AMBER "Lite": Some AMBER-Tools-Based Utilities

Romain M. WOLF

Novartis Institutes for Biomedical Research, NIBR, Basle, Switzerland

December 16, 2010

Abstract

AMBER "Lite" is a small set of utilities making use of the free AMBER Tools package (currently for version 1.4 or higher). The main focus is on the preparation of files for MM(GB)(PB)/SA-type simulations. The utilities can be used as delivered or they can serve as a starting point for further development. Examples are included to illustrate the concepts or to test the correct functioning of the installation. The text also contains a (very) condensed introduction to some AMBER file preparation concepts. For more precise and detailed information, users should read the original AMBER manuals (available for free from the home page of AMBER (*http://ambermd.org/*) and scan through the excellent tutorials prepared over the years by a number of people.

Contents

1 License and Feedback

The AMBER Lite package (© Novartis Institutes for Biomedical Research, Basel, Switzerland) is free software under the GNU General Public License (GPL), as are the parts on which the package builds, namely the Amber tools *ptraj*, *leap*, *antechamber*, *sqm*, *pbsa* and the *NAB package*, all available under GPL license from the official AMBER site *http://ambermd.org/*.

Users are free to modify the tools according to their needs. Strange or obviously wrong behavior should be communicated to the author (at *romain.wolf@novartis.com* or *romain.wolf@gmail.com*). Feedback (positive or negative) is welcome although I cannot guarantee continuous support. I will do my best to answer questions, correct bugs, or add features if they seem useful and if my time allows it.

2 Introduction

For many standard simulation tasks, only a limited number of tools within the AMBER package are required. Furthermore, the full set of routines can be confusing for new or casual users. The constantly enhanced and updated AMBER tutorials certainly offer an excellent entry point. The set of tools described hereafter should present another initiation, based entirely on the freely available portions of AMBER code. The emphasis in the AMBER Lite tools is on the MM(GB)(PB)/SA approach to compute (relative) free energies of interaction between ligands and receptors, a major task in structure-based drug discovery. The tools are simple enough to be understood, modified, and enhanced.

One section (Appendix A.1) is dedicated to the preparation of PDB files prior to use them with AMBER. In my own experience, this is a critical part in setting up simulations. Scanning through the AMBER Mail Reflector, I find many reported problems and questions which originate from ”bad” or badly prepared PDB starting files.

Another section (Appendix B) gives a brief introduction on AMBER ”masks” and NAB ”atom expressions”, used to select parts of molecular structures. Users should also read in detail the corresponding information in original AMBER documentations. Wrong partial selections are tricky because they may often go unnoticed, i.e., everything seems to run OK but the results are totally flawed.

2.1 Installation

Python (version 2.4 or newer) and a **C-compiler** for generating the binary executables of NAB-based applications must be available.

The **AMBER Tools package (version 1.3 or better 1.4)** must be installed and the environment variable pointing to its main directory (*$AMBERHOME*) must be set correctly. The `$AMBERHOME/bin` subdirectory must be in the executables path ($PATH). If the *AMBER Tools* installation passes the tests that are delivered with that package, the utilities described in this document should also work.

The Python scripts do not require special packages or modules other than those included in (most?) standard Python distributions. They have only been tested on UNIX-like systems like Linux and Mac OSX, but not under MS-Windows.

The *AMBER Lite* distribution has the following file structure:

`amberlite/` is the root folder;

- `../python` contains the Python scripts;
- `../src` contains the NAB source files (extension ".nab");
- `../bin` should eventually contain all binary NAB applications and also soft links to the Python scripts in the `../python` subfolder so that only this single folder has to be added to the global $PATH variable;
- `../doc` is for documentation and contains this manual and the GPL license text;
- `../examples` and its subfolders contain the files used as examples in Appendix C of this manual.

The simplest first-time installation procedure is to expand the file *amberlite.tgz*, go to the generated`amberlite` directory, and execute the `install.py` script. The script will check that the AMBERHOME environment variable is set and that all required AMBER Tools executables are found in the path. It will then create a '`bin`' subdirectory, compile the NAB sources, put the resulting binaries into the `bin` subdirectory and also make symbolic links to the python scripts in the same directory.

You must finally add the resulting `bin` directory to your PATH environment variable.

2.2 Python Scripts

The following Python scripts are currently included:

pytleap prepares AMBER parameter-topology (PRM) files, AMBER coordinates (CRD) files and corresponding PDB files for proteins, organic ligands (or peptides), and receptor/ligand complexes, using as input PDB files (for proteins and peptides) or SDF files for organic molecules. It is basically a wrapper around *tleap* and *antechamber*.

pymdpbsa is a full analysis tool for MD(GB,PB)/SA computations, given an MD trajectory (or a single PDB file) of a receptor-ligand complex and the individual PRM files for the complex, the receptor, and the ligand.

The Python scripts take **command line options** many of which assume default values. If the default values apply, these options can be omitted. Most options are of the form `--option` ***`value`*** where ***`value`*** can be a filename, an integer, a float, or a special string (to be included in quotes). Typing just the executable name or followed by `--help` lists the options and exits.

Common errors, like e.g. missing files, are captured by the scripts which also always check that the AMBERHOME environment variable is set and that all required binary executables are available and in the execution path.

The *pymdpbsa* script creates a **temporary subdirectory** of the current working directory. Computations are executed in this temporary folder and all output is stored there also. When finished, the resulting data are copied back to the starting directory. By default, the temporary directory is **not** removed. The user can explicitly request its automatic removal via the `--clean` option. Alternatively, it can be removed manually later. Temporary directories have names which make them easy to identify and all have the extension `.tmpdir` (see details later).

2.3 NAB Applications

The NAB applications are written in NAB language,[1] which is "C" with numerous additional functions specific to computational chemistry problems. NAB works as a precompiler, generating C-code from the NAB source which is then processed through the default C-compiler. NAB functionality has much in common with the "big" AMBER modules, but there are also some notable differences:

The NAB applications cannot handle explicit solvent and periodic boundaries but work only with implicit solvation models. The possibilities to use restraints on atoms are also more limited and use a notation different from the AMBER 'mask' scheme (explained later). Otherwise, they deliver results which are fully compatible with original AMBER simulations under identical conditions.

The NAB applications presented here use the same parameter-topology files as AMBER modules like, e.g., *sander*, but they read coordinates (initial atom positions) from PDB files and not from AMBER-specific coordinate (CRD) files. The only output format for MD trajectories is the "binpos" format which can be read by various other packages or can also be converted to other formats via the *ptraj* utility included in AMBER Tools.

[1] NAB was created David Case's group (Tom Macke, W.A. Svrcek-Seiler, Russell A. Brown, Istvan Kolossvary, Yannick Bomble and David A. Case) at the Scripps Research Institute (La Jolla, CA, USA) and maintenance is continued at his current location at Rutgers University, NJ, USA.

The following NAB-based tools are currently included:

ffgbsa returns the AMBER energy (MM + GB polar solvation + "non-polar" solvent-accessible surface term) of a system, given its PRM and PDB file.

minab is a crude conjugate-gradient minimizer using PRM and PDB files as input and generating a PDB file with the refined coordinates.

mdnab is a molecular dynamics routine with a minimum of user-specified options which takes PRM and PDB files as input and writes out the MD trajectory in the *"binpos"* format.

These NAB applications are single-line commands taking a number of arguments (which makes it easy to incorporate them into other scripts). In contrast to the Python scripts, they do not use the (more) convenient `--option` scheme, but **require the command line arguments in the correct order**. Entering just the name of the application without arguments lists a help which shows and explains the arguments to be used.

There is no extensive exception handling in the NAB applications. User errors are punished by simple crashes of the applications!

Users who want to modify NAB applications must edit the source, re-compile it into a NAB binary (using the command `nab` *`source.nab`* `-o` *`binary_name`*), and then copy the binary into a directory of their executable path.

3 Coordinates and Parameter-Topology Files

Simulations with AMBER modules require defined data and control files. The error-free generation of these files is often a discouraging hurdle for beginners or users who do not use AMBER regularly.

At least two data file types are required: a **coordinates (CRD)** file for AMBER modules (or **PDB** files for NAB applications) with atom positions and a **parameter-topology (PRM) file** containing all force field data required for the system. The two file types **must** have the **same number** of atoms and all atoms in the **same sequential order**. Not respecting this fundamental rule leads to severe flaws. The separation of coordinates and topology has the advantage that the same topology file can be used for various different starting coordinates. However, any change in the coordinate file that alters also the number of atoms or even their sequential order is not allowed. This is a frequent source of error and re-using PRM files created some time in the past under not well documented conditions is strongly discouraged.

The current tool delivered with AMBER to prepare coordinate (CRD or PDB) and parameter-topology (PRM) files is called ***leap*** (*tleap* for the terminal variant and *xleap* for the graphics variant). There is also a more recent *sleap* program which is supposed to replace *tleap*. The current AMBER Lite scripts still use the original (antique?) *tleap* routine because it is well tested and robust. The new *sleap* will be used in the next AMBER Lite version.

Since *leap* is not particularly user-friendly,[2] a Python script ***pytleap*** (see section 4) has been prepared which runs the terminal version of *leap* in the background and does not require a direct interaction with *leap* itself, at least for simple tasks like preparing a protein or a receptor-ligand complex for simulations with implicit solvent.

For small organic molecules, *pytleap* first invokes ***antechamber***[1] before passing them through *leap*, allowing the usage of the *gaff* force field[2] for organics without directly interfering with *antechamber* itself.

The Appendix A (page 24) gives a short outline of the most important preparation steps required on the raw data (mostly PDB files) before using any AMBER-related tools. Those recipes may not be the most elegant ones but they work in most cases and help avoid common problems.

[2] ...a common feature for routines which perform lots of different and fairly complex tasks...

4 *pytleap*: Creating Coordinates and Parameter-Topology Files

pytleap calls the *tleap* and/or *antechamber* utilities in the background. It is invoked by a single command line with a set of options and eventually creates the files required for an AMBER simulation, starting from a PDB file (protein) and/or an SDF file (ligand). The script is especially useful to set up receptor-ligand complexes for simulations using MM(GB)(PB)/SA and related techniques, but can also be used for isolated proteins or ligands.

Proteins (or peptides) are read as PDB files in *pytleap*. Other formats are not supported. Be sure to have a "clean" PDB file as described in Appendix A.

The SDF format for small organic ligands is chosen for reasons of compatibility. SDF files can be written by most standard molecular modeling packages and contain the information required by the *antechamber* package to generate the files for AMBER simulations. The format is simple and includes the connectivity with bond orders. Note that the SDF file of the ligand **must** have **all hydrogens** included. Also, the formal charge on the ligand (if any) is **not** read from the SDF file but **must** be explicitly specified (see later). For charge calculations, we use the *sqm* semi-empirical QM routine from AMBER Tools instead of MOPAC. After some tests, we have opted for less severe gradient requests then those used by default in *antechamber* to speed up the partial charges generation for ligands: `grms_tol` is set to 0.05. We include the peptide bond correction by setting `peptide_corr=1`.

