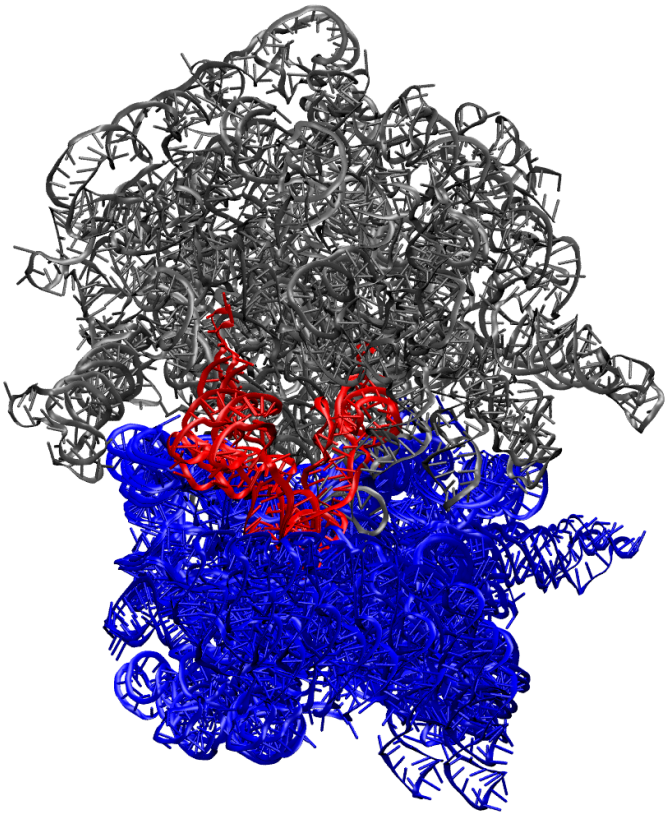
UPPSALA
UNIVERSITET

MMB 3.0



Reference guide

oktober 30, 2019

Copyright and Permission Notice

Table of Contents

1 What's new?

Release 2.18 introduces Nucleotide Conformers – NtC's, which were implemented in MMB by Emanuel Peter, Jiří Černý, and Bohdan Schneider. From 2.19 we now support triclinic density maps, that is to say unit cell axes need no longer be orthogonal. We also the add Planck's-Law noise generator.

For any published work which uses MMB, please cite one or more of the following:

Dourado, D. & Flores, S. C. (2014). A multiscale approach to predicting affinity changes in protein-protein interfaces. *Proteins*. doi:10.1002/prot.24634
Turning limited experimental information intio 3D models of RNA, by Samuel C Flores and Russ B Altman, *RNA* 16(9):1769-78 (2010).
Predicting RNA structure by multiple template homology modeling, by Samuel C. Flores, Yaqi Wan, Rick Russell, and Russ B. Altman (2010) *Proceedings of the Pacific Symposium on Biocomputing.*
Fast flexible modeling of RNA structure using internal coordinates, by Samuel C. Flores, Michael Sherman, Chris Bruns, Peter Eastman, Russ Altman (2011) *Transactions in Computational Biology and Bioinformatics* 8(5): 1247-57.

2 Biopolymers and monoAtoms

In this Appendix, we describe how to instantiate biopolymers (RNA, protein), as well as single atoms such as counterions. Note that the number of biopolymers and series of single atoms is limited by the number of characters available as chain identifiers.

2.1 Biopolymer sequences and first residue numbers

MMB can instantiate RNA chains using the following syntax:

```
RNA <chain ID> <first residue #>  <sequence in single letter
code>
```

The RNA sequence uses the single letter code (A,U,G,C). Similarly, you can instantiate DNA chains like this:

```
DNA <chain ID> <first residue #>  <sequence in single letter
code>
```

The DNA sequence uses the single letter code (A,T,G,C). You can instantiate a protein chain as:

```
protein <chain ID> <first residue #> <sequence in single
letter code>
```

The protein chains use the 20 canonical amino acid alphabet for specifying the sequence.

Note that as of release 2.12, you can make the chain ID as long as you wish. This means that you are not limited to the 144 printable ASCII characters. In the output PDB file, the following tag will indicate the long chain ID:

```
REMARK-SimTK-long-ChainID mySuperLongNameForThisChain
```

.. where of course "`mySuperLongNameForThisChain`" will be replaced by whatever chain ID you specified. This will be followed by the corresponding ATOM records. The ATOM records will have a blank (" ") in column 22 (the chain ID column). Finally, after all the ATOM records for that chain have been printed, there will be another tag, like this:

```
REMARK-SimTK-long-ChainID
```

.. followed by nothing. This "turns off" the long chain ID specification. The next chain may be a normal chain (single-character chain ID in column 22) or there may be another chain with a long ID bracketed by "`REMARK-SimTK-long-ChainID`" tags as before.

There is one more way to instantiate sequences, which works for protein, RNA, and DNA. You can issue the command:

```
loadSequencesFromPdb
```

And MMB will go to your input structure file (last.??.pdb) and look for RNA and protein chains. It will extract the chain ID's, residue numbers, insertion codes, and residue types from there. It will also match the internal coordinates to the Cartesian coordinates it finds there, as usual. You will then be able to issue commands that involve residues in those chains, as before. Please note that you CANNOT use this command with long chain IDs, it just won't work. If you have long chain IDs, just instantiate the chains explicitly.

In addition to removing the need for you to specify these chains manually, this command also has the advantage of handling insertion codes and gaps in the numbering. You will be able to append an insertion code to the right of the residue number in *any* command, e.g. `constrainToGround A 130B` (where B is an insertion code).

The residue numbers and insertion codes do need to be increasing from the top to the bottom of the input structure file, though. Before using this command, you should clean up the input structure file, removing anything that is not RNA or protein – including DNA, water, ions, or other molecules. `loadSequencesFromPdb` gives you one more advantage: it can read non-canonical nucleic acid types, following the PDB atom record format (https://cdn.rcsb.org/wwpdb/docs/documentation/file-format/PDB_format_1992.pdf). The disadvantage is that it just converts them to the nearest canonical equivalent, according to the following table:

3-letter code	Parsed residue type	1-letter code
ala	alanine	a
arg	arginine	r
asn	asparagine	n
asp	aspartic acid	d
cys	cysteine	c
cyx	disulphide bridged cysteine	x
glu	glutamic acid	e
gln	glutamine	q
gly	glycine	g
his	histidine	h
ile	isoleucine	i
leu	leucine	l
lys	lysine	k
met	methionine	m
phe	phenylalanine	f

monoAtoms

pro	proline	p
ser	serine	s
thr	threonine	t
trp	tryptophan	w
tyr	tyrosine	y
val	valine	v
sol	solvent	h
adp	adp	h
cli	cl	h
hoh	hoh	h
adn	adenosine	A
gua	guanosine	G
cyt	cytosine	C
ura	uridine	U
thy	thymine	T
h2u	uridine	U
omc	cytosine	C
omg	guanosine	G
psu	uridine	U
5mc	cytosine	C
7mg	guanosine	G
5mu	uridine	U
1ma	adenosine	A
2mg	guanosine	G
m2g	guanosine	G
da	deoxyadenosine	A
dg	deoxyguanosine	G
dc	deoxycytosine	C
dt	deoxythymidine	T

Whether you use `loadSequencesFromPdb` or specify the sequences manually, it is possible to use the +/- operators to increment or decrement a residue ID by some number of residues. For instance,

```
constrainToGround A 130A+2
constrainToGround A 130A-1
```

will constrain residues two residues to the C-terminus and one residue to the N-terminus of 32B. Do not insert any spaces between the +/- operators and either of their arguments. You can use the +/- operators with *any* command that takes residue numbers as an argument. You can also use user variables (which begin with "`@`," explained elsewhere in this document).

2.2 Clarifying arithmetic operations on residue numbers

Let's clarify the +/- operators for residue numbers a little more. In the context of residue numbers, the leftmost term in such an expression MUST be a real residue number that exists in the biopolymer. The remaining terms are strictly integers which will be summed to determine the increment/decrement along the sequence to be applied to the residue number. For example, assume the first residue number is 129. Then this is OK:

```
@myRes 130
mobilizer Rigid A @myRes+1  @myRes+1+1
mobilizer Rigid A @myRes+1  @myRes+2-1
```

The expressions evaluate as:

`@myRes+1` : If the residue that follows `130` in chain A is 130A, it will return 130A. This is not an integer sum. Note that residue `130` MUST exist, or an error will be tripped.
`@myRes+1+1` : This will first sum all terms except the leftmost, strictly arithmetically, and get 1+1 = 2. If the residue that follows `130A` is `130B`, then it will take the residue number 130, go 2 places towards the end terminus, and return `130B`.
`@myRes+2-1` : This will return `130A` in this example.

This is NOT OK :

```
mobilizer Rigid A 1+@myRes  1+1+@myRes
```

`1+@myRes` : MMB will look for a residue `1` .. and not find it (in this example the biopolymer begins at residue 129. It will then throw an error. MMB will NOT conclude that you are looking for `131`, even if `131` does exist on chain A.

As mentioned earlier, insertion codes are OK:

`130A+1` : is OK, and will return whatever the next residue actually is in the biopolymer, in this example 130B.
`130A+@myRes` : is OK, if there are indeed at least 130 residues following 130A. if the portion of the biopolymer after 130A is shorter than this, then it will throw an error.

