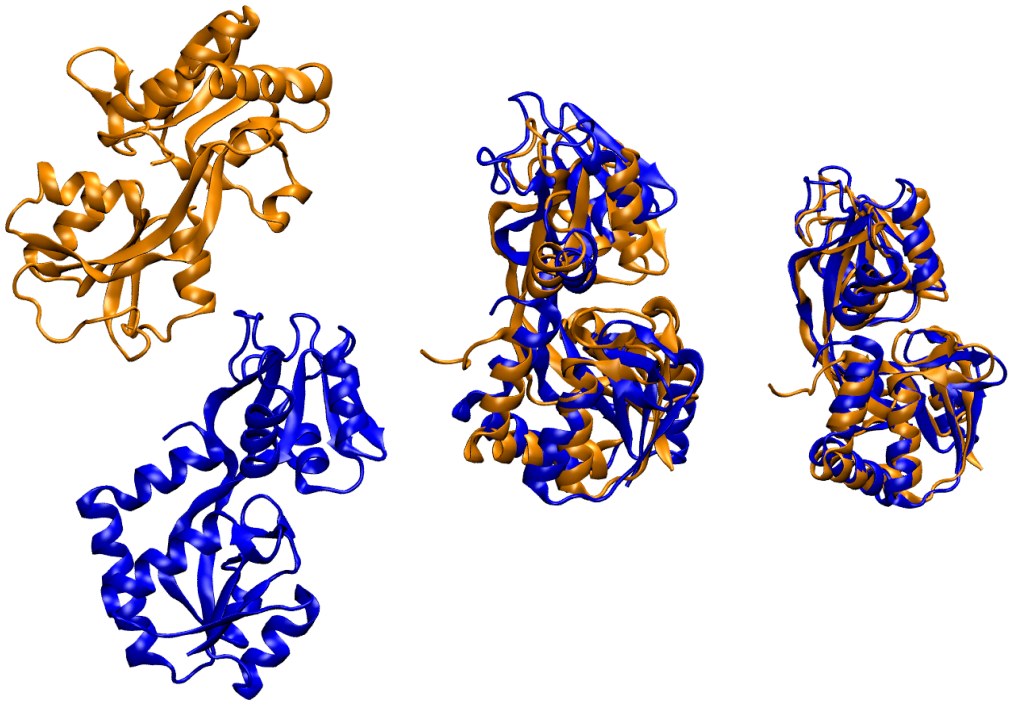
Next turn on the electrostatic and LJ interactions, for a zone 0.7 nm in radius around all flexible residues (in our case that means hinge residues). All residues outside this zone feel no such interactions. In the case of chain A residues, this is just a computational economy measure:

```
setDefaultMDParameters
physicsRadius .7
```

Remember we said the interactions have to be turned off all the time for the template chain. It is just a “ghost”, the alignmentForces are pulling A onto B, but otherwise A does not interact with B:

```
# Turn off physics for the template chain:
deactivatePhysics B
readBlockEnd
```

Stages 2 and 3 should be done by now. Open trajectory.2.pdb and trajectory.3.pdb in your molecular viewer. You should see something like the following:



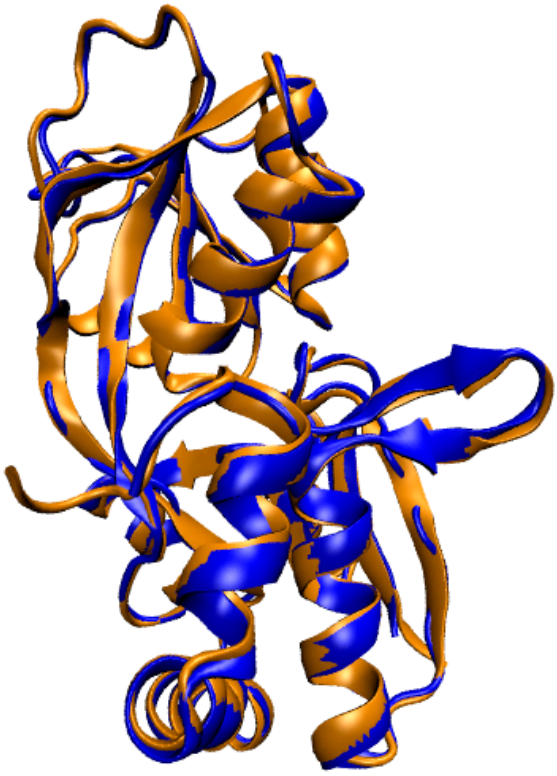
Left: Model (blue) and target (gold) in their initial, separated positions. Center: Model and target rigidly aligned. Right: Model semiflexibly aligned with target.

9.6 On your own: complete the morph with a fully-flexible alignment

You will notice that at the end of stage 3, the model is not fully aligned with the target. In this exercise, you will make the model fully flexible. You should end up with something like the image below.

Hints:

1. You will need to do this at stage 4.
2. Check that the model is fully flexible.
3. The simulation will be slower now, so reduce the reporting interval by ~ 10X.