To generate AMBER files for a **protein-ligand complex**, prepare the protein in PDB and the low-molecular-weight ligand in SDF, i.e., save both components in distinct files (and make sure that the protein PDB file does not contain the ligand anymore). In the case of protein-peptide (or protein-protein) complexes, you must also separate the two entities, in this case into distinct PDB files, since individual parameter-topology files have to be generated for the complex and for each component separately if MM(GB)(PB)SA computations are envisaged later.

Obviously, the geometry of the entire complex must be reflected in the coordinates of the respective files. *pytleap* will only combine the protein and the ligand into a single structure, assuming that the ligand fits the target in a desired way. It will of course not "dock"!

4.1 Running *pytleap*

Note: Since *pytleap* and the modules called by it read or write temporary files with defined names, it is wise to **run one single instance of *pytleap* in a directory**. Not respecting this rule will lead to confusion and errors!

Typing `pytleap` without any arguments (or followed by `--help`) results in the following output:

```
--------------------------------------------
 pytleap version 1.2 (December 2010)
--------------------------------------------
Usage: pytleap [options]

Options:
  -h, --help         show this help message and exit
  --prot=FILE        protein PDB file                       (no default)
  --pep=FILE         peptide PDB file                       (no default)
  --lig=FILE         ligand MDL (SDF) file                  (no default)
  --cplx=FILE        name for complex files                 (no default)
  --ppi=FILE         name for protein-peptide complex files (no default)
  --chrg=INTEGER     formal charge on ligand                (default: 0)
  --rad=STRING       radius type for GB                     (default: mbondi)
  --disul=FILE       file with S-S definitions in protein   (no default)
  --sspep=FILE       file with S-S definitions in peptide   (no default)
  --pfrc=STRING      protein (peptide) force field          (default: ff03.r1)
  --lfrc=STRING      ligand force field                     (default: gaff)
  --ctrl=FILE        leap command file name                 (default: leap.cmd)
```

The command line options are presented here below:

`--prot` *`filename`* uses the PDB file *filename* as the protein structure. The PDB file must be ”clean”, according to the rules outlined in the Appendix A.1. The *leap* module adds hydrogens with correct names (and also missing heavy atoms, if any), attributes the correct partial charges and AMBER atom types,[3] and eventually writes out the files for the protein as mentioned in section 4.2.

`--pep` *`filename`*reads a (clean) PDB file *filename* as the peptide structure. There is no difference to the `--prot` option except that a second (separate) peptide (or protein) can be read in and combined later with the structure read via `--prot` to a protein-peptide (or protein-protein) complex (see `--ppi` below).

`--lig` *`filename`* uses the SDF file *filename* as the ligand structure. The ligand file **must include all hydrogens**. The structure is processed through *antechamber* that generates various files required by *tleap* to build the PRM file for the ligand. Inside *antechamber*, the ligand becomes a molecule (residue) with the name ”**LIG**”. This name is then taken over by *leap* and appears as such in the resulting PRM and PDB files. The name ”LIG” is the default name for a ligand in the *pymdpbsa* (section 8). See also option `--chrg` when using the `--lig` option. **Note:** We assume here that a ligand is a **single-residue** low-molecular-weight organic molecule.

[3] Charges and atom types will correspond the chosen force field parameter set and the libraries going with them.

`--cplx` ***filename*** **(no extension!)** will generate the AMBER files PRM, CRD and a PDB file of the complex of the protein and the ligand read in with the `--prot` and `--lig` options. When generating AMBER files for the complex, the files for the individual protein and ligand are always generated also. They are useful when running MM(GB)(PB)/SA computations later (section 8). **This option only makes sense when both the `--prot` and `--lig` options are also chosen.**

`--ppi` ***filename*** works the same as `--cplx`, except that it generates a complex between two units supposed to be clean proteins (peptides), not requiring any intervention of *antechamber*. Furthermore, `--cplx` and `--ppi` cannot be used in the same run, i.e., we can only deal with either a protein/organic-ligand complex or a protein/protein (or protein/peptide) complex.

`--chrg` ***integer*** must be used if an organic ligand read from an SDF file is formally charged (even if the charge is also given in the SDF file). For neutral ligands, this option can be omitted. For charged ligands however, it is required! Enter it as an integer reflecting the correct total charge of the ligand. The computation of partial charges via the AM1-BCC method[3,4] will fail if the formal charge on the ligand does not make sense with the chemical structure including all hydrogens and *pytleap* will quit.

`--rad` ***radius_type*** is used to choose the atomic radii for Generalized-Born. The default radius type is the "*modified Bondi*" option to be used with the GB option `gb` set to 1. For `gb` = 2 or 5, the original AMBER documentation suggests the radius type `mbondi2`.

`--disul` and `--sspep` ***filename*** are used to generate disulfide bonds. Disulfide bridges must be prepared in the original PDB file by renaming the involved cysteine residues from CYS to CYX (see A.2.2). The *filename* in this option must relate to a file that contains pairwise integer numbers of cysteine residue names to be connected (one pair per line!). These numbers must correspond to the ones in the original PDB file![4] See the example in section C.1. We consider that this explicit formation of disulfide bonds is to be preferred over "automatic" S-S bond formation, be it by using Sγ distances or by relying on `CONECT` records in PDB files. **NOTE: `--disul` applies to the file read in via `--prot` while `--sspep` is applied to the molecule read in via –pep.** If both proteins (peptides) have disulfide bonds, you must use separate definition files for the respective S-S links!

`--sspep`: see above.

`--pfrc` *filename* specifies the force field parameter set for the protein. Since AMBER can use different force fields, this option allows to choose among them. The selection actually does not call the parameter file itself but a *leap* command file that initializes it. These special *leap* files all have a name *leaprc.xxxx* and are retrieved when the `AMBERHOME` environment variable is set correctly. **You must only specify the *xxxx* part of the name!** Thus, `ff99` selects the parm99 parameter set, while the **default** `ff03.r1` selects the latest parm03 force field with the correct charges for N- and C-terminal residues also. Make sure to have this file (with the full name *leaprc.ff03.r1* included in the directory where all default `leap` command files are kept.[5]

`--lfrc` *filename* selects the force field for the ligand. At this point, the default *gaff* force field is the only reasonable choice in most cases and you can omit this (default)

[4] In 'weird' PDB files where insertions and deletions get special names, trying to keep a 'standard' numbering of residues for the main protein of a family, much can go wrong. In these cases, it is best to renumber the residues sequentially in the PDB file before referring to residue numbers.

[5] The full path to this place is `$AMBERHOME/dat/leap/cmd`.

option.

`--ctrl` ***`filename`*** can be used to change the default name of the leap command file generated by *pytleap* (default `leap.cmd`). In general, this is not necessary, except if you would like to keep this particular file and protect it from being overwritten by the default name the next time you use *pytleap* in the same directory.

NOTE: This version of *pytleap* does not offer the possibility to add solvent and counter-ions. It would be straightforward to add these options to the script if you are familiar with *leap*. Alternatively, you could use the *leap.cmd* (or alike) created by *pytleap*, edit it with a standard editor to add specific *leap* commands, and then resource it through *tleap* (e.g., with `tleap -f leap.cmd`).

4.2 Output from *pytleap*

Output from *pytleap* varies with the chosen command line options (see 4.1). Coordinate (CRD) files, parameter-topology (PRM) files and a corresponding PDB file are always generated. Hydrogen names in the output PDB files are "wrapped", making these files readable also by elder software packages which require this format. Note that the actual atom names in the PRM file are unwrapped. This has no consequence on computations. However, special residue names like HIE, HID, HIP, CYX, etc., are kept and may lead to flawed representations of the PDB files in software packages which do not recognize these residue names. The *ambpdb* routine included with AMBER Tools can be used to regenerate "standard" residue names if you need them.

Files generated by *pytleap* have a '`.leap.`' string in their name to identify them as "created by *leap*". **You should always use the corresponding *.leap.* files (or copies of them) for simulations!** This guarantees that the CRD, PDB, and PRM files are compatible, having the same number and sequence of atoms.

In addition, a file `leap.cmd` is left over. This is the file that was generated by *pytleap* and run through *leap*. The file `leap.out` is the output from *leap*, with the messages that would have been generated by running *leap* interactively. Finally, the `leap.log` file is the standard log from *leap*.

A special SYBYL MOL2 file is created when running *pytleap* on a ligand (i.e., a low-molecular-weight organic compound which is processed through *antechamber*). This file has the format of a generic MOL2 file, apart from the fact that atom types are not SYBYL but *gaff* atom types. The name of this file is `filename.ac.mol2`, with `.ac.` marking it as a file generated by *antechamber*.[6] The partial charges are those from the AM1-BCC method.[3,4]

Some additional files may be left over when *antechamber* is invoked. One **important file** is the `*.leap.frcmod` file containing additional parameters which are not in the original *gaff* parameter file. They are generated based on equivalences, "guessing", or empirical rules described the gaff paper.[2] The `frcmod` file can also be used as a quality check for the ligand parameters. Large *frcmod* files with many "guessed" parameters (especially for torsion angles) should be considered carefully.

Finally, the input and output files of the semi-empirical tool *sqm* are left. The ouput file (*sqm.out*) might be useful for debugging if the partial charges seem totally inadequate

[6] Opening this MOL2 file in a standard software that can read MOL2 files may lead to strange results because the gaff atom types do not reflect chemical elements as standard SYBYL MOL2 files with TRIPOS force field atom types.

despite the correct usage of the `--chrg` option (if required).

4.3 Error Checking

If you have experience with the *leap* application, look at the *leap.cmd* file that was created via *pytleap*. All the options that you have chosen should be represented as correct *leap* command lines. Furthermore, the *leap.output* and *leap.log* files should not show any errors, at best some warnings. If in doubt that the parameter-topology files have been correctly generated, look at these warnings and decide if they are benign. Eventually, the NAB application *ffgbsa* described below (section 5) can be used to run a single AMBER energy evaluation on the system. If the results returned by *ffgbsa* look very strange for a supposedly reasonable structure, you probably have a serious issue with your set of CRD, PDB, and PRM files.

5 Energy Checking Tool: *ffgbsa*

The NAB routine *ffgbsa* is an energy function called by the *pymdpbsa* application presented later (section 8). It can also be used as a standalone routine to check the AMBER energy of a molecular system and to test the correct working of a PDB/PRM file combination. It is invoked as:

```
ffgbsa pdb prm gbflag saflag
```

The order in the command line input is compulsory! `pdb` is the PDB file of the system and `prm` the related PRM file. `gbflag` is a flag to switch on one of the Generalized-Born (GB) options in AMBER and can be 1, 2, or 5.[7] Other values switch off GB and a simple distance-dielectric function $\epsilon = r_{ij}$ is used.

When `saflag` = 1, the solvent-accessible surface area (SASA) is also computed (via the *molsurf* routine included with NAB) and returned in Å^2, together with a SASA energy term which is simply $SASA * 0.0072$ in this case. The default cutoff for non-bonded interactions is 100 Å , i.e., virtually no cutoff for most systems. An example for the usage of *ffgbsa* is given in section C.2.