`1+125A` : is NOT OK, even if residue 1 exists. Because `125A` is not an integer. Anything terms other than the first must be integers.

Note that for commands that do NOT expect a residue number, arithmetic operators, numbers, and user variables can be in any order.

2.3 monoAtoms

monoAtoms

The `monoAtoms` command specifies single atoms (e.g. monatomic ions) The syntax follows:

```
monoAtoms <chain ID> <first residue #> <# of atoms> <name of atom>
```

Currently only the following atom names are supported:

```
Mg+2, Cl-, Na+, K+, Li+, Ca+2, Cs+, Rb+
```

The single atoms created with this command support the `atomSpring, atomTether, springToGround,` and `constrainToGround` commands, just like the biopolymers. They do not support the `mobilizer` command. The `constraint` command works with monoAtoms to some degree. monoAtoms are automatically added to the physics zone.
You can read the positions of monoAtoms from your input PDB file. You have to be careful with atom and residue names in that file though. For example, for a magnesium ion, the atom name should be "`Mg+2`", while the residue name should be "`MG`". Note that this is different from the PDB convention, in which both atom and residue name are "`MG`".

2.4 Water droplets

The `waterDroplet` command puts a water droplet of a specified `<droplet chainID>` and `<radius>` about the point `<X> <Y> <Z>.` There are tethers (with optional parameter `[tether strength]`) which constrain the water to a distance of 1.1 * `<radius>` about `<X> <Y> <Z>`. The syntax is:

```
waterDroplet <droplet chainID>  <X> <Y> <Z> <radius>
     [tether strength]
```

Alternatively, you can provide a biopolymer (protein, RNA, or DNA) chain ID and residue number, and the water droplet will be centered about that residue:

```
waterDropletAboutResidue <biopolymer chain ID>
     <biopolymer Residue Number> <radius> <tether strength>
     <water droplet chain ID>
```

3 A word about units

I am making a special, very short chapter on units. In MMB 2.10 and earlier, some forces such as atomSpring, springToGround, atomTether, etc. took Å as the unit for dead lengths and ground locations. For consistency, we are going back to nm for the length unit. This is because internally all the math is done in nm, kJ/mol, ps, and daltons (g/mol). This implies that spring constants are in kJ/mol/nm^2. For example, if you want to make a spring which in Amber99 units (Å, kcal/mol, ps) would be 310 kcal/mol/Å^2, the equivalent spring in our choice of units would be 129790.8 kJ/mol/nm^2.
Please note that if you have any dead lengths or ground locations in your MMB 2.10 or earlier script which you are using with MMB 2.11, you will need to manually change them from Å to nm.

4 Forces

In this Appendix, we describe options for using the `baseInteraction, aromatic` two-residue forces, the `atomSpring, atomTether,` and `springToGround forces`, and the `contact` steric forces. Note that since forces are additive, there is no hard limit on how many forces can exist in the system or even acting on a single residue, base, or atom.

4.1 baseInteraction

The syntax for this command 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>
```

The following combinations of first base pairing edge, second base pairing edge, and glycosidic bond orientation are permitted:

```
WatsonCrick WatsonCrick Cis
WatsonCrick WatsonCrick Trans

WatsonCrick Hoogsteen Cis
WatsonCrick Hoogsteen Trans

WatsonCrick SugarEdge Cis
WatsonCrick SugarEdge Trans

Hoogsteen Hoogsteen Cis
Hoogsteen Hoogsteen Trans

Hoogsteen SugarEdge Cis
Hoogsteen SugarEdge Trans

SugarEdge SugarEdge Cis
SugarEdge SugarEdge Trans

WatsonCrick Bifurcated Cis
Stacking3 Stacking5 Cis
Stacking5 Stacking5 Trans
Stacking3 Stacking3 Trans
HelicalStackingA3 HelicalStackingA5 Cis
Superimpose Superimpose Cis
```

You might notice that some of these are actually not in the Leontis and Westhof classification. These are explained below:

- Stacking* simply specifies a stacking interaction between consecutive residues on a chain. The numbers indicate which face is interacting on each base. For example:
  `baseInteraction A 120 Stacking3 A 121 Stacking5 Cis`

  Means that the face of base 120 which would ordinarily point towards the 3' end of the strand in a helix, will be stacked on the face of base 121 which would ordinarily point to the 5' end of the helix.

- HelicalStacking* works the same as Stacking, but adds the offset appropriate for consecutive bases in a helix. HelicalStackingA3/HelicalStackingA5 is automatically

applied to all consecutive bases in helices, unless you specify `setHelicalStacking FALSE`. MMB assumes an A-form helix exists whenever it finds three consecutively numbered RNA residues on a single strand Watson-Crick base paired with three consecutively numbered residues on the same or another single RNA strand. If you want to generate a helix where this is not the case, you should manually apply HelicalStackingA3 / HelicalStackingA5 interactions.

4.2 nucleicAcidDuplex

This command generates WatsonCrick/WatsonCrick/Cis interactions between two specified segments on the same or different RNA chains. It is a shortcut for manually specifying each such interaction for every pair of canonically interacting residues in the duplex. The syntax is:

```
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)`

For example:

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

Makes the segments between residues 1 and 3 (inclusive) and between 10 and 8 (inclusive) into two halves of a duplex, by applying a base pairing interaction between 1 and 10, 2 and 9, and 3 and 8.

4.3 NtC

This command is new to 17.8. Previously, we would use baseInteraction's to impose the bse stacking geometry in double helices. These we would even impose automatically whenever three consecutive Watson-Crick base pairs were detected. Actually you should now turn off that behavior if you want to use NtC's:

```
setHelicalStacking False
```

WatsonCrick/WatsonCrick/Cis interactions between two specified segments on the same or different RNA chains. It is a shortcut for manually specifying each such interaction for every pair of canonically interacting residues in the duplex. The syntax is:

```
NtC  <chain identifier>
                    <first residue number>
                    <second residue number, almost always the
                next consecutive residue>
                    <NtC class>
                    <force constant>
```

For example to force the consecutive residues 1 and 2, on chain D, into A-form helical stacking, with force constant 1.5, we would issue:

```
NtC D 1 2 AA00 1.5
```

As a side note, the force constant 1.5 for NtCs, and `forceMultiplier` of about 200 for the baseInteraction's, seem to be a pretty good combination for many purposes.

4.4 Units

Before we describe the various variants of user-applied springs, let's clarify the units. MMB and molmodel use nm, kJ/mol, ps, and daltons (a.k.a. atomic mass units = u, or g/mol). The Amber99 force field, on the other hand, uses nm, kcal/mol, and ps, and u. The user

4.5 atomSpring

The `atomSpring` command creates a linear spring connecting two atoms. Two optional parameters (square braces []) specify the dead length and spring force constant.

```
atomSpring <first chain ID>
           <first residue number>
           <first atom name>
           <second chain ID>
           <second residue number>
           <second atom name>
           [<dead length>
           [<spring constant>]]
```

4.6 atomTether

The `atomTether` command, as the name implies, applies no force if the distance between atoms is less than a certain `<dead length>`, and applies an attractive force with Hookean `<spring constant>` when the distance exceeds the former. Default values for the last two parameters are 0.0 and 3.0, respectively, as they are for `atomSpring`.

Make `<spring constant>` large for a strict "dog leash" or small for a permissive restraint.

```
atomTether <first chain ID>
           <first residue number>
           <first atom name>
           <second chain ID>
           <second residue number>
           <second atom name>
           [<dead length>
           [<spring constant>]]
```

4.7 springToGround

The `springToGround` command creates a linear spring connecting a specified atom and a specified location in Ground. Two optional parameters (square braces []) specify the dead length and spring force constant.

```
springToGround <atom chain ID>
               <atom residue number>
               <atom name>
               <X location in Ground>
               <Y location in Ground>
               <Z location in Ground>
               [<dead length> [<force constant>]]
```

4.8 alignmentForces

The `alignmentForces` keyword supports a series of parameters and commands, and supersedes the old `threading` and `gappedThreading` commands. Like those commands, it applies cross-strand springs connecting like-named atoms in sequence-aligned residues. By default it does not require an alignment; instead it figures out a global alignment for you using a dynamic programming algorithm and the BLOSUM62 substitution matrix (thanks Seqan!). This command should work with any biopolymer type.

Parameters within `alignmentForces` apply to any `alignmentForces` commands following that parameter. They do not apply to any commands that appear above it in the input file. So it is a good habit to specify all parameters first, then start with the commands. This parameter says that you want an ungapped alignment:

```
alignmentForces     noGap
```

It is implemented as a prohibitively high gap penalty for the alignment, which is always a gapped alignment. In any event, if you want to later re-enable gapped alignments, issue:

```
alignmentForces     gapped
```

A cool new feature that comes with `alignmentForces` is one which allows you to set the dead lengths of the springs not to zero as before, but to some specified fraction of their initial lengths. Issue:

```
alignmentForces deadLengthFraction <fraction>
```

.. where if `<fraction>` in the interval (0,1], the springs will be set to `<fraction> * (initial length).` By default, `<fraction>` is zero, exactly like the old behavior. If you want to return to that behavior, explicitly set this to zero.