10 Exercise 6: Efficiently generate alternate protein conformations

10.1 Objectives

In exercise 2 you learned how to use rigidification, `randomizeInitialVelocities`, and sterics to do thermal exploration of RNA conformations. In this exercise you will do the same for protein. Specifically this will teach you:

- The `ProteinBackboneSterics` sterics type arameter
- An efficient way to generate alternate conformations of proteins.
- Doing alignments and RMSD calculations in VMD

10.2 Introduction

There are many reasons to generate alternate conformations of proteins. Maybe you want to make an ensemble for protein-protein or protein – small molecule docking. Maybe you are trying to elucidate functional mechanisms. In any case, generating alternate conformations is often done by randomly moving atoms, or by using normal modes. These methods do not conserve the correct domain structure. For many hinge bending proteins, domain structure is conserved throughout the motion to some degree. In this exercise, you will find the alternate conformations that are possible under the assumption that only the hinge residues are flexible. You will use the `ProteinBackboneSterics` scheme, in which only the N, Cα, and C atoms get collision detecting spheres, to avoid generating clashing structures.

10.3 The command file

Open the file `commands.GlnBP-thermal-exploration.dat`. Most of the contents will be familiar to you. You haven't used this sterics parameter before:

```
contact ProteinBackboneSterics A 1 220
```

The hinge residues are 89-90 and 180-182:

```
mobilizer Rigid A 1 88
mobilizer Rigid A 91 179
mobilizer Rigid A 183 220
```

The first and third fragments comprise a discontinuous domain that we will fix to ground:

```
constrainToGround A 1
constrainToGround A 220
```

Leaving the second domain all the motion permitted by the hinge (and sterics).

10.4 Run MMB

Copy `1GGG.short.pdb` to `last.1.pdb`. The former is an open form of Glutamine Binding Protein (GlnBP).

Now run MMB against the command file, e.g.:

```
./MMB       -c commands.GlnBP-thermal-exploration.dat
./MMB  -c commands.GlnBP-thermal-exploration.dat
```

This will create a `trajectory.2.pdb`.

10.5 Analyze the conformational coverage

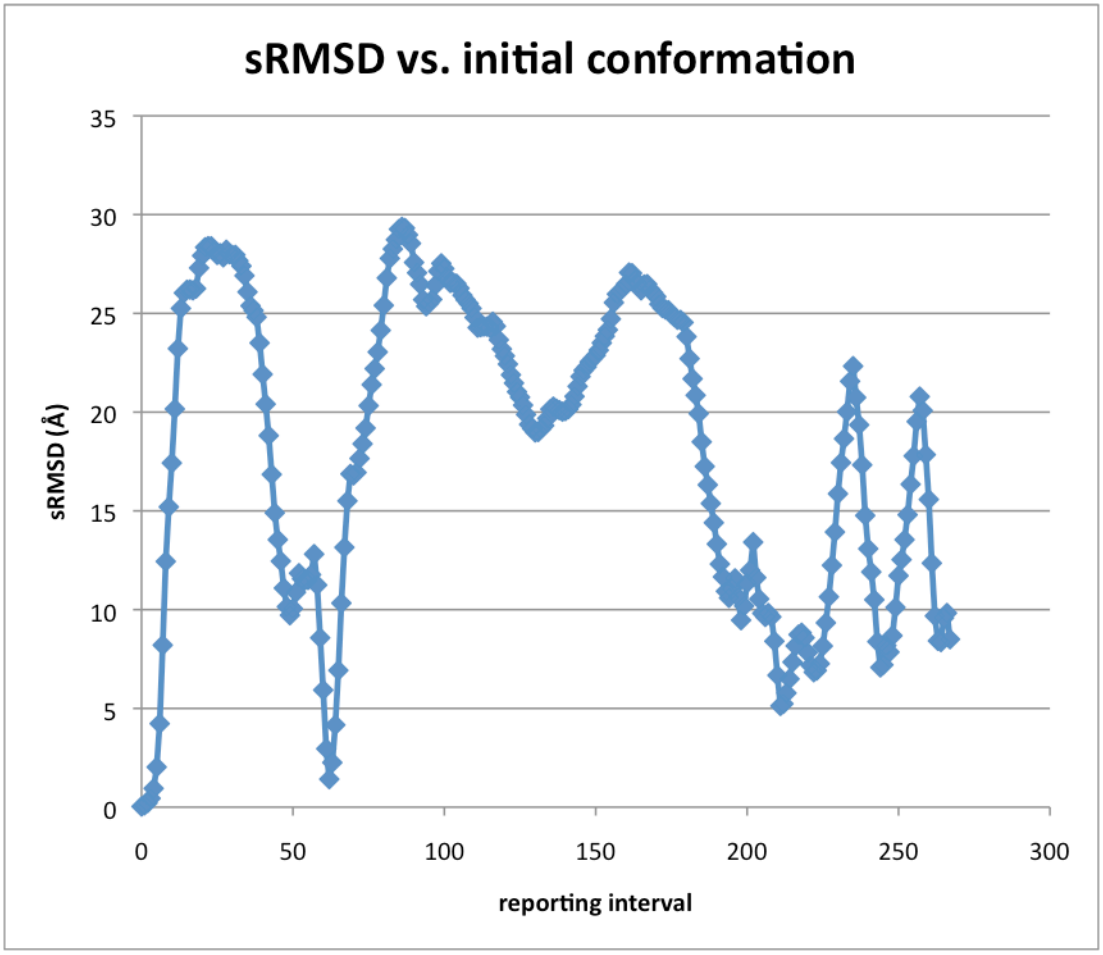
The idea behind a thermal exploration of this nature is that your ensemble may contain an alternate conformation which exists under certain conditions. You would not know this alternate conformation in a practical situation, so to have more confidence that your sampling is reasonably comprehensive, you might see how often the trajectory returns to its initial conformation, within perhaps 2Å RMSD or so. Actually we calculate this RMSD only over the mobile domain (in our case residues 87 to 175), a quantity Ruben Abagyan calls sRMSD. You can easily calculate this sRMSD using VMD. A sample script follows:

```
# loop a variable i from 1 to 269:
for {set i  1} {$i < 269} {incr i} {
# select residues 91 to 179 of the reference structure (frame 0, or
the starting conformation).  "atomselect 0" means choose the first
(in this case, the only) trajectory that is loaded in VMD. Put this
structure in a variable called sel0:
set sel0 [atomselect 0 "resid 91 to 179" frame 0];
# create a selection set (sel1) consisting of the same domain in frame
i :
set sel1 [atomselect 0  "resid 91 to 179"  frame $i];
# compute the RMSD between sel0 and sel1 :
set my_rmsd [measure rmsd $sel0 $sel1] ;
# print the RMSD :
puts $my_rmsd
# end the loop:
}
```