Remarks regarding the usage of *molsurf*:

The correct way to evaluate the SASA is to augment the radii of all atoms by the probe radius (usually 1.4 Å) and then run *molsurf* with a probe of radius of zero. This is also the implementation in *ffgbsa* here. The atom radii values are given in the following table:

Table 1: *Atom Radii Used in* molsurf

atom	radius (Å)	atom	radius (Å)
C	1.70	H	1.20
N	1.55	O	1.50
S	1.80	P	1.80
F	1.47	Cl	1.75
Br	1.85	I	1.98
any other	1.50		

Note: In some rare cases *molsurf* fails to give back a valid surface area. Scripts calling *ffgbsa* must be prepared to capture this. The *pymdpbsa* procedure described later catches such instances and excludes value sets in which the error occurs from the statistical analysis (cf. end of section 8.5.1).

[7] These values correspond to the "*igb*=" options in AMBER commands and stand for different implementations of the GB scheme.

6 Energy Minimizer: *minab*

The main purpose of this (very) simple minimizer is to refine a system prior to MD runs, mainly to remove potential hotspots which might destabilize the MD initiation. Using it for other purposes is at the discretion of the user.

The NAB routine *minab* uses the conjugate gradient minimizer of NAB to refine the energy of a system. To circumvent cutoff problems,[8] the cutoff for non-bonded interactions (vdW and Coulomb) is set to 100 Å and that for GB is fixed to 15 Å. The non-bonded list is not updated at all. The default for the gradient rms is set to 0.1.

For large systems, this is far from efficient. However, as stated above, the main purpose of this routine is to get rid of hotspots prior to running MD and in general, a few hundred iterations are sufficient to guarantee a decent structure for MD, especially when the MD starts with a heat-up phase as used in the *mdnab* application described in section 7.

The *minab* routine is invoked by:

```
minab pdb prm pdbout gbflag niter ['restraints' resforce]
```

Just typing `minab` without arguments gives a help screen. The explanation for the arguments follows:
`pdb` and `prm` are the PDB and **corresponding** PRM file of the system;
`pdbout` becomes the PDB file of the refined system;
`gbflag` is the GB flag which can be 1, 2, or 5 while any other value switches to distance-dependent dielectrics (as in section 5);
`niter` is the maximum number of iterations;

and for the optional arguments:

`restraints` specifies residues or atoms to be tethered in their motion (NAB atom expression **between quotes**);
`resforce` is a float specifying the restraint potential in kcal·mol^{-1}·Å^{-2}.

The `restraints` entry must be an atom expression according to the NAB rules outlined in B.2. If for example all Cα atoms should be restrained, this entry would be '::CA'. **If the restraint mask is given, the restraint potential `resforce` must also be specified.**

Since *minab* is a simple command-line tool, it can be called by other routines or scripts where a rough energy refinement is desired. The output (by default to the screen) can be captured for later analysis into a file via a simple redirect (">").

[8] The current cutoff scheme for non-bonded interactions in AMBER modules and NAB does not use a switching function to smooth the cutoff. This can lead to problems every time the non-bonded list is updated. Thus a fairly short cutoff distance with frequent list updates usually ends in line search problems before the required number of iterations or the requested rms of the components of the gradient is reached.

7 Molecular Dynamics "Lite": *mdnab*

The NAB application *mdnab* has been written for simple molecular dynamics with a minimum number of settings required by the user. Its main purpose is to run moderately short trajectories to be used e.g. for MM(GB)(PB)/SA applications.

Most settings are hardcoded and can only be changed by editing and re-compiling the source *mdnab.nab*.

The following (non-mutable) defaults are used:

The cutoff for non-bonded interactions and GB is always 12 Å. An update of the non-bonded list occurs every 25 steps. The integration step is 2 femtoseconds (using "rattle" to allow this fairly large step). The temperature is controlled via Langevin dynamics with a friction factor ("gamma_ln") of 2 for the production phase. The production temperature is fixed at 300 K. And *mdnab* **always saves one frame per picosecond**, independent of the length of the trajectory.

A heating and equilibration phase is automatically invoked prior to the actual production trajectory recording: 100 steps from 50 to 100 K, 300 steps from 100 to 150 K, 600 steps from 150 to 200 K, 1000 steps from 200 to 250 K, 3000 steps from 250 to 300 K, and an additional 10000 steps at 300 K.[9]

mdnab is started by

```
mdnab pdb prm traj gbflag picosecs ['restraints' resforce]
```

The command *mdnab* without arguments lists the possible arguments, the **sequence** of which is **compulsory**. The command line arguments are similar to those in *minab*:
`pdb` and `prm` are the PDB and corresponding PRM file of the system;
`traj` is the name for the production phase trajectory which will be saved in the binary "binpos" format (the extension `.binpos` is automatically attached);[10]
`gbflag` is the GB flag which can be 1, 2, or 5 (as in section 5), or anything else to switch off GB and use a distance-dependent dielectric function $\epsilon = r_{ij}$;
`picosecs` is the total number of picoseconds to run the production phase;

with the optional arguments:

`restraints` specifies atoms to be tethered in their motion (given as a NAB atom expression **between quotes**, see section B.2);
`resforce` is the restraint potential in kcal·mol^{-1}·Å^{-2} **which has to be given** if a restraint expression is specified.

While the trajectory is saved to the specified file name (the `traj` command line argument), the full output goes to the screen. To capture the output for later inspection, use the UNIX "redirect" (>) to a file and end the command line with a & (making *mdnab* a background job).

Note that only the production phase of the trajectory is recorded into the `traj` file. The heat-up phases are only documented in the general output (to the screen or to a text file, if redirected).

[9] Since these last 10000 steps at 300 K are run under identical conditions as subsequent the production phase, the user can simply extend the "equilibration" by discarding all frames from the production phase up to the point where the trajectory can be considered "stable" (noting that "stable" or "steady-state" are not well-defined terms anyway).

[10] This format can be read by various software packages like *VMD*, but can also be translated into other formats using the AMBER utility *ptraj*.

8 MM(GB)(PB)/SA Analysis Tool: *pymdpbsa*

8.1 Brief Overview on MM(GB)(PB)/SA Concepts

The original MM(GB)(PB)/SA procedure was developed in the late 1990's and the user should refer to some original papers on this subject.[5–8] The goal was to develop a relatively fast molecular-mechanics (-dynamics) based method to evaluate free energies of interactions. MM stands for Molecular Mechanics, PB for Poisson-Boltzmann, and SA for Surface Area.

The free energy for each species (ligand, receptor, or complex) is decomposed into a gas-phase energy (”enthalpy”), a solvation free energy and an entropy term, as shown in equation 1.

$$\begin{aligned} G &= E_{gas} + G_{solv} + S && (1)\\ &= E_{bat} + E_{vdW} + E_{coul} + G_{solv,polar} + G_{solv,nonpolar} + S_{rot,trans,vib} && (2) \end{aligned}$$

where E_{bat} is the sum of bond, angle, and torsion terms in the force field, E_{vdW} and E_{coul} are the van der Waals and Coulomb energy terms, $G_{solv,polar}$ is the polar contribution to the solvation free energy and $G_{solv,nonpolar}$ is the nonpolar solvation free energy.

The sum $E_{bat}+E_{vdW}+E_{coul}$ is the complete gas phase force field energy, the molecular mechanics (”MM”) part.

The polar solvation free energy $G_{solv,polar}$ can be evaluated via implicit solvation models like Poisson-Boltzmann (PB) or Generalized-Born (GB). The nonpolar contribution $G_{solv,nonpolar}$ is usually computed by a simple linear relation for a ”cavity” term

$$G_{solv,nonpolar} = \gamma \cdot SASA + const. \qquad (3)$$

where SASA is the solvent-accessible surface and γ has the dimension of surface-tension. Similarly, one could also use the volume enclosed by the SASA (SAV)

$$G_{solv,nonpolar} = p \cdot SAV + const. \qquad (4)$$

with p having the dimensions of pressure.

In a more sophisticated approach, $G_{solv,nonpolar}$ can be split into a repulsive (”cavity”) and an attractive (”dispersion”) term, as decribed in detail in the 2007 paper of Ray Luo and coworkers.[9]

The rotational and translation entropy loss about complexation (six degrees of freedom) is $3 \cdot kT$ (i.e., ~ 1.8 kcal·mol^{-1} at 300 K). The vibrational entropy can be evaluated, for example, via normal mode analysis. It has become common practice in recent work to exclude the entropy terms from MM(GB)(PB)/SA computations. This is acceptable when only relative free energies are computed to compare similar ligands in similar receptors. Furthermore, the entropy computation is the fuzziest part of the procedure and contributes to the largest fluctuations in the overall free energy when evaluating it over a number of MD frames.

The free energy of interaction in the complex can then be evaluated as:

$$\Delta G_{int} = G_{complex} - G_{receptor} - G_{ligand} \qquad (5)$$

In the early work, separate dynamics trajectories were recorded for all three species in explicit solvent. The solvent was then discarded, the free energy was evaluated according to the procedure above for a number of frames for each species. Eventually, ΔG was calculated by

$$\Delta G_{int} = \langle G_{complex} \rangle_{traj} - \langle G_{receptor} \rangle_{traj} - \langle G_{ligand} \rangle_{traj} \tag{6}$$

where $\langle G_i \rangle_{traj}$ is the average value for species i over all selected frames recorded during the production phase of the MD trajectory.

In the meantime, the method has been implemented and used in many variants, all of which have their advantages and disadvantages. The method presented hereafter is among the simplest and cheapest in terms of CPU power. It is based on a single trajectory of the complex alone. Each recorded frame is then split into receptor and ligand and equation 5 is applied to compute the interaction energy of the frame. The final interaction energy is then the average over the ΔG values of the selected frames. Also, the entropy is not evaluated at all.

8.2 Pitfalls and Error Sources

While the basic concepts are simple, there are many pitfalls. The initial idea was to compute values for the free energy of binding close to experimentally observed ones, without further tuning of parameters. However, since the computations of energy terms are based on force field parameters (internal energy, van der Waals interactions, and vibrational entropy via normal-mode analysis) and on concepts like atomic radii and partial charges (electrostatics and polar solvation terms), discussions on the quality of parameters are inevitable.

An issue not discussed in enough detail in many papers reporting MM(GB)(PB)/SA (and variants) is the quality of the MD trajectory. Unstable trajectories with unreasonably strong fluctuations or important transitions (conformational changes, ligand pose variations, etc.) will always yield questionable results. If such transitions happen, they must be checked carefully before the results are used for MM(GB)(PB)/SA.

In the "one-trajectory" approach implemented here, there is an additional pitfall. Since both the receptor and the ligand are only considered in the bound state, strain energy from distortions in the complex is not evaluated. This may not be an issue for the receptor if there are no strong induced-fit effects. For the ligand however, this can amount to a perceivable difference if the bound state adopts a conformation which is definitely higher than for the unbound ligand in solution. Such "errors" may partially cancel when series of similar ligands are compared in the same receptor. But it obviously adds to the fuzziness of the results. When in doubt, a trajectory of the ligand alone (under identical conditions as for the complex) should be recorded to assess the average energy of the ligand in the unbound state.

8.3 Some Technical Remarks on *pymdpbsa*

pymdpbsa uses *ffgbsa* (see section 5) or the stand-alone Poisson-Boltzmann solver *pbsa* to evaluate energies. The tool *ptraj* is called to decompose the MD trajectory into individual frames for the complex, the ligand, and the receptor.

Because various temporary files are generated during execution, *pymdpbsa* automatically creates a subdirectory in which all calculations are run. This subdirectory (extension `.tmpdir`) contains all temporary files and also the final results, copies of which are transfered to the starting working directory upon completion. By default, the temporary directory is **not** removed automatically.