The spring constant is now set separately from the actual alignment command:

```
alignmentForces forceConstant <force constant>
```

.. where if `<force constant>` should be > 0.0. The units of the force constant are kJ/(mol-nm^2).

The only behavior which has been lost in `alignmentForces` is the old backbone-only threading, but I can reinstate it upon request.

You can specify the stretches of residues to be aligned (e.g. if you want to align based on a certain domain). Actually I strongly recommend doing this, because it is easy to get a slightly (or terribly) incorrect local alignment when doing a global alignment based on the full length sequence – Bioinformatics 101! On the other hand if you tell it the boundaries of regions you know to be aligned, you will probably agree with the resulting global alignment. Anyway, it works like this:

```
alignmentForces     <chain 1 ID>
                <start residue 1>
                <end residue 1>
                <chain 2 ID>
                <start residue 2>
                <end residue 2>
```

On the other hand, if you trust SeqAn to do a global alignment, just give it the chain IDs:

```
alignmentForces     <chain 1 ID>
                <chain 2 ID>
```

You should always inspect the alignment. Search for "`SeqAn sequence alignment follows:`" in the (admittedly) verbose output. You may see something like this:

```
        --PGVGCVPAAEHRLREEILAK
                  |  ||   |
```

```
        DYDAIPWLQNVEPNLRPKLL--
```

..where "-" are deletions, and "|" are perfect matches. Both matches and mismatches will get `atomSpring's`. Indels of course will not.

4.9 contact

You can also apply space-filling Contact spheres to a range of residues using the `contact` command. (The idea is similar to that of the parameters `addSelectedAtoms` and `addAllHeavyAtomSterics`)

```
contact   <contact type>
          <chain identifier>
          <residue number for first residue>
          <residue number for last residue>
```

The first residue should be lower numbered than the second, and both residues should be on the same chain. You can also issue:

```
contact   <contact type>
          <chain identifier>
```

And the contact spheres will be applied to every residue on the specified chain.

There are two kinds of permitted values of `contact type`. In the fixed type, the atom identities are hard-coded and can't be modified by the user, but the contact sphere radii and stiffness (both of which are the same for all atoms regardless of atom name) correspond to the `excludedVolumeRadius` and `excludedVolumeStiffness` parameters which are set in the MMB input file (e.g. commands.dat). These include:

`AllAtomSterics` : Puts one sphere on each atom of the chain, except for the end caps on proteins (when used).
`AllHeavyAtomSterics` : Puts one sphere on each atom of the chain EXCEPT hydroges, and again except for the end caps on proteins.
`RNABackboneSterics` : Puts one sphere on each of the following atoms: P, O5*, C5*, C4*, C3*, and O3*. An error will result from attempting to apply this to proteins, as anytime when you attempt to put sterics on an atom which doesn't exist on a given residue.

The second type of sterics are user configurable, in the parameter file (e.g. parameters.csv). Here the user can choose on which atoms to put the spheres, with a maximum of four atoms. The radii and stiffness can be controlled separately for each atom name. A different choice of zero to four atom names can be chosen for each residue type (4 residue types for RNA, 20 for protein). The user can add as many steric schemes to the parameter file as he/she wishes; as supplied the `parameters.csv` file has two: `SelectedAtoms` and

`ProteinBackboneSterics`. For the first one, the parameters look like:

RECORD	A		SelectedAtoms	SelectedAtoms	X	P	C4*	N9
RECORD	C		SelectedAtoms	SelectedAtoms	X	P	C4*	N1
RECORD	G		SelectedAtoms	SelectedAtoms	X	P	C4*	N9
RECORD	U		SelectedAtoms	SelectedAtoms	X	P	C4*	N1

The second column is the residue type, and columns 7,8, and 9 are the atom names. Note that the glycosidic nitrogen is named differently for purines vs. pyrimidines. Subsequent columns give the sphere radii, stiffnesses, and information to identify these as `contact parameter` entries. Parameters become available for use immediately upon being entered in the parameter file, much as for MD force field parameter files.

It is also possible to apply a specified steric scheme to all residues within a certain distance of a specified residue. The distance is measured by between nearest atoms spanning the two residues. The syntax is:

```
applyContactsWithin <radius (nm)> <contact scheme> <chain>
<residue>
```

4.10 Restraining to ground

Much as residues can be constrained to each other (see next chapter), any residue of any chain can also be restrained to ground, meaning that a force can be applied to pull all six translational-rotational degrees of freedom to an equilibrium position and orientation in Ground:

```
restrainToGround <chain ID> <residue number>
```

Keep in mind that unlike a constraint, a restraint acts as a spring and thus allows some displacement with respect to ground. Any displacement at the end of a stage is carried over to the next stage, potentially leading to a "creeping" effect. Two parameters which are relevant to this command are `restrainingForceConstant` and `restrainingTorqueConstant`. These set the translational and angular restitution force constants.

4.11 Density based force field

As explained in the tutorial, MMB's density based force field is formulated following Klaus Schulten's MDFF as follows:

Where *i* is the atom index, m_i is the mass of atom *i*, is the electronic density at the nuclear position of atom *i*, A is a user-adjusted scaling factor, and is the gradient operator. Accordingly, is the density-derived force vector applied to atom *i*. This is computed for and applied to every atom *i* in the system.

To turn on the density based force field on or off, you just need to specify which chains you want to be subjected to such forces. For instance:

```
fitToDensity
```

Specifes that all chains in the system should be fitted to the map. If you only want certain chains to be fitted, with the remaining chains not subjected to these forces, just specify each chain to be fitted like this:

```
fitToDensity <chain ID>
```

Lastly, if you only want certain stretches of residues to be fitted, you can issue:

```
fitToDensity <chain ID> <start residue> <end residue>
```

.. and only the residues starting at `<start residue>` and ending at `<end residue>` of chain `<chain ID>` will be fitted.

Your density map must be in XPLOR, OpenDX or Situs format. To specify the location of the density map, file, use:

```
densityFileName <density file name>
```

The scaling factor (A in the equation above) defaults to unity, but you can set it to any floating point number (including negative numbers) as follows:

```
densityForceConstant     <scale factor>
```

In nucleic acid density maps generated by CryoEM, very often the phophates are not visible because of their negative charge. Trying to fit these leads the phosphates (which have high atomic numbers) being fitted to densities they don't belong to. You can leave out the phosphates (P, O5', O3', OP1, and OP2) from the fitting procedure by setting `densityFitPhosphates` to false (or 0):

```
densityFitPhosphates     <bool>
```

Note that this will slow down your run A LOT. Slowdown is linear with number of nucleic acid residues with density forces active. We will work on making it work faster.

Another recent (2019) addition is noise. If you use synthetic density maps, e.g. for benchmarking new fitting methods, you may note that there is a dearth of plug and play

programs for adding noise. The literature is quite clear that white noise won't cut it – you need correlated noise. I provide a feature to add noise, inspired by Planck's Law for blackbody radiation. I reason that noise is thermal and both photons and phonons are bosons, so Bose-Einstein statistics apply. The unit cell is taken as an oven, with the lowest-frequency mode having a half-wavelength equal to the cell dimension in that direction, and the highest-frequency mode being set by the Nyquist frequency. This is admittedly rather hand-wavy thinking. For one thing it completely ignores the macromolecular structure which should be quite a noise generator of its own. Anyway, with time we will see how it compares with real noise. The important thing for now is it gives you a parameter – temperature – which lets you adjust the frequency distribution of the noise in an intuitive way. Low temperatures emphasize blobby low-frequency noise, high temperatures give more fuzzy high-frequency noise. Continuing the hand-waving, I set the speed of light c , the Boltzmann constant k_B, and Planck's constant h, to unity. Then in Planck's law I divide through by $h\nu$, getting the number density:

$$N = 1 / (e^{\nu/T}-1)$$

To generate the noise MMB sums over all of the frequency components, scaled by $N(\nu, T)$. This gives noise *amplitude.* After the sum, we square the amplitude at each grid point to get intensity. The intensity is then summed to the density. I reason density is more of an intensity than an amplitude. Following that argument, it would have been better to sum the noise and density amplitudes before squaring, but of course we don't have the density amplitude.