You can put this script in a file and read it in. It's pretty easy just to dump it as a single line into the TK console (Extensions -> TK Console), like this:

```
for {set i  1} {$i < 269} {incr i } { set sel0 [atomselect 0 " resid
91 to 179  " frame 0 ]; set sel1 [atomselect 0  "  resid 91 to 179  "
frame $i]; set my_rmsd [measure rmsd $sel0 $sel1] ; puts $my_rmsd   }
```

You will get a stream of sRMSD numbers. Cut and paste these into your favorite spreadsheet. You should be able to make a graph like this:



Note the sRMSD dips below 5Å a couple of times. You can run this longer if you are not convinced you've gotten good sampling. You can also compare this to the closed structure (PDB ID: 1WDN). That requires some aligning and careful definition of `sel0` and `sel1`.

11 Exercise 7: Solve protein structure by NMR constraints

11.1 Objectives

You've learned a lot so far. You've learned how to do everyday modeling tasks such as morphing, conformational sampling, and homology modeling. You learned how to turn base pairing contacts into 3D structure of RNA. I will progressively make things more challenging for you. In this chapter you will learn how to turn distance constraints, such as can be obtained from NMR experiments, into 3D structure with a little help from the Amber99 force field. This will involve the following tasks:

- Use the `atomTether` command
- Turn a list of distance constraints into MMB commands
- Turn on the Amber99 force field for the entire system

11.2 Instructions

You are reasonably far along now, so you don't need detailed instructions for everything – thus I'll skip a few basic steps. You will need to turn on all terms of the force field:

```
globalAmberImproperTorsionScaleFactor     1
globalBondBendScaleFactor                 1
globalBondStretchScaleFactor              1
globalBondTorsionScaleFactor              1
```

```
globalCoulombScaleFactor                  1
globalVdwScaleFactor                      1
```

In this exercise we are turning on physics everywhere. So you can just leave `physicsWhereYouWantIt a`t the default value, or set it explicitly:

```
physicsWhereYouWantIt FALSE
```

You might want to use a temperature that leads to some oscillation about equilibrium:

```
temperature 100
```

You will also need the sequence. You can extract it from `1UAO.short.pdb` using the `extract_FASTA.awk` script.

Lastly, you will want to add the distance constraints. Let's say you know that on chain A, atom 2HA of residue 1 is at most .45nm from atom HE3 of residue 9. The way you would enforce that is:

```
atomTether A    1  2HA  A    9  HE3 .4500  300.00
```

where the last (optional, defaults to 30 if left out) number specifies the spring constant of a spring that will pull the two atoms together if they're more than 4.5Å apart. I suggest making this 300 for this application, because empirically I've found this is strong enough.

Unfortunately the people that made the 1UAO structure didn't use the PDB atom naming convention. So we will have to correct the atom names. The following substitutions are necessary:

Everywhere:
HA2 2HA
HB2 2HB
HG2 2HG
HG21 1HG2

HG22 2HG2
HG23 3HG2

Residue 4 only:
HD2 2HD

If you want to skip this hassle, just use `1UAO.atoms-renamed.pdb`.

I've included a list of distance constraints called `1UAO-disre-simple.txt`. Try to use it to generate the `atomTether` commands. You may find the `parse-restraints.pl` script useful.