The following files are necessary to run *pymdpbsa* on a receptor-ligand complex:

- a molecular dynamics trajectory file of the complex (any format that can be read by *ptraj*, including Z-compressed ones and binary binpos files like those created by *mdnab*, see section 7);
- three AMBER parameter-topology PRM files, one for the complex, one for the ligand alone, and one for the receptor alone (as created by *pytleap*, see section 4);

8.4 Running *pymdpbsa*

Invoking *pymdpbsa* without any arguments (or with `--help`) will list all possible options.

```
---------------------------------------------------------
 pymdpbsa version 0.6 (December 2010)
---------------------------------------------------------
Usage: pymdpbsa [options]

Options:
  -h, --help    show this help message and exit
  --proj=NAME   global project name
  --traj=FILE   MD trajectory file                (default: traj.binpos)
  --cprm=FILE   complex prmtop file               (default: com.prm)
  --lprm=FILE   ligand only prmtop file           (default: lig.prm)
  --rprm=FILE   receptor only prmtop file         (default: rec.prm)
  --lig=STRING  residue name of ligand            (default: LIG)
  --start=INT   first MD frame to be used         (default: 1)
  --stop=INT    last MD frame to be used          (default: 1)
  --step=INT    use every [step] MD frame         (default: 1)
  --solv=INT    0 for no solvation term (eps=r)
                1, 2, or 5 for GBSA
                3 for PBSA
                4 for PBSA/dispersion             (default: 1)
  --clean       clean up temporary files          (default: no clean)
```

You only need to specify options that are different from the default. Thus, you can avoid entering a lot of options by simply selecting file names like *com.prm*, *rec.prm*, and *lig.prm* for the PRM files, calling the trajectory file *traj.binpos*, and by giving the ligand the residue name LIG in your original structure file (the default if *pytleap* in section 4 was used).

`--proj` has to be followed by a the global name of the project and all output files will incorporate this string. The name of the temporary directory created will also start with the project name (followed by sequence of random characters and the extension '.tmpdir'). When this options is omitted, the project name becomes `None` (not really useful for later identification).

`--traj` is followed by the filename of the trajectory. As already mentioned, the trajectory file can be any format which can be processed by the AMBER tool *ptraj*. If the trajectory file name is *traj.binpos*, this option can be omitted.

`--cprm`, `--lprm`, `--rprm` are used to feed in the names for the PRM files of the complex, the ligand, and the empty receptor. None of these PRM files is generated by *pymdpbsa*. They must be specified by the user. If the *pytleap* utility (see section 4) has been used on a complex, these three files should have been created. If you want to use default names, rename these files to `com.prm`, `rec.prm`, and `lig.prm`.

`--lig` is used to specify the name of the ligand. This is the (up to 4 characters long) ”residue” name the ligand would have in a PDB file. If the complex has been prepared via *pytleap*, the ligand name will probably be LIG (i.e., the default). Note that the ligand is supposed to be one single residue in that case. Alternatively, the ligand can also be specified by its residue number. Thus if the ligand is residue 281 in the PDB file of the complex, you may specify `--lig 281`. This also allows to have multi-residue ligands like in protein-peptide (protein-protein) complexes. If e.g. the ligand covers residues 134 to 156 in the **overall** PDB file of the complex, you can specify `--lig '134-156'`.[11]

`--start`, `--stop`, and `--step` set the first and last frame of the MD trajectory to be used for evaluating the energy, and the step size (e.g., `--step 5` means every fifth frame). **By default, these values are all 1**, i.e., only the first frame is used. Thus, the free energy of interaction for a single PDB file can be computed by specifying as ’trajectory’ (with `--traj`) the name of the PDB file and neglecting the start/stop/step options.

`--solv` followed by an integer chooses the solvation option. The **default** is `'--solv 1'`. For values other than 1 to 5, the returned electrostatic energy term is evaluated with a distant-dependent dielectric function $\epsilon = r_{ij}$ with no additional polar solvation correction. For values 1, 2, or 5, the corresponding GB variant (`igb` in AMBER) is used with a nonpolar contribution of 0.0072 * SASA (where the solvent-accessible surface SASA is computed via *molsurf*); for `solv = 3`, GB is replaced by PB and the non-polar solvation energy term is 0.005 * SASA + 0.86; for `solv = 4`, the polar solvation free energy part is computed with PB, the nonpolar portion is evaluated by a ”cavity” term and a ”dispersion term”;[9] the detailed settings for this approach are identical to those suggested in the original *pbsa* documentation.

`--clean` removes the temporary directory, including all PDB or CRD files for the various MD frames. By default, these files are kept. You might choose to keep the files for debugging purposes in initial runs or for some graphics of overlays (since proteins are automatically RMS-fitted to the Cα during the *ptraj* extraction). In any case, the relevant data are saved to the working directory, even when the `--clean` option is used.

[11] Using quotes to include more complex atom masks is a safe way to circumvent problems with the shell interpretation.

8.5 Details on Internal Workings and Output of *pymdpbsa*

The internal workings and the output of *pymdpbsa* vary depending on the `--solv` options. In all cases, the *ptraj* tool is called to split the trajectory into individual frames. Since each interaction energy evaluation requires three files (complex, receptor, ligand), the splitting of a trajectory with N frames results in 3·N files.[12]

8.5.1 Distance-Dependent Dielectrics or Generalized Born

For `--solv` = 0, 1, 2, or 5, the *ffgbsa* routine is called to evaluate energy terms. Since *ffgbsa* required PDB files as coordinate input, the trajectory is split into individual PDB files. These files are named according to the project, the part of the structure (C for whole complex, R for receptor alone, L for ligand alone), and the frame number.

Thus a file `TEST.R.pdb.45` would be the PDB file of the empty receptor corresponding to frame 45 of the trajectory of the project named TEST.

Each run creates **four tables** with energy values returned by *ffgbsa*: one for the ligand, one for the receptor, one for the complex, and one for the interaction energies. The tables inherit the name of the project, followed by L, or R, or C, or D, (ligand, receptor, complex, and energy difference) and the extension ”`.nrg`”. These tables are simple text files and can be used as input for plotting routines, e.g., to check possible drifts or strong fluctuations. An excerpt of a `*.D.nrg` output is shown next:

```
  10     -56.84       0.00     -55.30     -61.67     67.98      -7.85
  20     -58.67      -0.00     -52.84     -68.51     70.51      -7.83
...
...
  90     -57.21       0.00     -56.83     -52.23     59.57      -7.72
 100     -59.20       0.00     -57.10     -41.51     47.27      -7.86
```

The first column is the frame number, followed by the total energy, the internal force field term (stretch, bend, and torsion terms), the van der Waals term, the Coulomb term, the Generalized-Born term, and the solvent-accessible surface term. **Note** that the internal force field term **must** be zero (within the limits of precision) in the `*.D.nrg` tables because we use a single trajectory and do not account for distortions in the receptor or the ligand. The corresponding columns in the respective C, R, and L tables will not be zero. In the special case `--solv 0`, the GB column has also zero values only.

The final evaluation summary is stored in a file with the project name and the extension ”`.sum`”. The summary shows averages and corresponding standard deviations and mean errors for all energy terms. All values are given in kcal·mol^{-1}. The header lines show additional information useful for later documentation. An example is shown below:

[12] The splitting into ligand and receptor is performed by separate *ptraj* calls. Depending on the part to be written out, the *ptraj* command ”`strip`” followed by an AMBER mask is used to remove the rest of the structure. Thus for example, if the ligand is a residue called LIG, the ligand alone is obtained with the strip mask ”`':*&!:LIG'`” meaning ”strip off all residues but not the residue named LIG”.

```
=======================================================================
Summary Statistics for Project SOLV5
Frames                : 10 to 100 (every 10)
Solvation             : GB (--solv=5)
Trajectory File       : traj.binpos
Complex  parmtop File : com.prm
Receptor parmtop File : rec.prm
Ligand   parmtop File : lig.prm
=======================================================================
-----Ligand Energies---------------------------------------------------
Etot  =    -169.82 (  3.62,   1.14) Ebat  =      64.78 (  4.69,   1.48)
Evdw  =      20.51 (  2.25,   0.71) Ecoul =    -192.35 (  1.61,   0.51)
EGB   =     -68.33 (  1.51,   0.48) Esasa =       5.56 (  0.06,   0.02)
-----Receptor Energies-------------------------------------------------
Etot  =   -4045.29 ( 31.63,  10.00) Ebat  =    4157.74 ( 37.50,  11.86)
Evdw  =    -756.47 ( 15.16,   4.79) Ecoul =   -4863.38 ( 94.96,  30.03)
EGB   =   -2681.64 ( 91.98,  29.09) Esasa =      98.45 (  0.54,   0.17)
-----Complex Energies--------------------------------------------------
Etot  =   -4276.73 ( 34.61,  10.94) Ebat  =    4222.53 ( 39.39,  12.46)
Evdw  =    -791.58 ( 14.82,   4.69) Ecoul =   -5110.62 (102.32,  32.36)
EGB   =   -2693.26 ( 97.49,  30.83) Esasa =      96.20 (  0.56,   0.18)
-----Interaction Energy Components-------------------------------------
Etot  =     -61.62 (  2.90,   0.92) Ebat  =      -0.00 (  0.01,   0.00)
Evdw  =     -55.62 (  1.43,   0.45) Ecoul =     -54.89 ( 12.81,   4.05)
EGB   =      56.70 ( 11.87,   3.75) Esasa =      -7.81 (  0.09,   0.03)
=======================================================================
```

For `--solv` = 0, 1, 2, or 5, the solvent-accessible surface is computed via the NAB subroutine *molsurf* in *ffgbsa*. The surface returned by *ffgbsa* is multiplied by a surface tension of **0.0072** to yield the ”nonpolar” free energy component in kcal/mol. For details about the calls to *molsurf*, see section 5.

As mentioned before, the *molsurf* routine is generally robust, but has shown problems in some rare cases. Since the *pymdpbsa* script requires the output from *molsurf* (called via *ffgbsa)*, we have built in a catch for these rare cases. If *molsurf* should fail, the returned surface value is set to zero for that frame and *pymdpbsa* emits a warning. In later statistical evaluations, frames with this problem are excluded from the evaluation, i.e., average values and standard deviations relate to ”healthy” frames only.

8.5.2 Poisson-Boltzmann

For `--solv` = 3 or 4, the *pbsa* routine is called. This is done by generating a temporary input (control) file for *pbsa* called `pbsasfe.in`. The output of *pbsa* goes to `pbsasfe.out`. Both files are left over after the run and can be used to verify that everything went correctly.

Since *pbsa* requires CRD files, the trajectory is split into AMBER restart files rather then PDB files. The name giving is the same as for the PDB files (see 8.5.1) except that the ”pdb” part in filenames is changed to ”crd”.