The temperature is set by the user as:

```
densityNoiseTemperature <double, defaults to 0.0>
```

There is also an overall scale factor:

```
densityNoiseScale <double, defaults to 0.0>
```

If this is set to zero (or close to zero) then noise generation is turned off. In our manuscript we mostly set to around 0.1. The signal-to-noise ratio is another important quantity.

$$SNR = \sum_{x,y,z} \frac{density}{noise}$$

Where x,y,z sum over all grid points and density and noise are both intensites. To get this quantity, just search for "`signalToNoiseRatio`" in the admittedly-verbose stdout. Once the stage is done you should find the following density maps in your working directory:

```
noise.xplor
density.xplor
noisyMap.xplor
```

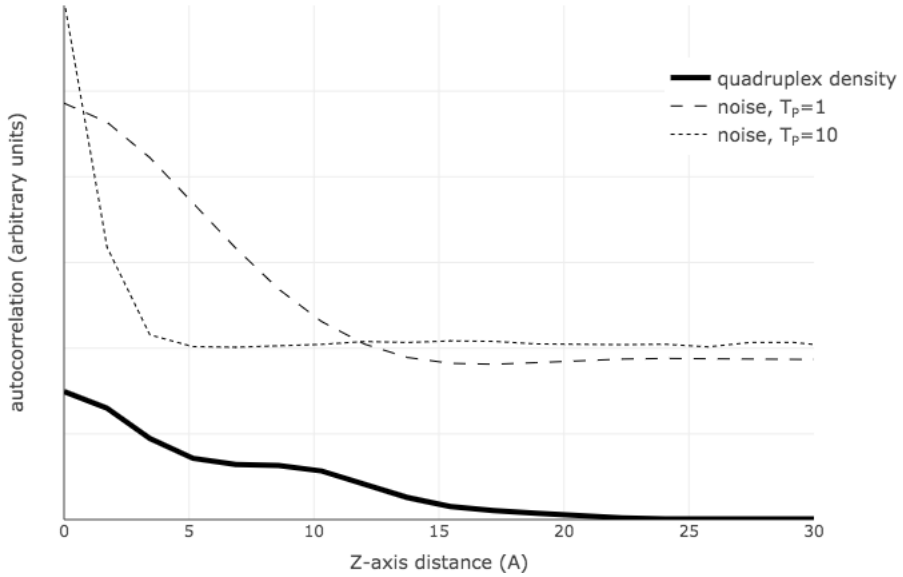
These are separate maps of the noise, the "clean" density (which should be identical to what was read from the input `densityFileName`, and the density + noise.

Note that when we summed over all frequencies that went almost like nx*ny*nz. Then we do that over all grid points so in the end the complexity is $(nx*ny*nz)^2$. So this can be veeery slow for maps of nontrivial size. Thus it is a good idea to save `noisyMap.xplor` for later use, rather than generating new noise every time.

Lastly, you can choose to compute the autocorrelation function along the Z-axis, for the noise and the noiseless input density. To turn that on, just set this to `True`:

```
densityNoiseComputeAutocorrelation <bool, defaults to 0>
```

The autocorrelation gets written to the terminal, just search for "`densityAutocorrelation`." Here I show a plot from our (currently) in-prep manuscript (Emanuel Peter et al., 2019). It shows the density autocorrelation for a density map simulated at 5Å from 2o4f.pdb, and also for the noise at temperatures of 1 and 10.



4.12 Electrostatic Density force-field

On the same idea as the Density Map fitting, you can provide MMB with an Electrostatic Potential grid, typically from APBS. The residues you select for fitting will then be driven into the map according to the partial charges of the atoms. Negative ones will tend to go in positive volumes and positive ones to negative volumes.
As MMB use Amber to determine the atoms charges, it is strongly recommended to use an electrostatic map computed with an Amber force field.

The usage is exactly the same as described for `fitToDensity` with the following commands:

```
fitElectroDensity
electroDensityFileName
electroDensityForceConstant
```

4.13 Physics where you want it

Physics where you want it, introduced in release 2.4, allows you to turn on the all-atoms force field only for certain regions of your system, referred to as the *physics zone*.

To specify a range of residues to be added to the physics zone, use:

```
includeResidues <chain ID> <first residue in range> <last
residue in range>
```

Sometimes it will be convenient to include all residues within a certain radius of a specified residue. For this you would use:

```
includeAllResiduesWithin <distance> <chain ID> <residue
number>
```

Note that `includeResiduesWithin` is an alias for `includeAllResiduesWithin` . The `distance` (in Å) is measured between key atoms, CA for protein and C4* for RNA and DNA.

You can also simply set `physicsRadius` to a value > 0. If this is set, all residues within `physicsRadius` of the "flexible"atoms will be included in the *physics zone.* "Flexible" atoms are defined as atoms belonging to a mobilized body of mass < 40. This is technically a parameter rather than a command, so is listed in that section separately. The syntax is just:

```
physicsRadius <radius>
```

Default behavior is for all atoms to be subjected to the non-bonded force field terms. If that is what you want, just don't specify either of the above commands.

You can create physics zones at all interfaces between atoms belonging to different mobilized bodies. For example, if you have a domain hinge bending protein, with the hinge flexible, it would add the hinge plus residues at the domain-domain interface to the physics zone. something to watch out for, is that a discontinuous domain would get its inter-fragment interface added to the physics zone. Syntax is:
`includeIntraChainInterfaceResidues <chain> <depth>`
Where `<chain>` is the biopolymer chain ID in question, and `<depth>` is the greatest atom-atom distance, with atoms of different mobilized body index, that will lead to those atoms being included in the physics zone.

You can also add specified *inter*-chain interfaces to the physics zone. In the most general polymorphism, you can specify the interface between one chain or set of chains, and another chain or set of chains:

```
     physicsInterfaces <depth (nm)> <chain 1> [<chain 2>
[<chain 3> [ ...etc]]]  Versus <chain 1> [<chain 2> [<chain 3>
[ ...etc]]]
```

.. You can call this command to get an informative message about the other polymorphisms.

Lastly, we have found that small chemical groups such as methyl or alcohol can spin out of control in the absence of viscous forces, leading to small time steps and excessive computational expense. To deal with this, you can scale the inertia of such small groups with:

```
smallGroupInertiaMultiplier <inertia scale factor>
```

Any nonnegative floating point number can be used here; we suggest 11.0.

4.14 Potential rescaling with the "Scrubber"

In Flores and Altman (RNA 2010) we found that kinetic trapping occurs often in computational RNA folding, as it does experimentally. To get out of these traps we created the *scrubber*. Potential rescaling refers to cyclically varying forces. In MMB, we use a rectangular waveform. For a fraction of the time (1 - `dutyCycle`) all forces (including baseInteraction's, sterics, Amber99 force field, springToGround's, etc.) will be turned off. Then for the remainder of the period (`dutyCycle`) these forces will be turned back on. The length of the period is set with the `scrubberPeriod` parameter (in ps, as always).

```
dutyCycle <"on" fraction>
scrubberPeriod <potential rescaling period, in ps>
```

This is used in some of the MMB tutorial examples.

4.15 addRingClosingBond

This add a ring closing bond. It only creates bonds between atoms in the same chain. Use it like this:

```
addRingClosingBond <chainID> <residueID1> <atomName1>
<bondCenterName1>  <residueID2>  <atomName2> <bondCenterName2>
```

bondCenter's are named as e.g. `bond1, bond2, .. bondN`. For example, in a disulphide bridge (bonding atoms `SG`), you need to specify `bond1,` since `bond1` is bonding to `CB`.

5 Mobilizers and constraints

In this Appendix we describe `mobilizer` commands, which define or modify the internal coordinate topology of the molecule as well as `constraint` commands, which add constraint equations that reduce the degrees of freedom of the system.

It is important to keep in mind the crucial difference between these two in Internal Coordinate Mechanics. A `mobilizer` command can reduce or increase the number of bodies that exist in a system; in the former case you will always save computer time. On the other hand a `constraint` command adds constraint equations which must then be solved; while the net effect depends on masses and forces, computational cost typically increases. Mobilizers control bond mobilities, which here can `be Free, Torsion,` or `Rigid`.
`Free` means that the bond can change its length, angle, and dihedral.
`Torsion` means it can change only its dihedral angle.
`Rigid` means it has no degrees of freedom.

One must also avoid overconstraining the system. For example, if two rigid molecules are already `Weld`'ed (see below) to each other, do not put additional constraints on this pair of

molecules, even if they are nominally applied to different residues. While this is easy to keep track of for two bodies, watch out for more insidious ways of overconstraining. For example, if A is `Weld`'ed to B, and B is `Weld`'ed to C, do not then `Weld` C to A.

5.1 mobilizer

The `mobilizer` keyword is used for specifying the bond mobilities for a stretch of residues. This command is overloaded. The first variant has the following syntax:

```
mobilizer      <bond mobility>
          <chain identifier>
          <first residue number>
          <last residue number>
```

The first residue should be lower numbered than the second, and both residues should be on the same chain. Bond mobility can be set to `Free, Torsion, Rigid,` or `Default`. The "`Default`" bond mobility is special, as we'll explain in a moment. Don't forget you can use the keywords `FirstResidue` or `LastResidue`, or do arithmetic on the residue numbers using the "`+`" operator, as described earlier.

You can also simply say:

```
mobilizer      <bond mobility>
          <chain identifier>
```

… and this will set ALL residues in chain `<chain identifier>` to `<bond mobility>`.

Lastly, you can say:

```
mobilizer      <bond mobility>
```

... and this will set all residues in ALL chains to `<bond mobility>`.

5.2 applyMobilizersWithin

The `applyMobilizersWithin` command is used to specifying the bond mobilities for all biopolymer residues within a certain radius of a specified residue. It has the following syntax:

```
applyMobilizersWithin    <bond mobility>
                    <radius>
                    <chain identifier>
                    <residue ID>
```

constraints

The `radius` is measured between nearest atoms spanning the two residues and (like always) in Å. The acceptable values of `<bond mobility>` are as listed above.