The `parse-restraints.pl` script looks like this:

```
# perl ./parse-restraints.pl 1UAO-disre-simple.txt 1UAO.short.pdb
#open the restraints file (first argument):
open RESTRAINTS, $ARGV[0]    or die $!;
#load restraints into an array:
@restraints = <RESTRAINTS> ;
close (RESTRAINTS);
int r;
#for each restraint:
for ($r = 0; $r < scalar(@restraints); $r++)
{
    #first atom number
    int $i ; $i =  substr($restraints[$r],0,3);
    #second atom number
    int $j ; $j =  substr($restraints[$r],10,3);
    #distance:
    int $dist ; $dist =  substr($restraints[$r],20,5);
    # initialize to "*" so we can later tell if not read
    $atomName1 = "*";
    $atomName2 = "*";
    int $residueNumber1; $residueNumber1 = -1;
```

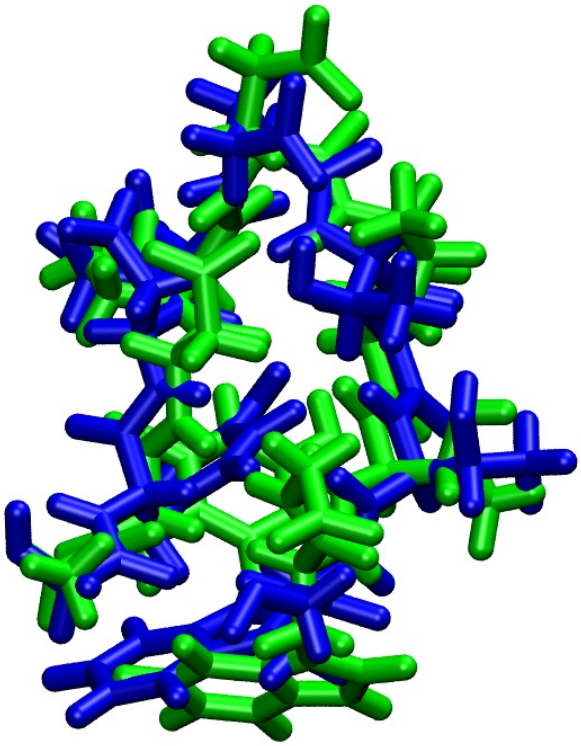
```
    int $residueNumber2; $residueNumber2 = -1;
    $chain1 = "A";
    $chain2 = "A";
    #open PDB file (second argument)
    open PDB, $ARGV[1]    or die $!;
    #for each line in PDB file:
    while (<PDB>) {
        #parse the atom number
        int $PDBatomNumber; $PDBatomNumber = substr($_,6,5);
        #if first atom number matches an atom number in the PDB file
        if ($PDBatomNumber == $i)        {
            # extract atom name, residue number, and chain ID:
            $atomName1      = substr($_,12,4);
            $residueNumber1 = substr($_,22,4);
            $chain1         = substr($_,21,1);
        }
        if ($PDBatomNumber == $j)        {
            # extract atom name, residue number, and chain ID:
            $atomName2      = substr($_,12,4);
            $residueNumber2 = substr($_,22,4);
            $chain2         = substr($_,21,1);
        }

    }
#print out MMB atomTether commands:
print  "atomTether  $chain1  $residueNumber1  $atomName1    $chain2
$residueNumber2 $atomName
2 $dist \n";
}
#done!
```

You will also need to convert the distances from nm to Å. While you're add it, set the spring constant to 300.0.

It's best to make your own MMB input file. But if you just want the right answer, look at the provided `commands.NMR.dat`.

11.3 Results

Your structure should agree with the published one (`1UAO.short.pdb` in your MMB distribution) within about 1.3Å RMSD.



12 Exercise 8: Fitting to electron density maps

12.1 Objectives

You may have found some of the preceding exercises redundant in some sense, perhaps repeating in protein what was already done in RNA, etc. Perhaps you could be forgiven for falling asleep. In this exercise we will do something completely different – fitting atomic coordinates to electronic density maps, which could have come from a cryo-electron microscopy (CryoEM), small-angle X-ray scattering, crystallographic, or other experiment. The skills you picked up in previous lessons about selective rigidification, constraints, forces, even "Physics where you want it" or straight-out all-atoms force fields will serve you well as you efficiently build 3D models. The following parameters will be new to you:

- `densityForceConstant`
- `densityFileName`

The following command will also be new:

- `fitToDensity`

If you want a challenge, you can also learn how to extract the sequence (in single-letter code) from and to renumber the residues in a PDB file. Hopefully you will also gain some insight into the flexibility of the ribosome.

12.2 Introduction

Electron density maps can be produced by cryo-electron microscopy (Cryo-EM), Small Angle X-ray Scattering (SAXS), X-ray crystallography, and other means. They are an important source of structural information. However they are hard to interpret without solving for the nuclear positions. Most of the structures in the PDB were once electronic densities, and have been fitted with nuclear positions.

There are many fine pieces of software available for fitting 3D structure to density maps. At Uppsala, "O" is quite popular. I will not attempt a full review of such programs here. Our approach, however, follows the work of Klaus Schulten, who invented Molecular Dynamics Flexible Fitting (MDFF). In MDFF, the atoms in the molecule or complex are subject to a conventional Molecular Dynamics force field, plus an additional force which is proportional to the atomic mass and the gradient of the electronic density. In MMB, we adapt this force as follows:

$$\vec{f}_i = A \cdot m_i \cdot \vec{\nabla} D(x_i, y_i, z_i)$$

Where i is the atom index, m_i is the mass of atom i, $D(x_i, y_i, z_i)$ is the electronic density at the nuclear position of atom i, A is a user-adjusted scaling factor, and $\vec{\nabla}$ is the gradient operator. Accordingly, $\vec{f}_i$ is the density-derived force vector applied to atom i. This is computed for and applied to every atom i in the system.