The script eventually calls *pbsa* by:

```
pbsa -O -i pbsasfe.in -o pbsasfe.out -p prmfile -c crdfile
```

The generated output tables are named as for the non-PB settings in section 8.5.1. However, the content of the tables varies slightly:

```
  10        5.85      -55.31     -61.69      92.17     -40.36      71.02
  20        2.61      -52.83     -68.54      92.79     -38.97      70.16
...
...
  90       -1.51      -56.83     -52.21      78.21     -40.25      69.57
 100        0.35      -57.10     -41.55      67.08     -39.25      71.16
```

The first column is the frame number, followed by the total energy, the van der Waals term, the Coulomb term, the Poisson-Boltzmann term, the solvent-accessible surface ("cavity") term, and the "dispersion term" (which is zero if the option `--solv=3` was used).

The final evaluation summary is stored in a file with the project name and the extension "`.sum`". This file is similar to that shown in section 8.5.1 except that some specific terms vary. An example is shown here:

```
========================================================================
Summary MDPBSA Statistics for Project SOLV4
Solvation              : PB+SAV+DISP (--solv=4)
Frames                 : 10 to 100 (every 10)
Trajectory File        : traj.binpos
Complex  parmtop File  : com.prm
Receptor parmtop File  : rec.prm
Ligand   parmtop File  : lig.prm
========================================================================
-----Ligand Energies----------------------------------------------------
Etot  =      32.16 (  2.27,   0.72) Evdw  =      -4.64 (  1.92,   0.61)
Ecoul =     106.04 (  1.70,   0.54) Epb   =     -72.46 (  1.37,   0.43)
Ecav  =      53.22 (  0.36,   0.11) Edisp =     -49.99 (  0.43,   0.14)
-----Receptor Energies--------------------------------------------------
Etot  =  -17754.52 ( 38.42,  12.15) Evdw  =   -1662.84 (  8.07,   2.55)
Ecoul =  -14139.19 (122.39,  38.70) Epb   =   -2662.75 ( 90.70,  28.68)
Ecav  =    1848.33 (  6.20,   1.96) Edisp =   -1138.08 (  5.39,   1.70)
-----Complex Energies---------------------------------------------------
Etot  =  -17719.32 ( 40.29,  12.74) Evdw  =   -1723.10 (  8.18,   2.59)
Ecoul =  -14088.04 (126.86,  40.12) Epb   =   -2652.15 ( 94.66,  29.93)
Ecav  =    1862.08 (  6.73,   2.13) Edisp =   -1118.11 (  5.69,   1.80)
-----Interaction Energy Components--------------------------------------
Etot  =       3.04 (  4.12,   1.30) Evdw  =     -55.62 (  1.43,   0.45)
Ecoul =     -54.90 ( 12.81,   4.05) Epb   =      83.06 ( 12.49,   3.95)
Ecav  =     -39.47 (  0.88,   0.28) Edisp =      69.96 (  0.84,   0.27)
========================================================================
```

8.6 Using *pymdpbsa* for Single-Point Interaction Energy

Since *ptraj* can read a single coordinate set (frame) as a "trajectory", *pymdpbsa* can also be used to generate the free energy of interaction for an isolated PDB or CRD file of a receptor-ligand complex. Just specify for the "trajectory" (`--traj`) the name of the single PDB or CRD file and leave the `--start`, `--step` and `--stop` options to their default of 1. Any of the `--solv` options can be used.

Obviously, the PRM files for the complex, the receptor, and the ligand, must be available also and must be specified if their names are different from the default.

In the single-point case, the output looks the same as for the multiple-frame evaluations. The tables have only one line (record) and the statistical data like standard deviation or standard error in the `.sum` file are all zero of course.

A Appendix A: Preparing PDB Files

The only required or useful data in a PDB file to set up AMBER simulations are: atom names, residue names, and maybe chain identifiers (if more than one chain is present), and the coordinates of heavy atoms. Non-protein structures (especially low-molecular-weight ligands) will cause problems, with the exception of water and some ions which are automatically recognized if their names in the PDB file correspond to the internal names in the AMBER libraries.

NOTE: Recent changes in *leap* are supposed to handle some of the hurdles (like generation of disulfide bonds) described below ”automatically”. I have not tested these options intensively. I suppose that they can be relied on in most cases but I still recommend to follow the recipes given below to be on the safe side.

A.1 Cleaning up Protein PDB Files for AMBER

This is a crucial step in the preparation and many potential problems and subsequent errors depend on this step!

Analyze the PDB file visually in any viewer that can represent (and maybe modify) the file. Alternatively, use a text editor. Delete all parts which are judged irrelevant for the simulation. Be aware that anything not protein or water can be expected to cause trouble later.

If the x-ray unit cell in the PDB file contains more than one image, choose the entity you want to use and delete the other(s).

If there is a **ligand**, save it as an MDL standard data file (SDF). Many software packages are able to do this directly. You may also save the ligand in PDB format and then use some other tools later to convert it into a decent SDF file (**including correct bond order and all hydrogens**). It is crucial to **keep the coordinates of its heavy atoms at their original location**. Then delete it from the PDB file. The ligand must treated separately later.

Delete all water molecules that are not considered relevant. Some waters might be essential for ligand binding. If those waters are kept, they should be made part of the receptor (as distinct ”residues”), not of the ligand. *leap* recognizes water if the residue name is WAT or HOH. In later simulations, they may have to be tethered (more or less strongly) to their original positions to prevent them from ”evaporating”.

Apply the same delete procedure to ions, co-factors, and other stuff that has no special relevance for the planned simulation.

Get rid off all protein (or peptide) hydrogens that are explicitly expressed in the PDB file. The AMBER *leap* utility adds hydrogens automatically with predefined names. Having hydrogens in PDB files with names that *leap* does not recognize within its residue libraries leads to a total mess.

Eventually, **remove also all connectivity records**. These are mostly referring to ligands, or, in some cases, to disulfide links. The latter should be explicitly re-connected (see later) without relying on connectivity records in the PDB file.

The final PDB file of the protein should only contain unique locations[13] for heavy atoms of amino acids (and maybe oxygens of specific water molecules). Missing atoms

[13] In some PDB files, the same amino acid may be represented by different states (conformations). You must decide which unique location you want to use later in the simulations.

in amino acids are mostly allowed since *leap* can rebuild them if the **residue names** are **correct** and if the **atoms** already present have **correct names** also.

Make use of ”TER” records to separate parts in the PDB file which are not connected covalently. This is especially important in protein structures in which parts are missing (gaps). Not separating the loose ends by a ”TER” record may lead to strange (and wrong) behavior in *leap* or later in the simulations. Apply the same rule to individual water molecules which you want to keep and separate each water by a ”TER” record.

A.2 Special Residues, Name Conventions, Chain Terminations

Tautomeric and protonation states are not rendered in PDB files. If a defined state for a residue is required, its **name** in the PDB file must reflect the choice. The following subsections deal with these cases. **Important:** if you change a residue name in a PDB file, make sure to change it for **all** atoms of that residue!

Note also that PDB files written out by *leap* will keep the ”special” names, which sometimes leads to annoying effects in software packages which are not prepared for amino acids called HIE, HIP, CYX, and alike. You might consider to change these names back to the standard prior to using these PDB files in other software packages. You can also use the *ambpdb* AMBER utility to do that (see the original AMBER documentation for details on this tool).

A.2.1 Histidine: HID, HIE, HIP

Histidine can exist in three forms (δ, ϵ, and protonated). The PDB file must reflect the choice of the user. In the current versions of *leap* command files included with AMBER, ϵ-histidine is the default, i.e., a ”HIS” residue in a PDB file will be translated automatically to HIE (for ϵ-histidine). If the residue is called ”HID” in the PDB file, the resulting residue for AMBER will become δ-histidine, while ”HIP” will yield the protonated form.

A.2.2 Cysteine: CYS, CYX

Cysteine can exist in free form or as part of a disulfide bridge. PDB residues named ”CYS” are automatically converted into a free cysteine with a SH side chain end. If the cysteine is known to be in a **S-S bridge**, the residue name in the PDB file **must** be ”**CYX**”. In that case, no hydrogen is automatically added to the side chain which ends in a bare sulfur. However, S-S bonds to pairing cysteines are not automatically made but must be specified by the user. The *pytleap* Python script described in section 4 takes care of this through a special command line option and a file specifying which residues are to be connected (page 10).

A.2.3 Protonation: ASH, GLH, LYN

Sometimes the usually charged residues aspartate ”**ASP**”, glutamate ”**GLU**”, and lysine ”**LYS**” might have to be used in their uncharged form. The residue names must then be changed to ”**ASH**”, ”**GLH**”, and ”**LYN**”, respectively. A neutral form of **arginine** is not foreseen in AMBER (as the pKa of arginine is around 12, it is always considered protonated).

A.2.4 Terminals: ACE, NHE, NME

There are special **N- and C-terminal cap residues** which can be used to neutralize the N- and C-terminal in peptide chains when the defaults (NH_3^+ for the N-terminal and COO^- for the C-terminal) are not appropriate.

The ”**ACE**” residue $[-C(=O)-CH_3]$ can be used to cap the N-terminal. The PDB entry of the capping residue ACE (this **name** is **compulsory**) must be:

```
ATOM       1  CH3 ACE    resnumber     x      y      z
ATOM       2  C   ACE    resnumber     x      y      z
ATOM       3  O   ACE    resnumber     x      y      z
```

Note the **atom name ”CH3”** for this special carbon! Another name is not allowed! Hydrogens should be omitted. They are automatically added if the residue name and the heavy atom names are correct.

For capping the C-terminal, two possibilities are given. The first one is a simple NH_2 termination giving $[C(=O)-NH_2]$. This residue **must** be called ”**NHE**” in the PDB file and consists of a single atom to be named **N**:

```
ATOM       1  N   NHE    resnumber     x      y      z
```

The second possible C-terminal cap is $NH-CH_3$, resulting in $[C(=O)-NH-CH_3]$ at the C-terminal. Its entry in the PDB file **must** have the residue name ”**NME**” and has the following PDB entry:

```
ATOM       1  N   NME    resnumber     x      y      z
ATOM       2  CH3 NME    resnumber     x      y      z
```

As above for ”ACE”, the atom name for the carbon must be ”CH3”! ”NHE” and ”NME” residues are automatically completed with hydrogens. Do not enter them explicitly.

Important: The ”ACE” residue should be the first residue in a chain (strand) while ”NHE” or ”NME” should be the last. If cap residues are used to terminate gaps in incomplete protein chains, they must appear at the exact gap location, respecting N-terminal and C-terminal order. Gaps must be separated by a ”TER” record in the PDB file. See section A.3.

A.3 Chains, Residue Numbering, Missing Residues

AMBER preparation modules assume that residues in a PDB file are connected and appear sequentially in the file. If not covalently connected (i.e., linked by an amide bond), the residues must be separated by ”**TER**” records in the PDB file. Thus for example, a protein consisting of two chains should have a ”TER” record after the final residue of the first chain. Similarly, if residues are missing (e.g., not detected in x-ray, or cut by the user), the gap should also be separated by a ”TER” record. Terminal residues will be charged by default. If the user wants to avoid this (especially for gaps), these residues should be capped (by ACE and NHE or NME).

In general, *leap* and tools calling it refer to the original **input residue numbers**. Thus, residues are numbered (rather ”named”) according to the original PDB file for special commands like the disulfide connections.