5.3 mobilizeInterfaces

The `mobilizeInterfaces` command is used to specifying the bond mobilities for all biopolymer residues within a certain distance of all interfaces of a given biopolymer chain or chains. The syntax is:

```
mobilizeInterfaces  <interface depth>
                    <bond mobility>
                    <chain 1> [<chain 2> [<chain 3> […etc]]]
```

The `<interface depth>` is measured as the minimum distance between atoms on different chains across an interface. Note that this counts over *all* atoms, not just the Cα or C3*. We are now using OpenMM's neighborlisting for this, which is pretty economical even if you aren't set up to use the GPU. `<bond mobility>` is that desired at the interface – `Rigid, Torsion, Free,` or `Default.` `<chain 1,2, etc>` is the list of chains forming a complex, whose interfaces with the *rest* of the system you are interested in. For example, say you have a complex of chains A, B, and E. If you issue:

```
mobilizeInterfaces 0.6 Default A B
```

Then all residues at the interface between the complex AB, and chain E, to a depth of 0.6 nm, will get bond mobility `Default`. Note that this will do nothing in particular to the interface between A and B! In this case you could just as easily have issued:

```
mobilizeInterfaces 0.6 Default E
```

If you like to be explicit (a good habit, by the way), you can use the alternate syntax:

```
mobilizeInterfaces  <interface depth>
                    <bond mobility>
                    <chain 1> [<chain 2> [<chain 3> […etc]]]
                    Versus
                    <chain I> [<chain II> [<chain III>
                [….etc]]]
```

This sets the bond mobility to `<bond mobility>,` to a depth of `interface depth>`, for the interface between the set of chains (1,2,3..) and the chains (I,II,III..). In our example, you would issue:

```
mobilizeInterfaces 0.6 Default A B Versus E
```

5.4 singleBondMobility

The `singleBondMobility` command is used for specifying the bond mobility for a single bond:

```
singleBondMobility  <chain identifier for first atom>
          <residue number for first atom>
          <atom name for first atom>
          <bond mobility>
          <chain identifier for second atom >
          <residue number for second atom >
          <atom name for second atom>
```

The two atoms should be covalently bonded to each other, of course.

5.5 psiPhiMobility

`psiPhiMobility` is used for specifying the bond mobility for the bonds connecting the N to CA, and the CA to C on the protein backbone, along a given stretch of residues. It is equivalent to issuing the `singleBondMobility` command for the two mentioned bonds, for each residue in the range.

```
psiPhiMobility <chain ID>
               <residue number for first residue in range>
               <residue number for last residue in range>
               <bond mobility (Free, Torsion, or Rigid)>
```

You can also skip the residue numbers:

```
psiPhiMobility <chain ID>
               <bond mobility (Free, Torsion, or Rigid)>
```

.. and this will apply mobilizers to whole chain, from first to last residue. Lastly, you can skip the chain ID:

```
psiPhiMobility <bond mobility (Free, Torsion, or Rigid)>
```

.. and this will apply mobilizers to whole chain, for every protein chain in the system.

`psiPhiMobility` is simply a shortcut for a bunch of `singleBondMobility` commands, and so works in exactly the same way as the latter. This means that it is applied late – see the section "**Order of application of mobilizers**."

5.6 Default mobilizers

It is important to understand what is the default setting for the mobilizers in your system.

The default bond mobility leaves most bonds set to `Torsion,` but there are also some

constraints

`Rigid` bonds, depending on the residue type and atoms it connects. For instance, the bond mobilities for an RNA residue look like this:

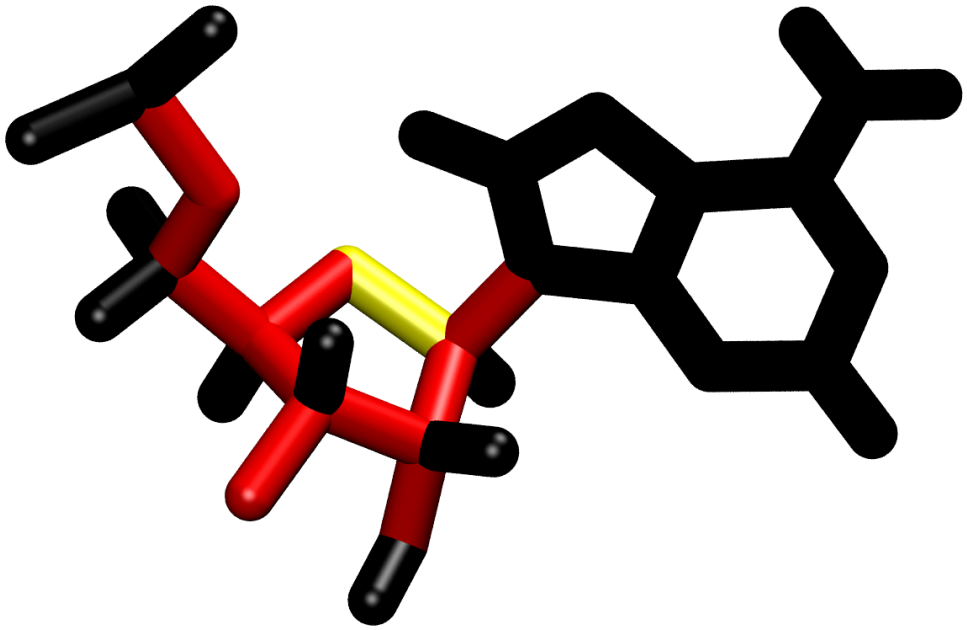


Figure 1 : Default bond mobilities for an RNA residue

Black: `Rigid`**; Red:** `Torsion`**; Yellow:** `Free`**.**

Similarly for a protein residue, most bonds are also `Torsion`. There are certain bonds and groups that are set to `Rigid`:

All peptide bonds (C-N)
All covalent bonds between hydrogens and heavy atoms
Proline N-Cα (This may be changed to `Torsion`)
Guanidinium group (Arginine CZ-NH1, CZ-NH1, CZ-NE)
Amide groups (Asparagine Cγ-Nδ2, Glutamine Cδ-NE2)
Cyclic groups in Tryptophan, Histidine, Phenylalanine, Tyrosine. EXCEPT that ring closing bonds are special (nonexistent topologically, subject to bonded MD force field terms):

- Tryptophan Cδ2-Cγ, CZ3-CH2
- Histidine, Tyrosine and Phenylalanine Cδ2-Cγ
- Proline Cδ-N

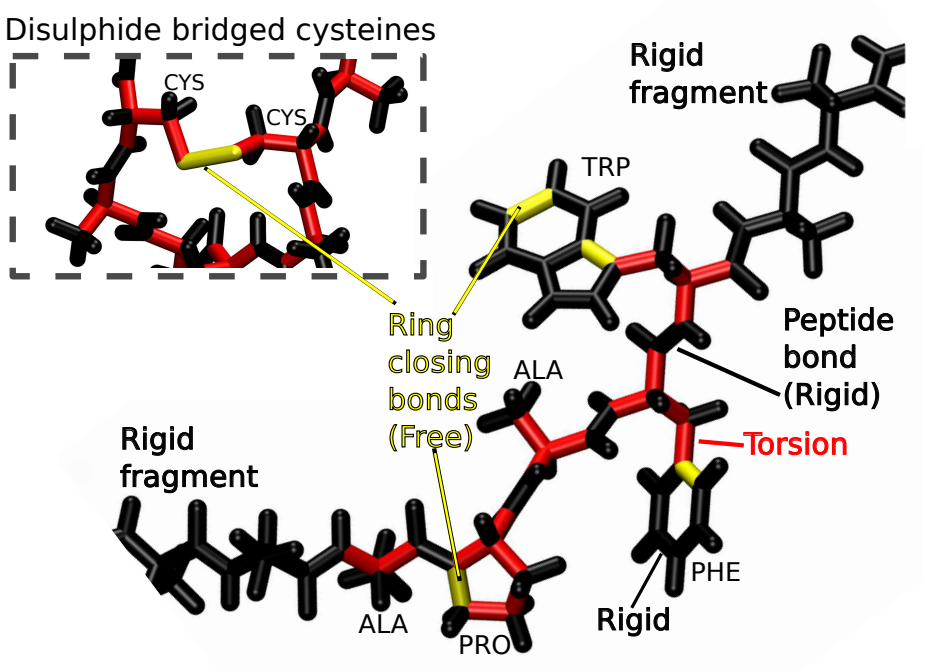


Figure 2 : Bond mobilities for proteins. By default, most bonds have Torsion bond mobility (red). Peptide bonds are Rigid, as are guanidinium and amide groups. Hydrogens and double-bonded oxygens are also connected with Rigid bonds. The user can create a ring closing bond which will have Free bond mobility. Ring closing bonds are special in that they do not topologically connect atoms, only apply bonded forces (bond stretch, angle bend, dihedral) – recall that closed cycles are not desirable in internal coordinates. Cyclic groups of Histidine, Tryptophan, Phenylalanine, and Tyrosine (but not Proline) are connected with Rigid bonds and so form a single body; the ring closing bonds do not change this. Peptide bonds are Rigid. All of these default bond mobilities can be overridden by the user. For example, the segments on either end of the chain have here been made Rigid. Lastly, the user can also create a disulphide bridge between cysteines using a ring closing bond (inset).