In this exercise, you will specify the sequence of a tRNA molecule, read in an initial set of nuclear coordinates, read in the density map of the ribosomal hybrid state, and then fit the tRNA molecule into the density. So let's get started!

12.3 Run MMB

In a practical situation, preparing a good starting model is an important part of the work. I used Venki Ramakrishnan's structure of the *T.thermophilus* ribosome in the classical state (2J00, 2J01, 2J02, 2J03), which I then semiflexibly morphed to match Jamie Cate's "R2" intermediate structure. You can read all about the why and wherefore in my 2011 paper in *Proceedings of the Pacific Symposium on Biocomputing*. Anyway, I took the morphed structure and re-centered it using COLORES, which is part of the Situs package. This is easier

than it might sound, but you won't have to do any of it, just use the coordinates in `tRNA.pdb`, which is in your MMB 2.4 distribution. Issue:

```
cp tRNA.pdb last.1.pdb.
```

Unfortunately this will actually take some time to converge. So start it now, so at least it will run for a couple of minutes while we finish going through the input file. Issue:

```
MMB      -c commands.tRNA-fitting.dat
```

Depending on your OS. Note that MMB 2.4.1 has a density fitting algorithm that is a full 10X faster than MMB 2.4! So make sure you are using at least MMB 2.4.1 for this exercise.

12.4 The command file

We first instantiate a tRNA molecule:

```
RNA V 5 CGCGGGAUGGAGCAGCCUGGUAGCUCGUCGGGCUCAUAACCCGAAGGUCGUCGGUCAAAUCCGGCCCCCGCAA
```

If you don't have the `commands.tRNA-fitting.dat` file, you can extract the sequence from a structure file that contains only the tRNA, using e.g. `awk -f extract-FASTA.awk <PDB file>.` You will find `extract-FASTA.awk` in your 2.4.1 distribution.

Next we rigidify the tRNA fully:

```
mobilizer Rigid
```

We have to turn off the rigid body momentum remover, since this would always be trying to recenter the molecules:

```
removeRigidBodyMomentum false
```

As you recall from our definition of $\vec{f}_i$ above, the scaling factor A is user-adjustable. In the command file, A is called `densityForceConstant`. Make this factor too small, and the

fitting will take forever. Make it too big, and the molecule might fly out into deep space. Turns out it's probably best to leave it at the default value of unity:

```
densityForceConstant  1.00
```

Now we specify the name of the electron density file, which has to be in XPLOR format:

```
densityFileName       ./tRNA.xplor
```

Then we activate the density-based force field for chain V:

```
fitToDensity    V
```

Note that we could just as easily have issued:

```
fitToDensity    V    FirstResidue    LastResidue
```

(which does exactly the same thing), or:

```
fitToDensity
```

(which fits all chains in the system, which in this case is also the same thing)

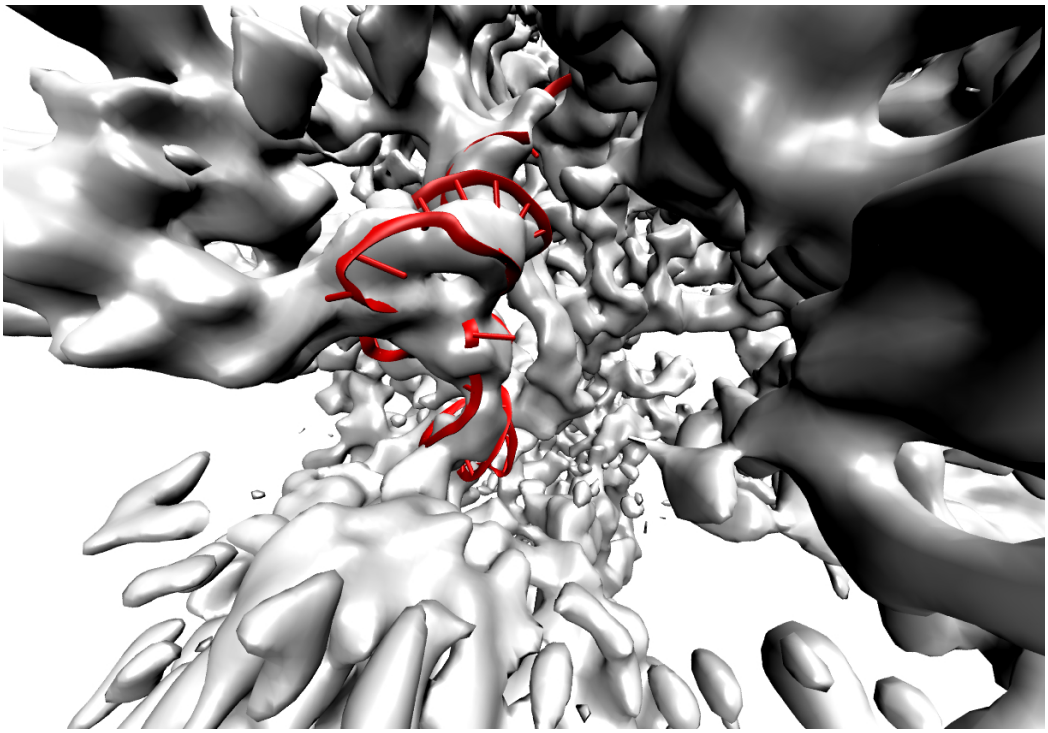
..I just wanted to make sure you understand the polymorphism of this command.