Important: In some PDB files, residue numbers are not following a simple sequential scheme. There may be added 'numbers' if the residue numbering should globally reflect that of a 'mother' protein of a whole family. In such cases, you may encounter residue numberings like e.g. 11.. 12.. 12A.. 12B.. 13.. etc, where 12A and 12B are insertions. This may lead to serious trouble when trying to refer to residue 'numbers' or 'names'. The safest way to avoid trouble is then to renumber the residues sequentially (without insertion or deletion letters) before using them in any tool that requires a precise reference to a residue name/number.

In **output files** from *leap* and related tools, **residues will always be numbered starting from 1**, irrespective of the original numbering. Gaps are not considered either. Thus if a protein chain runs from 21 to 80, with residues 31 to 40 (i.e., 10 residues) missing, the final numbering of residues will run from 1 to 50.

Important: The final residue numbers are the ones that must be used in later simulations to refer to individual residues via **AMBER masks** or **NAB atom expressions**. For example, if a protein chain with residues from 30 to 110 is prepared for AMBER simulations, the final numbering will go from 1 to 81. If the original residues 35 to 40 should be fixed or tethered, the actual residues to be specified are 6 to 11. This can lead to serious errors. So be careful about residue numbers. The script *pytleap* described later will always generate a new PDB file with exact AMBER residue numbering and atom names. This PDB file should be used as reference throughout all subsequent AMBER simulations. Above all, when using atom masks or atom expressions (see Appendix B), always check that they really refer to the desired atoms before running lengthy simulations. **Fixing or tethering wrong atoms are a common error which may easily go unnoticed**.

B Appendix B: Atom and Residue Selections

There are two standards to select atoms and residues in AMBER-related routines: the **AMBER ”mask”** notation, used by all original AMBER modules, and the **NAB ”atom expressions”**, which work only with NAB-compiled applications.

Users who only use the NAB routines presented in this document may skip to section B.2. Those who intend to use original AMBER routines should also become familiar with the AMBER masks notations.

B.1 Amber Masks

A ”mask” is a notation which selects atoms or residues for special treatment. A frequent usage is fixing or tethering selected atoms or residues during minimization or molecular dynamics.

The following lines are partially copied from the original AMBER documentation. For more details, refer to the entire section of that documentation describing the *ambmask* utility.[14]

The ”mask” selection expression is composed of ”elementary selections”. These start with **”:”** to select by residues, or **”@”** to select by atoms. Residues can be selected by numbers (given as numbers separated by commas, or as ranges separated by a dash) or by names (given as a list of residue names separated by commas). The same holds true for atom selections by atom numbers or atom names. In addition, atoms can be selected by AMBER atom type, in which case ”@” must be immediately followed by ”%”. The notation ”:*” means all residues and ”@*” means all atoms. The following examples show the usage of this syntax.

B.1.1 Residue Number List Examples

```
:1-10       = "residues 1 to 10"
:1,3,5      = "residues 1, 3, and 5"
:1-3,5,7-9 = "residues 1 to 3 and residue 5 and residues 7 to 9"
```

B.1.2 Residue Name List Examples

```
:LYS          = "all lysine residues"
:ARG,ALA,GLY  = "all arginine and alanine and glycine residues"
```

B.1.3 Atom Number List Examples

Note that these masks use the **actual sequential numbers of atoms** in the file. This is tricky and a serious source of error. You must know these numbers correctly. Using the atom numbers of a PDB file written out by an AMBER tool is an appropriate way to avoid pitfalls. **Do not use the original atom numbers from the raw PDB file you started with.**

```
@12,17        = "atoms 12 and 17"
@54-85        = "all atoms from 54 to 85
@12,54-85,90 = "atom 12 and all atoms from 54 to 85 and atom 90
```

[14] The utility *ambmask* is not part of the free Amber Tools but is available only together with the full AMBER package.

B.1.4 Atom Name List Examples

Atom names follow the standard names in PDB files for heavy atoms. For hydrogen atom names with more than 3 characters, the choice may be critical since some AMBER tools[15] wrap hydrogen atom names in the PDB files they write out, but internally use the "unwrapped" name version. For example, the second hydrogen atom at the first Cγ (e.g., in isoleucine) would be called HG12, but in the official PDB notation, it would be 2HG1. Since it very rarely (actually never) makes sense to fix individual hydrogen atoms in side chains, we do not worry about this. Even in ligand names, hydrogens are generally not the first choice of selection when fixing or tethering parts of the ligand.

```
@CA          = all atoms with the name CA (i.e., all C-alpha atoms)
@CA,C,O,N,H = all atoms with names CA or C or O or N or H
                   (i.e., the entire protein backbone)
```

B.1.5 Atom Type List Examples

This last mask type is only used by specialists and mentioned here for completeness. It allows the selection of AMBER atom types and requires detailed knowledge of AMBER force fields.

```
@%CT          = all atoms with the force field type CT
                (the standard sp3 aliphatic carbon)
@%N*,N3       = all atoms with the force field type N* or N3
                (N* is a special sp2 nitrogen, N3 is an sp3 nitrogen)
```

Note that in the above example, `N*` is actually an atom type. The * is **not** a wild card meaning "all N-something types"!

B.1.6 Logical Combinations

The selections above can be combined by various logical operators, including selections like "all atoms within a certain distance from...". The use of such combinations goes beyond this introductory script. Interested users should refer to the original AMBER documentation.

[15] Even that is not consistent because NAB-compiled routines use the unwrapped notation.

B.2 "Atom Expressions" in NAB Applications

NAB applications do not use the AMBER mask scheme outlined in the previous sections. They use simpler (but less powerful) selection criteria. The scheme is:

```
chains(or "strands"):residues:atoms
```

For example, `A:GLU:CA` would select all Cα carbons of all glutamate residues in chain A. A plain `::` would select all atoms in all residues and all chains (not very useful). `::H*` would select all hydrogen atoms in any chain and any residue, the `*` being a wild card for any sequence of characters. Similarly, `::*C*` would select all atoms which contain at least one "C" character, i.e., the wild card can be used in any position. The `?` can be used as a wild card for a single character. Thus, ::H? would select any atom starting with H plus one additional character (e.g., HC, H1, HN, but **not** HG11).

The wild card can also be used in residue names. `:A*:` would select all alanines, asparagines, and arginines.

Selections can be combined separated by a vertical bar "|". `:1-3,ALA:C*|:2-5:N*` would select all carbon atoms in residues 1 to 3, in all alanines **and** all nitrogen atoms in all residues from 2-5. If you would like to tether all Cα atoms of a protein and the oxygen atom of explicit water molecules (with residue names 'WAT'), you would use `::CA|:WAT:O*`.

Output from NAB applications always tells how many atoms have been selected for a special treatment. If you are not sure that your selection is correct, this number might at least be a hint. If you run a simulation with a protein having 200 residues and want to tether all Cα carbons, `::CA` should result in 200 selected atoms (provided that all residues have a well-defined CA atom, which they should).

C Appendix C: Examples and Test Cases

C.1 Example 1: Generating AMBER Files for Crambin with Disulfide Bonds

In crambin (`1CRN.pdb`, `...amberlite/examples/CRN`), there are 3 disulfide bonds connecting CYS3 to CYS40, CYS2 to CYS32, and CYS16 to CYS26. In the PDB file, these residues must all be changed from CYS to CYX. Then a text file (e.g. `sslinks`) should be created that looks like this:

```
 3   40
 2   32
16   26
```

In the `CRN examples` subfolder, the file *1crnx.pdb* is the modified *1CRN.pdb* file with the six cysteines above changed to CYX in their residue name. Also, everything has been removed except the `ATOM` records. Since we create explicitly the disulfide bonds via the `bond` command in *leap*, the connectivity records have been discarded also.

The **correct** command should be (assuming defaults for most settings):

```
pytleap --prot 1crnx.pdb --disul sslinks
```

where *sslinks* specifies the text file containing the numbers of the residues to be S-S linked (one pair per line). Now the disulfide bonds are recognized and registered in the PRM file, i.e., all bonded interactions for $-CH_2-S-S-CH_2-$ are correctly computed.

The file `leap.cmd` generated by *pytleap* shows the bonding between the corresponding SG atoms in the three disulfide linkages on lines 2 to 5:

```
set default pbradii mbondi
prot = loadpdb 1crnx.pdb
bond prot.3.SG prot.40.SG
bond prot.4.SG prot.32.SG
bond prot.16.SG prot.26.SG
saveamberparm prot 1crnx.leap.prm 1crnx.leap.crd
savepdb prot 1crnx.leap.pdb
quit
```

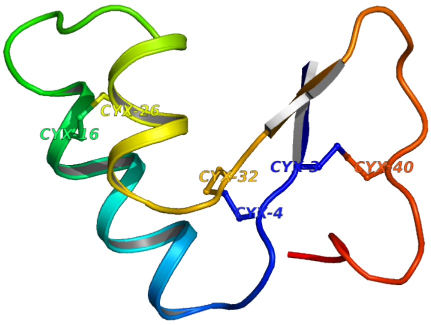

Figure 1: *The 3 S-S links in* 1crn.pdb

C.2 Example 2: Energy Minimization of the Crambin Structure

C.2.1 Starting Energy

We can use the files *1crnx.leap.prm* and *1crnx.leap.pdb* which were created in section C.1 to evaluate the AMBER energy terms in the unrefined crambin structure with the Generalized-Born option 1 and the SASA (nonpolar) energy:

```
ffgbsa 1crnx.leap.pdb 1crnx.leap.prm 1 1
```

The result is:

```
Reading parm file (1crnx.leap.prm)
title:

mm_options:  cut=100
mm_options:  rgbmax=100
mm_options:  diel=C
mm_options:  gb=1
      iter    Total       bad      vdW     elect   nonpolar   genBorn      frms
ff:      0   -813.56    611.05    -92.09   -980.60      0.00   -351.91  1.52e+01
sasa:    3079.71
Esasa = 0.0072 * sasa =      22.17
```

In this output, the line starting with "`ff:`" lists the total energy of the system and the components (`bad` = bond-angle-dihedral combined energy, i.e., the sum of the bonded terms). The line starting with "`sasa:`" gives the solvent-accessible surface in Å^{2}. The final line is the result from SASA multiplied by a surface tension of 0.0072. All energies are in kcal·mol^{-1}.[16]

This procedure is a good (although rough) health check of the PRM/PDB (and corresponding PRM/CRD) file pairs prior to using them in longer simulations. If the starting structure file is considered of good quality (no major steric bumps) but some of the values reported by *ffgbsa* look weird, there might be a serious error in the PRM file. If the coordinate file and the parameter-topology file are incompatible, e.g., different number of atoms or different order of atoms, *ffgbsa* will give very strange results in most cases (or fail completely).

C.2.2 Energy Minimization with *minab*

The structure refinement via conjugate gradient minimization can be carried out by the command (all on one line):

```
minab 1crnx.leap.pdb 1crnx.leap.prm crambin.min.pdb 1 1000
> crambin.min.out &
```

We use the GB option 1 and request a maximum of 1000 steps. No restraints are applied. The refined coordinates go to the PDB file *crambin.min.pdb*. The output of *minab* is redirected to the text file `crambin.min.out`. The final '`&`' puts the process into

[16] Note that the "nonpolar" term in the main energy components line is 0.00 because we compute this term separately. The "nonpolar" term in NAB applications can also include other terms (e.g., restraints) and is sometimes misleading.

the background. The minimization can be followed interactively by the command `tail -f crambin.min.out`

The last lines of the output are:

```
------------------------------
initial energy: -814.840 kcal/mol
final   energy: -1093.158 kcal/mol
minimizer finished after 619 iterations
refined coordinates written to crambin.min.pdb
------------------------------
```

Figure 2 shows the initial (green) and refined (orange) structure of crambin. The disulfide bonds in the refined structure are shown in CPK mode to emphasize that the S-S links have been correctly assigned. If this were not the case, the respective sulfur atoms would drift apart because of non-bonded interactions.