5.7 Order of application of mobilizers

In order to get the desired result out of MMB, you should understand the order in which these commands are applied. They go like this:

1. `mobilizer` (for `Rigid`, `Free`, and `Torsion`)
2. `applyMobilizersWithin` (for `Rigid`, `Free`, and `Torsion`)
3. `mobilizeInterfaces` (for `Rigid`, `Free`, and `Torsion`)
4. `mobilizer`, `applyMobilizersWithin`, and `mobilizeInterfaces` (for `Default`)
5. `singleBondMobility` (incl. `psiPhiMobility`)

A common mistake is to forget that before any commands are applied, all chains have a default bond mobility, as described above. Note also that the "`Default`" bond mobility isn't actively applied to residues – instead when you specify this, all *other* modifications to the residue bond mobility are removed, so it is simply never changed from its original bond

constraints

mobility.

Here is a simple example:

```
protein A 1 AAAAAA
mobilizer Rigid A 1 6
mobilizer Default A 3 4
```

Results in two `Rigid` stretches (1 to 2 and 5 to 6) – the output looks something like this:

```
/Users/Sam/svn/RNAToolbox/trunk/src/MobilizerContainer.cpp:44
Mobilizer stretch 0 BondMobility = Rigid
/Users/Sam/svn/RNAToolbox/trunk/src/MobilizerContainer.cpp:45
chain= A from residue 1 to 2
/Users/Sam/svn/RNAToolbox/trunk/src/MobilizerContainer.cpp:44
Mobilizer stretch 1 BondMobility = Rigid
/Users/Sam/svn/RNAToolbox/trunk/src/MobilizerContainer.cpp:45
chain= A from residue 5 to 6
```

5.8 constraint

The `constraint` command is used for specifying constraints to weld residues or monoAtoms together:

```
constraint     <chain identifier for first residue>
               <residue number for first residue>
               Weld
               <chain identifier for second residue>
               <residue number for second residue>
```

The two welded residues can be on different chains; in fact either or both residues can be in RNA or protein chains. The weld is applied on C3* atoms of RNA residues and on C atom s of protein residues. There is no preference for residue number ordering.

You can also specify which atoms you want welded, as follows:

```
constraint     <first atom chain identifier>
               <first atom residue>
               <first atom name>
               Weld
               <second atom chain identifier>
               <second atom residue number>
               <second atom name>
```

Lastly, you can weld to ground, either specifying the atom to be welded or using the default:

```
constraint     <chain identifier>
               <residue number >
               <atom name>
               Weld Ground
```

or:

```
constraint     <chain identifier>
               <residue number >
               Weld Ground
```

This syntax treats monoAtoms and biopolymer atoms on an equal footing, except that in the case of monoAtoms you cannot avoid providing the atom name.

5.9 Constraining to ground

Just as residues can be constrained to each other, any residue of any chain can also be constrained (rigidly attached) to ground:

```
constrainToGround <chain ID> <residue number>
```

See *Appendix: Parameters* for an explanation of the `constraintTolerance` parameter, relevant to this command.

Much more efficient are a couple of variants of this command. For example:

```
constrainToGround
```

(with no parameters) attaches each chain to ground using a Weld rather than a Free mobilizer. Thus rather than granting 6 DOFs and then removing them with constrain equations, the DOFs never exist to begin with.

Similiarly, you can choose the mobilizer type (`Free` vs. `Weld`) for all chains by issuing:

```
rootMobilizer <"Free" | "Weld">
```

Or, you can choose the mobilizer type for a specific chain by issuing:

```
rootMobilizer <Chain> <"Free" | "Weld">
```

5.10 Constraining rigid segments to each other or to ground

In the antibody design example (see Tutorial), we have a protein which has two rigid segments and one flexible segment. To prevent the two rigid segments from moving with respect to ground, we welded them to ground. Alternatively, maybe we could have welded

the rigid segments to each other, so the protein as a whole could move with respect to its binding partner (or ground, for that matter). Sometimes you may want a chain to have many rigid segments, all welded either to ground or to a specified residue. MMB has a convenient command for this.

If you want to weld all rigid segments of all chains to ground, just issue:

```
constrainChainRigidSegments
```

If you want to weld the rigid segments of a specified chain to ground, issue:

```
constrainChainRigidSegments <chain ID> Ground
```

where `<chain ID>` refers to the chain in question.

Lastly, if you want to weld the rigid segments of a specified chain to a specified residue (on the same chain), issue:

```
constrainChainRigidSegments <chain ID> <residue ID>
```

In this latter case, all the rigid segments in chain `<chain ID>` will be welded to the same residue `<residue ID>`. This means that the rigid segments will move together, allowing rigid body motions of the entire chain. If you want several chains to move together, just add a `constraint` command.

5.11 constrainInterfaces

The `constrainInterfaces` command is used to apply a `Weld` constraint across one or more pairs of biopolymer chains. within a certain distance of all interfaces of a given biopolymer chain or chains. The syntax is:

```
mobilizeInterfaces  <interface depth>
                    <chain 1> [<chain 2> [<chain 3> […etc]]]
```

The `<interface depth>` is measured as the minimum distance between atoms on different chains across an interface. If the minimum distance between the chains is greater than this, no constraint will be applied. The constraint will be applied to the first pair of atoms found, which spans the two chains and whose internuclear distance is smaller than `<interface depth>`. Note that this counts over *all* atoms, not just the Cα or C3*. We are now using OpenMM's neighborlisting for this, which is pretty economical even if you aren't set up to use the GPU. `<chain 1,2, etc>` is the list of chains forming a complex, whose interfaces with the *rest* of the system you are interested in. For example, say you have a complex of chains A, B, and E. If you issue:

```
constrainInterfaces 0.6 A B
```

Then chains A and B will each separately be constrained to chain E, unless one of the two former chains is more than 0.6 nm from E, in which case that chain will not be constrained to E. Note that this will not apply a constraint between A and B! In this case you could just as easily have issued:

```
constrainInterfaces 0.6 E
```

If you like to be explicit (a good habit, by the way), you can use the alternate syntax:

```
mobilizeInterfaces  <interface depth>
                    <chain 1> [<chain 2> [<chain 3> […etc]]]
                    Versus
                    <chain I> [<chain II> [<chain III>
                […etc]]]
```

This creates all possible pairwise constraints between `<chain 1,2,..etc>` and `<chain I, II..etc>` , skipping pairs of chains with minimum separation greater than `<interface depth>`. You would get the same result as before if you issue:

```
constrainInterfaces 0.6 Default A B Versus E
```

5.12 Coupling protein backbone ψ and φ angles

If you are modeling homomultimers or for some other reason wish to impose symmetric motion of protein backbones, you can use the following command:

```
couplePsiPhiAngles <Chain A> <start residue A> <end residue A>
<Chain B> <start residue B> <end residue B>
```

This forces corresponding residues in chains A and B, over the specified range, to have the same ψ and φ angles. You need to make sure that the ranges `<start residue A> <end residue A>` and `<start residue B> <end residue B>` have the same number of residues. However strictly speaking you could get away with these stretches not having the same sequence, since the coupled-coordinate constraints apply only to the backbone angles.

6 Direct structure editing

In this chapter we talk about editing structure *directly*, meaning modifying internal or Cartesian coordinates of structures, adding or removing atoms, etc. For now this chapter is quite short, other features will be documented soon.

6.1 Initial displacement

This command simply displaces the designated chain by a given Cartesian vector, with respect to its position in the input structure file:

```
initialDisplacement <chain> <X> <Y> <Z>
```

6.2 Imposing protein secondary structure

Have you ever simply wanted to impose a certain secondary structure in a certain region of your model? Say, make a helix-turn-helix into a single continuous helix? Well, with the new `setPhiPsiAngles` command you can do just that. It sets the phi, psi, and peptide dihedral angles to the defaults for `Alpha`, `ParallelBeta` and `AntiParallelBeta` secondary structures, overriding whatever values these dihedrals may have taken from the input structure file. The syntax is:

```
setPhiPsiAngles <chain ID> <start residue> <end residue> <
Alpha | ParallelBeta | AntiParallelBeta>
```

These are applied *after* structure matching. If the command is issued multiple times, the dihedral angles are set in the order the commands were issued.

6.3 Introducing a substitution mutation

A common structure editing operation is to substitute one residue type for another. The syntax is:
`substituteResidue <chain> <residue ID> <new residue type>`
The residue at position `<residue ID>` will simply be replaced by one of type `<new residue type>`. If you provided an input structure file, MMB will match all the internal coordinates it can based on identical atom names, and use default values for the remaining internal coordinates. For example if your mutate alanine to valine, it will match the Cα and Cβ positions, but will choose a position for Cγ1 and Cγ2 based on default bond lengths,

angles, and dihedrals. Note that substitutions in general will need to be equilibrated to ensure reasonable interatomic contacts.

6.4 Introducing an insertion mutation

Another common structure editing operation is to insert a residue. The MMB syntax for this is:
`insertResidue <chain> <residue ID> <inserted residue type>`
MMB will simply insert a residue at position `<residue ID>`. It will figure out the position respecting the PDB convention of residue numbers being ordered first by residue number, then by insertion code. You can insert in the middle of the chain, or at either terminus.