The rest of the parameters will be familiar to you.

```
temperature 1
numReportingIntervals 100
reportingInterval .01
firstStage 2
lastStage 2
```

12.5 View the results

VMD can display density maps. So read in `tRNA.xplor`. I rendered this using "Solid surface." Then read trajectory.2.pdb as a new molecule. You should be able to watch the tRNA move into its corresponding density. It should look something like this:



12.6 On your own

We just fitted the P/E site tRNA into the tRNA density map. If you want to fit a bigger subunit, try 16S. You can download the emd_1315 density map and fit the 16S from 2AVY (provided, or get from the PDB). You will need to extract the sequence of this subunit, and make everything but the neck region `Rigid`. You may consider the neck region to consist of residues 903 and 1373. Make sure you `Weld` together fragments of any discontinuous domain.

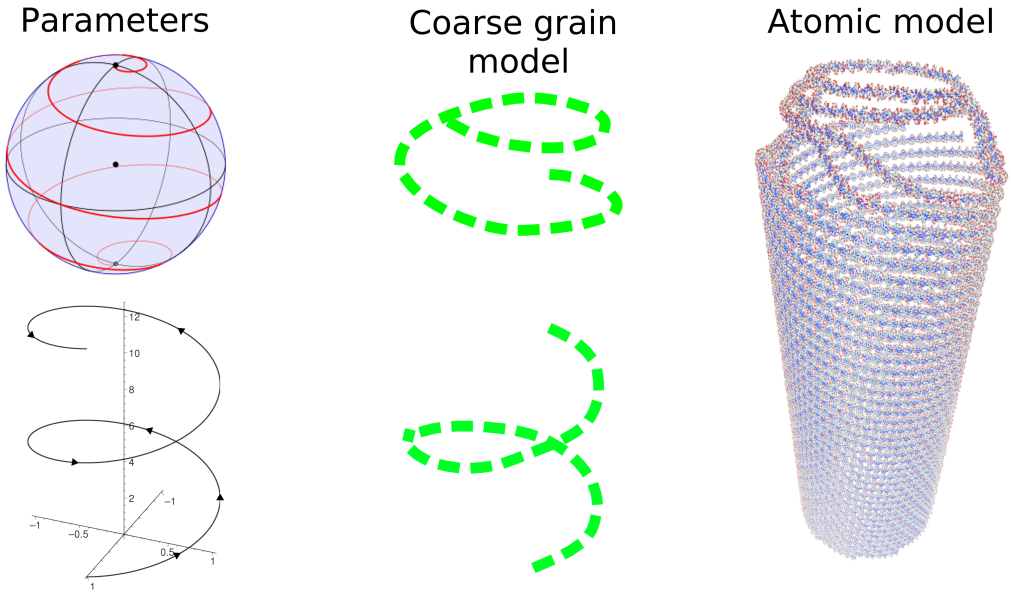
13 Exercise 9: Spiral genome tracing for viral DNA density maps

13.1 Objectives

For most viruses, determining the structure of the genome is a significant challenge. First, the resolution may be low. Second, even if the resolution is relatively good, the genome maybe thousands, tens of thousands, or even more base pairs long, so if you are trying to use traditional fitting software you may be facing months of full-time work not to mention excruciating eyestrain. However large regions of many viral genomes follow spherical- or cylindrical-spiral geometry. For these cases I have created the spiral genome tracing feature.

The figure below summarizes the process. There are two geometries that are currently supported: `sphere` and `cylinder`. Spirals have the property that consecutive windings are equidistant at closest approach – much like real DNA, which for physical reasons has a preferred interhelical distance. To make a coarse grained model of the DNA, we place one atom to represent each base pair, along the spiral trace. Of course we lose accuracy by using only one atom to represent the entire base pair. However we gain accuracy in that we can evaluate the fitting energy for the entire spiral at once. To understand why this is, consider the `pitch` parameter. A given trace could fit the observed density map perfectly at the beginning of the spiral, but if the `pitch` is off even by tenths of an Ångström, it will be completely wrong by the time it gets to the other side of the virus. Thus the ideal value of this parameter will be very clear from monitoring the fitting energy as we vary the `pitch`. You can set up a loop to sweep a parameter over a given range, and each consecutive evaluation will add only seconds to your compute time (reading in the density map takes a bit longer than that, but needs be done only once). When you are satisfied with the parameters, MMB can write out a command

file. Run that file, and the output will be an atomistic model, with each coarse-grained atom now replaced by one all-atom base pair.



The command that is new to you is `sphericalHelix`. It should always be followed by the subcommand "`writeCommands`" or any of the following parameters:

- `chainID`
- `center`
- `radius`
- `geometry`
- `cylinderHeight`
- `pitch`
- `helixAdvancePerBasePair`
- `spiralIsRightHanded`
- `startTheta`
- `endTheta`
- `phiOffset`
- `frequencyPhaseAmplitude`
- `spiralPdbFileName`
- `spiralCommandsFileName`

These are all explained in the "Spiral genome tracing" chapter of the MMB Reference Guide. Please stop reading this tutorial and read that chapter now.