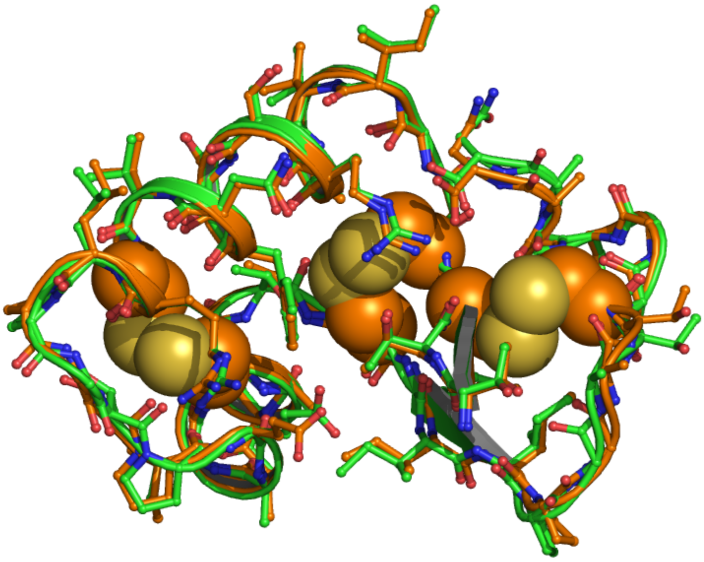

FIGURE 2: *Starting (green) and refined (orange) coordinates of 1CRN. Disulfide bonds in the refined structure are shown in CPK mode.*

C.3 Example 3: Preparation of a Complex between P38 MAP Kinase and Ligand

C.3.1 Cleaning Up PDB Entry *1OUK.pdb* for Usage with AMBER

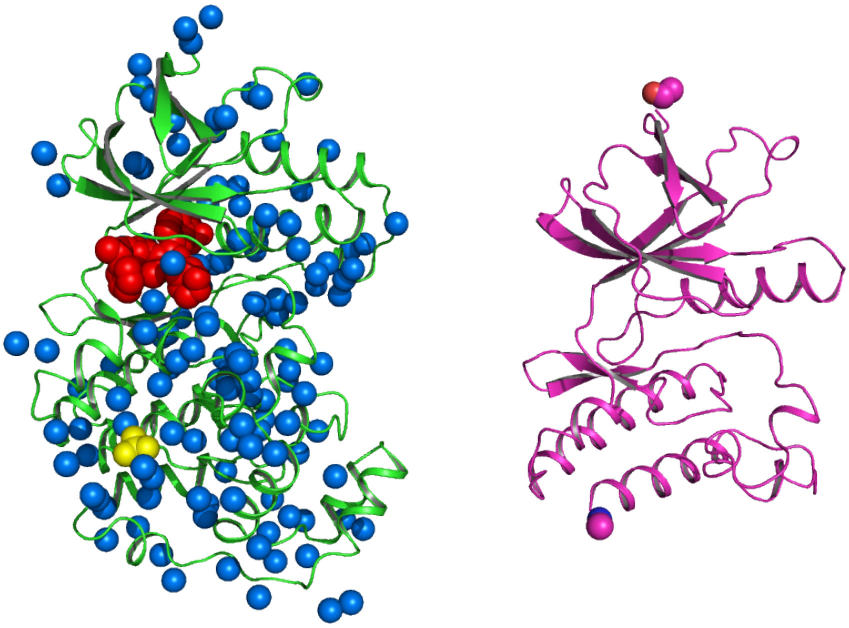

FIGURE 3: *Left side: original structure* 1OUK.pdb *with ligand (red), sulfate (yellow) and water molecules (blue); right side: final structure* p38.pdb *with the N- and C-terminal caps.*

In the `...amberlite/examples/P38` directory, the PDB file *1OUK.pdb* (P38 MAP kinase with inhibitor) is included in its original version. The structure (see Figure 3) contains a ligand (red), a sulfate ion (yellow), and a number of water molecules (blue). The file *p38.pdb* (also included in `...amberlite/examples/P38`) was created from this PDB file by cutting off a large part of the protein and deleting everything except the heavy atoms. The resulting ”nonnatural” N- and C-terminal were then completed by ACE and NME caps, respectively.[17] The resulting PDB file is ”clean” for *leap* and passes without errors. The ligand was processed separately into SDF format (file `lig.sdf` in `...amberlite/examples/P38`) including all hydrogens and bond orders. This file is ready to be processed via *antechamber* before re-complexing it with the protein (see later).

C.3.2 Generating AMBER Files for a P38/Ligand Complex

We re-use as a receptor the reduced and corrected PDB file from section C.3.1, *p38.pdb*. For the ligand, we use the *lig.sdf* file, containing the ligand with its heavy-atoms coordinates from the original pdb file *1OUK.pdb* and hydrogen atoms added via any other software that can handle this kind of problem. Note that the ligand has a formal charge of +1.

[17] This can be done by any modeling software which allows building, but make sure that the final atom and residue names in the cap residues are those described in section A.2.4.

The following command line will create the PRM, CRD, and PDB files for the empty receptor, the ligand, and the complex; partial charges on the ligand will be computed via the AM1-BCC method;[3,4] the complex will be named *com*:

```
pytleap --prot p38.pdb --lig lig.sdf --chrg 1 --cplx com
```

The longest part in the execution time is the processing of the ligand via the *sqm* tool to get the AMB1-BCC charges. During execution, various temporary files appear in the working directory. They result from the different modules called in *antechamber*. Most are removed when *antechamber* has finished.

The resulting AMBER files are called `*.leap.prm`, `*.leap.crd`, and `*.leap.pdb`, respectively. One set of files is generated for the ligand (`lig`), the receptor (`p38`), and the complex (`com`).[18]

Among the various other files left over, `lig.leap.frcmod` might be the most relevant to inspect since it contains parameters which were used in addition to those explicitly present in the *gaff* parameter set.

The file `lig.ac.mol2` is the MOL2 for the ligand containing the *gaff* atom types and the AM1-BCC partial charges. This file can be read by a variety of software packages but the atom elements will not be recognized because atom types are not original TRIPOS atom types, indicating the chemical element.

In the resulting complex, the ligand has the residue name LIG and the residue number 217. The N- and C-terminal caps ACE and NME get residue numbers 1 and 216.

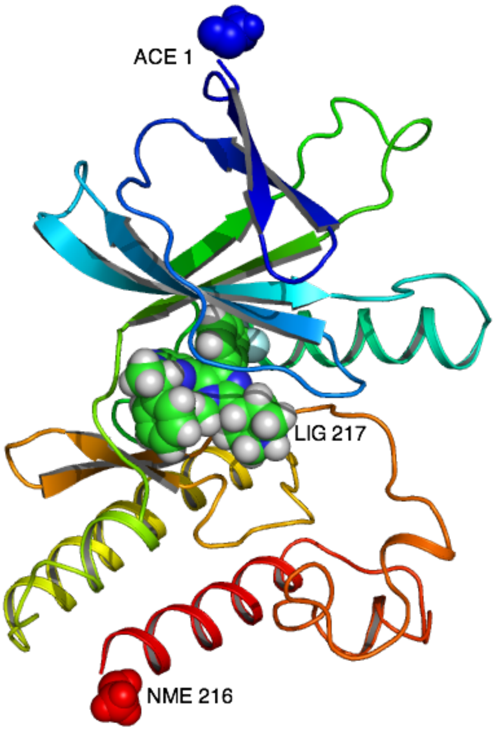

Figure 4: *The final complex structure* com.leap.pdb*: The ACE cap becomes residue 1, the NME cap is residue 216, and the ligand is residue LIG 217*

[18] Note that we use the `*.leap.*` name giving to underline that these files have been generated via leap. This is useful to avoid confusion, especially for the PDB or CRD files which must correspond to the respective PRM files.

C.4 Example 4: Interaction Energy between P38 and Ligand in the Unrefined (Original) Complex

We use the files generated in example 3 (section C.3.2). In order to make use of the default settings in the command line options, we copy the respective files to the default names proposed by *pymdpbsa*: Make copies (or symbolic links) of *com.leap.prm*, *p38.leap.prm*, and *lig.leap.prm* to *com.prm*, *rec.prm*, and *lig.prm*. Also, make a copy (or symbolic link) of *com.leap.pdb* to *com.pdb*.

Now the command

```
pymdpbsa --proj RAWPDB --traj com.pdb
```

computes the interaction energy. As "trajectory" (`--traj`) we specify the single complex PDB file `com.pdb`. We call the project "RAWPDB" and leave all other input options at their default, i.e., we also use the default GB option 1.

A subdirectory `RAWPDB_xxxxxx.tmpdir` is generated, where `xxxxxx` is a random sequence of characters. This temporary directory can be removed since the relevant output files are copied to the directory in which *pymdpbsa* was started. We could also have used the additional command line option `--clean` to remove the temporary directory automatically.

The output of interest is the summary file `RAWPDB.sum` (see also 8.5.1 and 8.5.2):

```
=======================================================================
Summary Statistics for Project RAWPDB
Frames                 : 1 to 1 (every 1)
Solvation              : GB (--solv=1)
Trajectory File        : com.pdb
Complex  parmtop File : com.prm
Receptor parmtop File : rec.prm
Ligand   parmtop File : lig.prm
=======================================================================
-----Ligand Energies---------------------------------------------------
Etot  =    -127.64 (  0.00,   0.00) Ebat  =     117.53 (  0.00,   0.00)
Evdw  =       3.79 (  0.00,   0.00) Ecoul =    -183.01 (  0.00,   0.00)
EGB   =     -71.71 (  0.00,   0.00) Esasa =       5.76 (  0.00,   0.00)
-----Receptor Energies-------------------------------------------------
Etot  =   -4072.40 (  0.00,   0.00) Ebat  =    2653.90 (  0.00,   0.00)
Evdw  =     657.95 (  0.00,   0.00) Ecoul =   -4409.58 (  0.00,   0.00)
EGB   =   -3068.98 (  0.00,   0.00) Esasa =      94.31 (  0.00,   0.00)
-----Complex Energies--------------------------------------------------
Etot  =   -4250.68 (  0.00,   0.00) Ebat  =    2771.43 (  0.00,   0.00)
Evdw  =     608.82 (  0.00,   0.00) Ecoul =   -4636.90 (  0.00,   0.00)
EGB   =   -3086.27 (  0.00,   0.00) Esasa =      92.24 (  0.00,   0.00)
-----Interaction Energy Components-------------------------------------
Etot  =     -50.64 (  0.00,   0.00) Ebat  =       0.00 (  0.00,   0.00)
Evdw  =     -52.92 (  0.00,   0.00) Ecoul =     -44.31 (  0.00,   0.00)
EGB   =      54.42 (  0.00,   0.00) Esasa =      -7.83 (  0.00,   0.00)
=======================================================================
```

C.5 Example 5: Minimization of P38 Complex with `minab` and Resulting Interaction Energy

We minimize the P38/ligand complex prepared in section C.3. We use the (renamed) PDB file *com.pdb*, the corresponding PRM file *com.prm*, gb = 1 and a maximum of 500 iterations. We tether Cα atoms with a force constant of 1.0 kcal·mol^{-1}·Å^{-2}. The refined coordinates are written to com.min.pdb. We redirect the output to a file `minab.out`.