6.5 Making a deletion mutation

It's even simpler to delete a residue. The MMB syntax for this is:
`deleteResidue <chain> <residue ID>`
MMB will simply delete the residue at position `<residue ID>`. You can also delete an entire range, like this:
`deleteResidue <chain> <start residue ID> <end residue ID>`
.. and MMB delete all residues in the specified range. Lastly, you can delete an entire chain, like this:
`deleteResidue <chain>`

6.6 Renumbering residues

The PDB has been referred to as a bioinformatician's nightmare. There are several reasons for this, one of which is that structural biologists can be rather liberal in their interpretation of the residue numbering rules, in particular what insertion codes mean and in what order they should go. There is also the problem of gaps. Some software packages, such as FoldX, don't handle insertion codes very well. So there are many reasons why you may wish to change the numbers so they start at some integer value and increase consecutively from there. To do this, simply issue:

```
renumberBiopolymerResidues
```

MMB will simply renumber all biopolymer chains to start at 1. Note that any commands issued in the same stage, that access residue numbers, implicitly or explicitly, will cause MMB to crash. Sorry. So if you need to renumber, do it in a stage that involves nothing else. You can do any modeling you wish before and after that stage. Here is an example of me renumbering 1A22.pdb:

```
firstStage 2
lastStage  3
readAtStage 2
    loadSequencesFromPdb 1A22.pdb
```

editing

```
    renumberBiopolymerResidues
    reportingInterval .0000000001
    numReportingIntervals 1
readBlockEnd
readAtStage 3
    # last.2.pdb will have renumbered residues, which will now
be read by:
    loadSequencesFromPdb
    # You can issue any commands you want here. Just use the
new residue numbers.
readBlockEnd
```

An alternative syntax involves calling `loadSequencesFromPdbAndRenumber`. This is just like `loadSequencesFromPdb`, but it renumbers right after the coordinate matching step. Unfortunately it is also pretty much incompatible with any residue-specific operations. Example usage is only slightly different:

```
firstStage 2
lastStage  3
readAtStage 2
    loadSequencesFromPdbAndRenumber 1A22.pdb
    reportingInterval .0000000001
    numReportingIntervals 1
readBlockEnd
readAtStage 3
    # last.2.pdb will have renumbered residues, which will now
be read by:
    loadSequencesFromPdb
    # You can issue any commands you want here. Just use the
new residue numbers.
readBlockEnd
```

7 Building arbitrary molecules (beta)

In this chapter we talk about building *arbitrary* molecules. Here MMB will turn user commands into molmodel commands. This feature is very much in beta, so you will need help if you are going to do anything complicated. There exist both known and (almost certainly) unknown bugs. This capability is in the trunk, but may not be in your distribution. Contact me if you are using this!

7.1 Example: benzene

This is the approach which is most general, building up the molecule one atom at a time:

```
# chain ID and arbitrary 3-character residue name:
molecule initialize B RNG
# create a trivalent atom, of element Carbon, name it C1:
molecule B setBaseAtom   TrivalentAtom C1 Carbon
# The TrivalentAtom has bonds named bond1, bond2, and bond3.
Attach the next atom to one of these three (here we chose
bond2) and specify the bond length (.14 nm):
molecule B bondAtom TrivalentAtom C2 Carbon C1/bond2 .14
# note that in the above, bond2 of C1 is now occupied.  It's
also implicit that bond1 of C2 is occupied, because that's by
default the bond on the child which is used for attachment.
# now all the other four carbons in the ring:
molecule B bondAtom TrivalentAtom C3 Carbon C2/bond2 .14
molecule B bondAtom TrivalentAtom C4 Carbon C3/bond2 .14
molecule B bondAtom TrivalentAtom C5 Carbon C4/bond2 .14
molecule B bondAtom TrivalentAtom C6 Carbon C5/bond2 .14
# Use the special atom type AliphaticHydrogen here, and use
another available bond.
molecule B bondAtom AliphaticHydrogen H1 Hydrogen C1/bond3 .1
molecule B bondAtom AliphaticHydrogen H2 Hydrogen C2/bond3 .1
molecule B bondAtom AliphaticHydrogen H3 Hydrogen C3/bond3 .1
molecule B bondAtom AliphaticHydrogen H4 Hydrogen C4/bond3 .1
molecule B bondAtom AliphaticHydrogen H5 Hydrogen C5/bond3 .1
molecule B bondAtom AliphaticHydrogen H6 Hydrogen C6/bond3 .1
# Now for atoms C2,C3,C4,and C5, we have now occupied bond1
(to attach to the preceding atom), bond3 (to attach a
hydrogen), and bond2 (to attach the succeeding atom). So no
```

(beta)

```
bonds left on those atoms.
# C1 has bond1 still available (because it has no parent
atom).  C6 has C2 available, because it has no child atom.

#Find convenient biotypes in your tinker parameter file for
these atoms.  In this case we used Phenylalanine CZ, with
Ordinality:Any for the carbons, and Phenylalanine HZ for the
aliphatic hydrogens:
molecule B setBiotypeIndex C1 Phenylalanine CZ Any
molecule B setBiotypeIndex C2 Phenylalanine CZ Any
molecule B setBiotypeIndex C3 Phenylalanine CZ Any
molecule B setBiotypeIndex C4 Phenylalanine CZ Any
molecule B setBiotypeIndex C5 Phenylalanine CZ Any
molecule B setBiotypeIndex C6 Phenylalanine CZ Any

molecule B setBiotypeIndex H1 Phenylalanine HZ Any
molecule B setBiotypeIndex H2 Phenylalanine HZ Any
molecule B setBiotypeIndex H3 Phenylalanine HZ Any
molecule B setBiotypeIndex H4 Phenylalanine HZ Any
molecule B setBiotypeIndex H5 Phenylalanine HZ Any
molecule B setBiotypeIndex H6 Phenylalanine HZ Any

# Now we add a bond to close the ring.  Recall the bond
centers that are available on C1 and C6:
addRingClosingBond B C1 bond1 C6 bond2
```

7.2 Example: ethane

Here is an example of how to build ethane:

```
molecule initialize B MTN
molecule B setBaseCompound MethylGroup
molecule B convertInboardBondCenterToOutboard
#molecule B bondAtom AliphaticHydrogen H4 Hydrogen methyl/bond
0.1112
molecule B bondCompound methyl2 MethylGroup MethylGroup/bond
molecule B setBiotypeIndex C MethaneC
molecule B setBiotypeIndex H1 MethaneH
molecule B setBiotypeIndex H2 MethaneH
molecule B setBiotypeIndex H3 MethaneH
molecule B setBiotypeIndex methyl2/C MethaneC
molecule B setBiotypeIndex methyl2/H1 MethaneH
molecule B setBiotypeIndex methyl2/H2 MethaneH
molecule B setBiotypeIndex methyl2/H3 MethaneH
#molecule B setBiotypeIndex H4 MethaneH
molecule B defineAndSetChargedAtomType MethaneC 1 -0.18
molecule B defineAndSetChargedAtomType MethaneH 34 0.06
```

7.3 Example: GDP

Here is an example of how to build GDP:

```
molecule initialize H   G
molecule H setBaseCompound  Guanylate
# Add alpha phosphate :
molecule H bondAtom QuadrivalentAtom PA Phosphorus
O5*/bond1 .14
molecule H setBiotypeIndex PA Phosphate,?RNA P Initial
molecule H bondAtom UnivalentAtom  OPA1 Oxygen PA/bond3
0.14800
molecule H bondAtom UnivalentAtom  OPA2 Oxygen PA/bond4
0.14800
molecule H setBiotypeIndex OPA1 Phosphate,?RNA OP Initial
molecule H setBiotypeIndex OPA2 Phosphate,?RNA OP Initial
molecule H bondAtom  BivalentAtom OPA3 Oxygen PA/bond2 .140
2.094
molecule H setBiotypeIndex OPA3 Phosphate,?RNA O5* Initial
molecule H bondAtom QuadrivalentAtom P Phosphorus
OPA3/bond2  .14
molecule H bondAtom UnivalentAtom  OP1 Oxygen P/bond3 0.14800
molecule H bondAtom UnivalentAtom  OP2 Oxygen P/bond4 0.14800
molecule H bondAtom UnivalentAtom  OP3 Oxygen P/bond2 0.14800
molecule H setBiotypeIndex P Phosphate,?RNA P Initial
molecule H setBiotypeIndex OP1 Phosphate,?RNA OP Initial
molecule H setBiotypeIndex OP2 Phosphate,?RNA OP Initial
molecule H setBiotypeIndex OP3 Phosphate,?RNA OP Initial
```

7.4 Example: water

I apologize in advance about how hard it is to make a simple water molecule!

```
molecule initialize H H2O
molecule H setBaseAtom BivalentAtom OW1 Oxygen
molecule H bondAtom UnivalentAtom HW1 Hydrogen OW1/bond1 .14
molecule H bondAtom UnivalentAtom HW2 Hydrogen OW1/bond2 .14 0
Free
molecule H defineBiotype O 2 TIP3P Oxygen
molecule H defineBiotype H 1 TIP3P Hydrogen
molecule H setBiotypeIndex OW1  TIP3P Oxygen  Any
molecule H setBiotypeIndex HW1  TIP3P Hydrogen  Any
```

parameters

```
molecule H setBiotypeIndex HW2  TIP3P Hydrogen  Any
molecule H defineAndSetChargedAtomType  TIP3P Oxygen Any 21  -
0.834
molecule H defineAndSetChargedAtomType  TIP3P Hydrogen Any 30
0.417
molecule H setDefaultBondAngle 104.52 HW1 OW1 HW2
```

8 Global parameters

This appendix, describes global parameters available to users. It does not cover *commands* such as `baseInteraction, aromatic, contact, mobilizer,` and `constraint.` The simplest difference between a *parameter* and a *command* is the following. A *command* can be issued an unbounded number of times, subject only to memory and computer time limitations. The major caveat is that in the case of `constraint` commands, one must not overconstrain the system. In contrast a *parameter* can only be set once (at least for a given stage); if a parameter is set multiple times for a given stage, only the last value of that parameter will be used. A listing of all user-configurable global parameters and their current values is printed at the beginning of every stage of an MMB run. Some additional parameters are available but rarely used or not recommended; contact the author with questions on these.