13.2 A simple example, no computing of density fitting energy, no optimization:

The command file we will use is called `commands.P68-fitting.dat`, in your `examples` directory. Pop that open with your text editor and follow along.

First, just do this, don't worry much about the explanation (which is anyway in the command file):

```
readPreviousFrameFile 0
numReportingIntervals 1
```

Do you recall how to employ user variables? Let's define pi so we can use the more intuitive degrees and then convert to rads:

```
@pi 3.14159265358979
```

We will create a single chain, chain ID Z:

```
sphericalHelix chainID Z
```

Spiral will be right handed:

```
sphericalHelix spiralIsRightHanded 1
```

I had to do a bit of fiddling to get some parameters just right, hence the use of arithmetic:

```
sphericalHelix center 31.89 31.89+2.0 31.89
sphericalHelix radius   16.3+2.3+(-2)*.05
```

The pitch turns out to be pretty constant, across shells:

```
sphericalHelix pitch 2.3
```

As of course is this:

```
sphericalHelix helixAdvancePerBasePair .34
```

The north and south pole regions have very poor resolution in this map, so we only trace over a relatively modest range of theta:

```
sphericalHelix startTheta 1.00
sphericalHelix endTheta 2.2
```

This is a rotational offset, about the polar axis:

```
sphericalHelix phiOffset (16-2)*20*@pi/180
```

We clear the frequencyPhaseAmplitudeVector, out of paranoia rather than necessity:

```
sphericalHelix frequencyPhaseAmplitude clear
```

If you want to make an atomistic (fine-grained) model, separately run MMB using this command file:

```
sphericalHelix spiralCommandsFileName commands.spiral.dat
```

This is where the fine grained structural coordinates will be written:

```
sphericalHelix spiralPdbFileName spiral.pdb
```

This tells MMB you want to generate the command file, with name specified with spiralCommandsFileName above:

```
sphericalHelix writeCommands
```

Howe. It is the last `sphericalHelix` command you write, for a given chainID. However if you define a new `sphericalHelix chainID`, and change any other `sphericalHelix` parameters you wish, you can issue `sphericalHelix writeCommands` again and an additional chain will be created.

This is the P68 density map, provided by Dominik Hrevik and Pavel Plevka. Maybe you don't need to use it:

```
#densityFileName  LocalRef_02_Cl02_res85_nocaps2_box.mrc
```

If you do have that file.. this command tells MMB you want to compute the fitting energy for the coarse-grained chain Z. As a suggestion, vary one of the parameters above (center, radius, pitch, etc) and see which value gives you the lowest fitting energy. You can just search the stdout for "Total Potential Energy":

```
#fitToDensity Z
```

13.3 Challenge: optimize geometric parameters

To do this, you will have to have your density map file, specified in `densityFileName`. In this exercise, you will vary some parameter, and determine the value of that parameter which minimizes the fitting energy. I provide no help beyond these hints:

You should set `firstStage` and `lastStage` to range over a sufficient number of stages. Each stage you will evaluate one value of your parameter.
You may find the variable `@CURRENTSTAGE` useful. MMB sets this to the number of the current stage.
Then set your parameter using arithmetic and `@CURRENTSTAGE`.

13.4 Going atomistic

If you did the example, you will see a file called `commands.spiral.dat` , in your working directory. You do not need to do the optimization challenge to get this.

You will need to copy the base-pair.*.pdb files into your working directory. You will find those in MMB's `extras` directory. Run MMB using the mentioned command file. It will take a few minutes. When it is all finished, the `last.n.pdb` file with the highest "`n`" will contain the full spiral of DNA.

14 Virtual assembly of a protein-DNA complex

Contributed by Erik Marklund

14.1 Objectives

Quite often a good experimental structure model for the particular biomolecule that you are to study is not available under the specific conditions that you demand. For instance, a protein may have been crystallized with the "wrong" ligand, lacking one or a few domains, etc. In such cases the combined data from several experiments can still be used to create a reasonable structure model that can e.g. be used for subsequent molecular dynamics. In this exercise you will see an example of how an existing protein structure model can be used in conjunction with sequence data to produce a model of a related protein with maintained protein-ligand interactions. There are no new commands in this exercise, but the alignment will go beyond the ordinary use of `proteinThreading`.

14.2 Introduction

The tetracycline repressor (TetR) regulates the genes for tetracycline resistance in bacteria. It is a commonly used system for conditional gene expression and has a high affinity for its operator, *tetO*. There is a X-ray crystallographic structure of operator-bound TetR class D (TetRD) in the protein databank (id. 1QPI), but not for the related TetR class B (TetRB). Their high level of sequence similarity, however, allows for structural alignment of TetRB onto TetRD to yield a structure model of TetRB. Because of the specific interaction with DNA the side-chain conformations of the DNA-binding regions require special attention.

14.3 Run MMB

The input structure file (1QPI) requires little preparation. It contains one monomer from a homodimer and one strand from a double stranded DNA helix. The crystallographic symmetry found in the pdb file can be used at a later point to generate the homodimer bound to the double stranded DNA helix. Because of that we will only perform a structural alignment of a monomer here.