For the command line

```
minab com.pdb com.prm com.min.pdb 1 500 ’::CA’ 1.0 > minab.out &
```

the output file `minab.out` would be:

```
Reading parm file (com.prm)
title:

        mm_options:  cut=100.000000
        mm_options:  nsnb=501
        mm_options:  diel=C
        mm_options:  gb=1
        mm_options:  rgbmax=15.000000
        mm_options:  wcons=1.000000
        mm_options:  ntpr=10
constrained 214 atoms using expression ::CA
constrained 214 atoms from input array
      iter    Total       bad       vdW     elect   nonpolar   genBorn      frms
ff:      0  -4378.52   2771.43    608.82  -4636.90       0.00  -3121.87  3.31e+01
ff:     10  -5630.74   2669.08   -446.65  -4721.49       0.11  -3131.79  4.86e+00
```

...more like this cut from this demo output...

```
ff:    490  -6861.80   2521.23  -1205.14  -5170.41      16.59  -3024.07  1.67e-01
ff:    500  -6862.46   2521.87  -1205.86  -5170.56      16.23  -3024.15  1.41e-01
------------------------------
initial energy: -4378.522 kcal/mol
final   energy: -6862.463 kcal/mol
minimizer stopped because number of iterations was exceeded
refined coordinates written to com.min.pdb
------------------------------
```

The minimization did not reach the requested default rms of the components of the gradient of 0.1, but stopped after the required 500 iterations.

The final line reminds that the refined structure has been saved into the PDB file `com.min.pdb`. Note that the energy term listed here under ”nonpolar” is actually the energy stemming from the restraints, in this example tethering all Cα atoms.

We can now repeat the interaction energy computation on the refined complex, using the same settings as for the raw PDB file in section C.4:

```
pymdpbsa --proj REFINEDPDB --traj com.min.pdb
```

with `--traj` now specifying the refined PDB file `com.min.pdb`. The resulting summary `REFINEDPDB.sum` is:

```
=======================================================================
Summary Statistics for Project MINPDB
Frames                 : 1 to 1 (every 1)
Solvation              : GB (--solv=1)
Trajectory File        : com.min.pdb
Complex  parmtop File : com.prm
Receptor parmtop File : rec.prm
Ligand   parmtop File : lig.prm
=======================================================================
-----Ligand Energies---------------------------------------------------
Etot  =     -210.13 (  0.00,   0.00) Ebat  =       34.74 (  0.00,   0.00)
Evdw  =       15.06 (  0.00,   0.00) Ecoul =     -195.36 (  0.00,   0.00)
EGB   =      -70.16 (  0.00,   0.00) Esasa =        5.61 (  0.00,   0.00)
-----Receptor Energies-------------------------------------------------
Etot  =    -6470.57 (  0.00,   0.00) Ebat  =     2487.56 (  0.00,   0.00)
Evdw  =    -1160.97 (  0.00,   0.00) Ecoul =    -4904.64 (  0.00,   0.00)
EGB   =    -2988.22 (  0.00,   0.00) Esasa =       95.70 (  0.00,   0.00)
-----Complex Energies--------------------------------------------------
Etot  =    -6750.93 (  0.00,   0.00) Ebat  =     2522.30 (  0.00,   0.00)
Evdw  =    -1205.93 (  0.00,   0.00) Ecoul =    -5170.51 (  0.00,   0.00)
EGB   =    -2990.31 (  0.00,   0.00) Esasa =       93.52 (  0.00,   0.00)
-----Interaction Energy Components-------------------------------------
Etot  =      -70.25 (  0.00,   0.00) Ebat  =        0.00 (  0.00,   0.00)
Evdw  =      -60.02 (  0.00,   0.00) Ecoul =      -70.51 (  0.00,   0.00)
EGB   =       68.07 (  0.00,   0.00) Esasa =       -7.79 (  0.00,   0.00)
=======================================================================
```

C.6 Example 6: Generate MD Trajectory for the P38-Ligand Complex with *mdnab*

We use *mdnab* to run a 100 picoseconds MD trajectory of P38 complex, using as starting geometry the refined complex `com.min.pdb` from the previous section:

```
mdnab com.min.pdb com.prm com 1 100 ’::CA’ 1.0 > md.out &
```

The trajectory will go to the file `com.binpos`, specified as the third command line argument (the extension ” `.binpos`” is appended automatically). We tether the Cα atoms with the same force as for the minimization in C.5 through the last two arguments *’::CA’* and *1.0*. The GB option ’1’ is used (fourth argument for the `mdnab` command).

The file `md.out` will start with:

```
Reading parm file (com.prm)
title:

mm_options:  cut=12.000000
mm_options:  nsnb=25
mm_options:  diel=C
mm_options:  gb=1
mm_options:  rgbmax=12.000000
```

```
mm_options:  rattle=1
mm_options:  dt=0.002000
mm_options:  ntpr=101
mm_options:  ntpr_md=10
mm_options:  ntwx=0
mm_options:  zerov=0
mm_options:  tempi=50.000000
mm_options:  temp0=100.000000
mm_options:  gamma_ln=20.000000
mm_options:  wcons=1.000000
constrained 214 atoms using expression ::CA
```

The last two lines shown above indicate that all Cα atoms (214 in this case) have indeed been tethered with a force constant `wcons=1.000000`. It is important to verify this line to make sure that the atom selection on the command line (in this case `'::CA'`) had the desired effect, especially if more complex expressions are used.

The output file `md.out` then continues through the heat-up and equilibration stages. Then the time is reset to zero and the production phase begins. The final lines in the example above are:

```
...
...
md:        49500     99.000     2137.72    -4563.14   -2425.42      302.35
md:        50000    100.000     2084.67    -4519.03   -2434.36      294.84

trajectory with 100 picoseconds was written to com.binpos...
```

confirming that 50000 steps (i.e. 100 picoseconds with a stepsize of 2 femtoseconds) have been recorded and written to the trajectory file `com.binpos`.

C.7 Example 7: Running *pymdpbsa* on the P38/Ligand Complex Trajectory

If we have previously renamed all PRM files to the expected defaults, since the ligand is called ”LIG” by default in *pytleap*, and since we want the default GB 1 option, we only enter the project name `P38`. We use every tenth frame from the total 100-frames production phase of the trajectory, so that `--start 10`, `--stop 100`, and `--step 10` are used.

The command line is:

```
pymdpbsa --proj P38 --traj com.binpos --start 10 --stop 100 --step 10&
```

The summary of the results goes into the file `P38.sum` and is shown below.

```
=======================================================================
Summary Statistics for Project P38
Frames                : 10 to 100 (every 10)
Solvation             : GB (--solv=1)
Trajectory File       : com.binpos
Complex  parmtop File : com.prm
Receptor parmtop File : rec.prm
Ligand   parmtop File : lig.prm
=======================================================================
-----Ligand Energies---------------------------------------------------
Etot  =    -171.86 (  3.80,   1.20) Ebat  =      65.11 (  3.96,   1.25)
Evdw  =      19.74 (  2.06,   0.65) Ecoul =    -191.30 (  2.13,   0.67)
EGB   =     -71.04 (  0.68,   0.22) Esasa =       5.62 (  0.05,   0.01)
-----Receptor Energies-------------------------------------------------
Etot  =   -4246.59 ( 39.66,  12.54) Ebat  =    4156.11 ( 34.62,  10.95)
Evdw  =    -772.67 ( 20.92,   6.61) Ecoul =   -4921.45 ( 38.73,  12.25)
EGB   =   -2804.11 ( 27.70,   8.76) Esasa =      95.53 (  0.57,   0.18)
-----Complex Energies--------------------------------------------------
Etot  =   -4478.81 ( 40.64,  12.85) Ebat  =    4221.22 ( 36.59,  11.57)
Evdw  =    -806.44 ( 22.34,   7.07) Ecoul =   -5161.88 ( 42.37,  13.40)
EGB   =   -2825.07 ( 30.74,   9.72) Esasa =      93.36 (  0.60,   0.19)
-----Interaction Energy Components-------------------------------------
Etot  =     -60.36 (  3.30,   1.04) Ebat  =      -0.00 (  0.01,   0.00)
Evdw  =     -53.51 (  3.65,   1.15) Ecoul =     -49.14 ( 12.02,   3.80)
EGB   =      50.08 ( 10.21,   3.23) Esasa =      -7.80 (  0.12,   0.04)
=======================================================================
```

The numbers in parentheses after the actual energy values are the standard deviation and the standard error of the mean (SEM). Note that the energy term `Ebat` (the sum of the bond, angle, and torsion terms) for the interaction energy is zero (or almost so, because of rounding errors). This is the obvious consequence of the single-trajectory approach because we neglect any strain in the ligand or the receptor. The strain would have to be evaluated by running three distinct trajectories (i.e., also for the free ligand and the empty receptor).

A directory `P38_xxxxxx.tmpdir` has been created which contains all files used for the computation, including the individual structures of each frame. You can savely remove this directory if you are only interested in the actual results, i.e., the summary file `*.sum` and the `*.X.nrg` tables, where `X` can be C (for complex), R (for receptor), L (for ligand), and D (for the actual ΔE and ΔG values).

References

[1] Wang, J.; Wang, W.; Kollman, P.; Case, D. *J. Mol. Graphics Modell.* **2006**, *25*, 247–260.

[2] Wang, J.; Wolf, R. M.; Caldwell, J. W.; Kollman, P. A.; Case, D. A. *J. Comput. Chem.* **2004**, *25*, 1157–1174.

[3] Jakalian, A.; Bush, B. L.; Jack, D. B.; Bayly, C. I. *J. Comput. Chem.* **2000**, *21*, 132–146.

[4] Jakalian, A.; Jack, D. B.; Bayly, C. I. *J. Comput. Chem.* **2002**, *23*, 1623–1641.

[5] Chong, L. T.; Duan, Y.; Wang, L.; Massova, I.; Kollman, P. A. *PNAS* **1999**, *96*, 14330–14335.

[6] Kollman, P. A.; Massova, I.; Reyes, C.; Kuhn, B.; Huo, S.; Chong, L.; Lee, M.; Lee, T.; Duan, Y.; Wang, W.; Donini, O.; Cieplak, P.; Srinivasan, J.; Case, D. A.; Cheatham, T. E. *Acc. Chem. Res.* **2000**, *33*, 889–897.

[7] Massova, I.; Kollman, P. *Perspectives in Drug Discovery and Design* **2000**, *18*, 113–135.

[8] Huo, S.; Massova, I.; Kollman, P. A. *J. Comput. Chem.* **2002**, *23*, 15–27.

[9] Tan, C.; Tan, Y.-H.; Luo, R. *J. Phys. Chem. B* **2007**, *111*, 12263–12274.