This chapter does not describe *staged* parameters. These are parameters for which not only the *value,* but also the *stage* at which they first take effect is specified, for example `temperature` and `dutyCycle`.

addAllAtomSterics	Bool	FALSE	Add steric contact spheres to all atoms. This is more expensive and more prone to kinetic trapping than addSelectedAtoms.
addAllHeavyAtomSterics	Bool	FALSE	Add steric contact spheres to all atoms EXCEPT hydrogens.
checkSatisfied	Bool	FALSE	At each reporting interval, list all the baseInteraction's and determine which were satisfied.
constraintTolerance	float	0.05	This determines the tolerance of the Weld constraint. If Weld'ed pieces are moving relative to each other, reduce this number.
cutoffRadius	float	0.1	This is the range of the MMB potential. See our Multiple-template homology modeling paper.
densityFileName	String		Name of file for fitting based on electron density, in .xplor format. If you need to convert from some other format, we recommend using mapman (e.g. rave_osx for mac). Instructions are here: http://xray.bmc.uu.se/usf/mapman_man.html#S10
densityForceConstant	Float	1	Scale factor for the density based forces
firstStage	int	1	Stage at which simulation should begin.
globalAmberImproperTorsionScaleFactor	float	0	

globalBondBendScaleFactor	float	1.0	These eight parameters set scaling factors for terms in the Amber99 potential. Most default to 0 for economy.
globalBondStretchScaleFactor	float	1.0	
globalBondTorsionScaleFactor	float	0	
globalCoulombScaleFactor	float	0	
globalGbsaScaleFactor	float	0	
globalVdwScaleFactor	float	0	
initialSeparation	float	20.0	Sets the separation between chains at stage 1, or whenever readPreviousFrameFile = false.
integratorAccuracy	int	0.001	Integrator tolerance, applies for variable step size time integrators.
integratorStepSize	int	0.001	Step size in ps, for fixed step size integrators.
integratorType	string	Verlet	Choose between Verlet, RungeKuttaMerson
integratorUseFixedStepSize	Bool	FALSE	self explanatory
lastStage	int	1	Stage at which simulation will end
leontisWesthofInFileName	string	./parameters.csv	MMB parameter file
loadTinkerParameterFile	Bool	FALSE	If FALSE, uses hard-wired Tinker parameters. If 1, reads parameters from tinkerParameterFileName
numReportingIntervals *alias* maxReportingIntervals	int	100	Number of reporting intervals per stage.
nastGlobalBondTorsionScaleFactor	int	10	Scale factor for NAST torsional potential
physicsRadius	Float	0	If this is set to a value > 0, all residues within physicsRadius of any “flexible” atoms will be added to the physics zone. “flexible” atoms are defined as those belonging to a mobilized body of mass < 40.
randomizeInitialVelocities	Bool	FALSE	Adds a random velocity to each body at the beginning of the simulation stage. Note that if you are have any non-interacting bodies (e.g. free ions with charges turned off) you may wish to apply initial velocities, otherwise the Nose-Hoover thermostat will leave them in their zero kinetic energy state.
reportingInterval	float	1.0	Duration of reporting intervals, in ps.
removeRigidBodyMomentum	Bool	FALSE	When True, periodically sets overall translational and rotational momentum to zero.
rigidifyFormedHelices	Bool	FALSE	
scrubberPeriod	float	4	Duration of one cycle of potential rescaling (ON time + OFF time) in ps.
safeParameters	Bool	TRUE	When TRUE, checks for syntax errors as well as some potentially dangerous parameter values.

setForceAndStericScrubber	Bool	FALSE	No longer user configurable. When dutyCycle < 1.0, this is automatically set to TRUE. It turns ALL forces (including baseInteraction's, sterics, Amber99 force field, springToGround's, etc.) off for (dutyCycle -1) fraction of each scrubberPeriod.
setHelicalStacking	Bool	TRUE	if TRUE, identifies three consecutive WatsonCrick/WatsonCrick/Cis base pairs as a helix and applies HelicalStackingA3/HelicalStackingA5/Cis baseInteraction's between the consecutive residues on each strand.
setTemperature	Bool	TRUE	Turns on thermostat.
thermostatType	string		Choices are NoseHoover and VelocityRescaling
tinkerParameterFileName	string		Name of the tinker-formatted parameter file. Only needed if the tinker force field is turned on.
baseInteractionForceMultiplier *alias* twoTransformForceMultiplier *alias* forceMultiplier	float	100	Scale factor applied to all baseInteraction and aromatic forces. 100 or 1000 is recommended to speed up modeling.
useFixedStepSize	Bool	FALSE	Specifies fixed-step-size time integration.

9 Macros

This appendix describes macros available to users. These macros set parameters on the user's behalf. These are provided in cases where the corresponding commands might be confusing to the user, or simply not under user control.

matchFast	This sets matchExact TRUE, matchIdealized FALSE, matchOptimize FALSE, and guessCoordinates FALSE. It is very economical. It is the default behavior, so usually there is no need to call this.
matchGapped	This sets matchExact TRUE, matchIdealized TRUE, matchOptimize TRUE, and guessCoordinates TRUE. It guesses positions for any missing atoms. matchFast does this just fine for side chains, but matchGapped can handle missing backbone atoms. There may be unphysical bond geometries at the boundaries between known and missing backbone atoms, but there are ways to heal this.
setDefaultMDParameters	Equivalent to issuing: globalBondTorsionScaleFactor 1.0 globalAmberImproperTorsionScaleFactor 1.0 globalBondBendScaleFactor 1.0 globalBondStretchScaleFactor 1.0 globalBondTorsionScaleFactor 1.0 globalCoulombScaleFactor 1.0 globalVdwScaleFactor 1.0 globalAmberImproperTorsionScaleFactor 1.0

10 User defined variables, parameter arithmetic, and conditional blocks

In this Appendix, we describe how to define numerical variables, and various ways to specify sections of the input file which are to be read or ignored at certain stages.

10.1 Comment marker

The comment marker is `#`, e.g.:

```
# Don't read this, it's just a comment
```

10.2 User defined variables

User variables are defined with the following syntax:

```
@<variable-name> <float or integer value>
```

The variable `@<variable-name>` can then be used wherever a literal integer or float is expected. If a float is assigned to the variable, and the variable is later used where an integer is expected, MMB will return an error. The definition of the variable should precede its first use in the input file. For example:

```
#declare @myStage variable and set to 3
@myIntervals 3
# now use it where a number (in this case an integer) is
expected:
numReportingIntervals @myIntervals
```

Don't use any punctuation or whitespace in `<variable-name>`.
Don't try to set `firstStage` or `lastStage` with a user variable.

10.3 Parameter arithmetic

User variables are pretty handy, and start to make the command file more like a programming language. In the same vein, MMB allows the '`+`' and '`-`' operators. This means that any integer or floating-point (double-precision) parameter value can be set using a combination of literals, user variables, and the above operators. There is no limit to the number of operators and operands. Here are a couple of examples:

```
@DUMMY 40
```

```
numReportingIntervals  @DUMMY+10-@DUMMY
@MYFLOAT 0.35
reportingInterval  4+@MYFLOAT-0
```

This is equivalent, of course, to:

```
numReportingIntervals  10
reportingInterval  4.35
```

Don't use any whitespace or additional punctuation (such as parentheses, commas, etc.) in an arithmetic expression. Note also that residue ID's are special (they're not integers), and their '+' operator follows different rules (see Chapter 2).

10.4 Conditional blocks

In many cases we will want to issue different commands and make different choices of parameter values at different stages of a job. For this purpose we can enclose a block of the input file in a conditional block, which is opened as follows:

`readFromStage <stage-number>` Read only if the current stage is equal to or GREATER than `<stage-number>`.

`readToStage` Read only if the current stage is equal to or LESS than `<stage-number>`.

`readAtStage` Read only if the current stage is EQUAL to `<stage-number>`.

`readExceptAtStage` Read only if the current stage is NOT EQUAL to `<stage-number>`.

The commands and parameters to be conditionally read follow, and the end of the block is indicated with a `readBlockEnd` statement, e.g.:

```
# start conditional block:
readAtStage 3
# read the following lines only at stage 3:
sequence C CCUAAGGCAAACGCUAUGG
firstResidueNumber C 146
baseInteraction A 2658 WatsonCrick A 2663 WatsonCrick Cis
contact C 146 SelectedAtoms C 164
# end conditional block:
readBlockEnd
# continue with the rest of the input file
```