Use the structure model of TetRD as an input file:

```
cp TetR.pdb last.1.pdb.
```

Then execute MMB to do the actual alignment:

```
./MMB       -c commands.TetR_threading_TUT.dat
./MMB  -c commands.TetR_threading_TUT.dat
```

(depending on your OS). This will thread the TetRB polypeptide chain onto the TetRD structure while maintaining the side chain interactions with DNA.

14.4 The command file

First we set up the environment and instantiate the TetRD and TetRB monomers and the DNA:

```
firstStage 2
lastStage  2
reportingInterval 1.0
numReportingIntervals  50
temperature 1.0
removeRigidBodyMomentum false

# TetR class D, bound to DNA
protein A 4 LNRESVIDAALELLNETGIDGLTTRKLAQKLGIEQPTLYWHVKNKRALLDALAVEILARHHDYSLPAA...
# TetR class B, no structure
protein B 4 LDKSKVINSALELLNEVGIEGLTTRKLAQKLGVEQPTLYWHVKNKRALLDALAVEILARHKDYSLPAA...

# DNA
```

```
RNA M 1 CCUAUCAAUGAUAGA
RNA N 1 UCUAUCAUUGAUAGG
```

Perhaps you notice the high sequence similarity of the regions shown above. The structure of TetRD has some stretches of residues that were not resolved in the experiment. The TetRB sequence contains the corresponding residues and has additional insertions that will be part of the final structure model. This means, however, that care must be taken to align the right parts of TetRB to TetRD, as there is not a one-to-one mapping of all residues in the two sequences. The DNA molecules (here instantiated as RNA for technical reasons) are not necessary for the alignment, but make the final output more comprehensive.

Let's set up the homology modeling of the backbone:

```
threading A      4       155     B       4       155 300.0
threading A      156     198     B       169     211 300.0
```

Here the insertions create a discrepancy between the two sequences in terms of residue numbering, as discussed previously. The same thing affects the mobilizers that keep most of the proteins rigid throughout the homology modeling:

```
mobilizer Rigid A 4 198
mobilizer Rigid B 4 22
mobilizer Rigid B 30 34
mobilizer Rigid B 50 155
mobilizer Rigid B 169 211
mobilizer Rigid M 1 15
mobilizer Rigid N 1 15
```

The mobilizers above are further complicated by the fact that sidechains that make DNA interactions can not be kept rigid, or their final conformations will be off with respect to the DNA. Therefore we have split up one rigid part into several shorter ones.

We anchor TetRD and the DNA to the ground:

```
constrainToGround A 4
```

```
constrainToGround M 1
constrainToGround N 1
```

14.5 View the results

Fire up your molecular viewer of choice to inspect your new structure model. last.2.pdb only contains the monomeric protein and single stranded DNA. Hence you will need to make use of the crystallographic symmetry information that is contained in the input structure file.

14.5.1 Symmetry expansion with PyMOL

Copy the `CRYST1` record from last.1.pdb and the coordinates from last.2.pdb to a new file:

```
grep CRYST last.1.pdb > TetRB_threaded.pdb
cat last.2.pdb >> TetRB_threaded.pdb
```

In PyMOL you can now make a symmetry expansion. Open PyMOL, load the file `TetRB_threaded.pdb`, and execute `symexp`:

```
Load TetRB_threaded.pdb
symexp S_, TetRB_threaded, all, 1.5
```

This generates symmetry related copies of the monomeric protein and DNA locally. Note that this command is likely to create more copies than you need, so a few newly generated objects may need to be deleted from the selections/objects panel to the right. Once you have the homodimer you will be able to see if the inserted loops cause any clashes between the monomers that may need further processing. As you will see, the inserted loops are nicely situated in regions that are not occupied by any other atoms, so the entire structure is a plausible structure model of the TetRB-operator complex

The protein-DNA interface of a structurally aligned TetRB homodimer. The homodimer was constructed from the monomeric protein and DNA with the help of the `symexp` command in PyMOL. Not only is the structure model devoid of side-chain clashes; the specific interactions with DNA were reconstituted in the homology modeling process.

14.5.2 Symmetry expansion using other tools

Unfortunately, VMD currently lacks the capability to create the full homodimer directly from the crystallographic symmetry information contained in the pdb file. There are other tools at our disposal, however. Examples of such are XPAND (http://xray.bmc.uu.se/usf/) and CCP4 (http://www.ccp4.ac.uk/), both of which are free to use. Unfortunately, neither XPAND nor CCP4 are guaranteed to work out of the box, but if either of them is already present on your system you could try to make use of it. Finally, there is a web service – Quat (http://sysimm.ifrec.osaka-u.ac.jp/pdb_quat/) – that can do expansions according to both crystallographic and non-crystallographic symmetry. Before submitting your structure to Quat it is strongly recommended that you remove TetRD from the pdb file! Quat may destroy the chain labeling, so it's better to have as few chains as possible before submitting it. For this reason you may choose to also omit the DNA from the Quat input file since it is already symmetry expanded. Inspect the structure afterwards to make sure that the symmetry expansion produced sensible copies!

In principle the symmetry operations can be done in VMD with the help of rotations and translations, but requires some level of familiarity with the crystallographic space groups. In this case the other monomer(s) can be generated by rotating all atoms 180 degrees around the y-axis followed by translation by half a unit cell along the z-axis. This can also be accomplished by putting your favorite scripting language to good use.