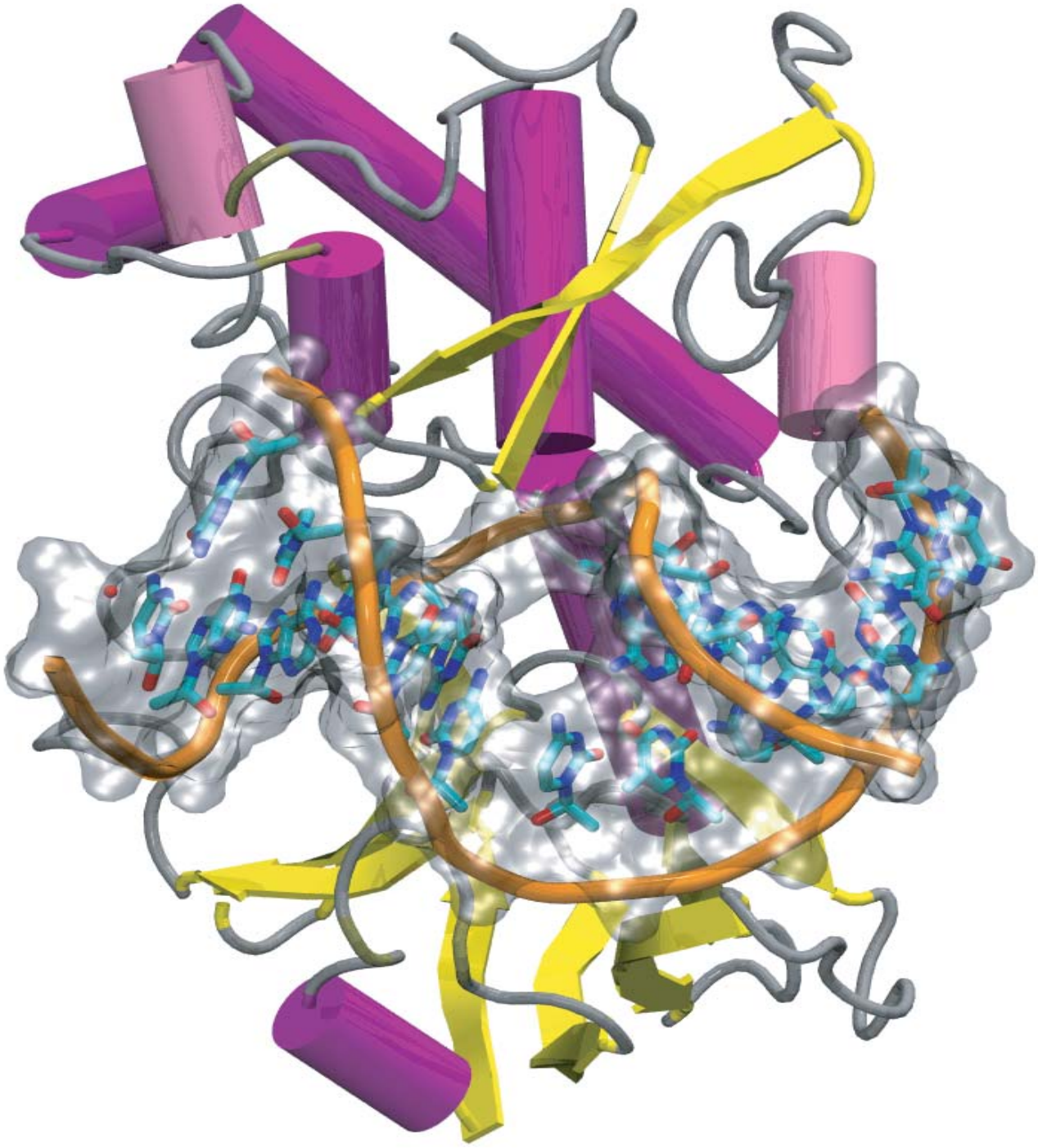
Amber 8
Users' Manual



AMBER 8
Users' Manual

Principal contributors to the current codes:
David A. Case (The Scripps Research Institute)
Tom Darden (NIEHS)
Thomas E. Cheatham III (University of Utah)
Carlos Simmerling (Stony Brook)
Junmei Wang (Encysive Pharmaceuticals)
Robert E. Duke (University of North Carolina)
Ray Luo (UC Irvine)
Kenneth M. Merz (Penn State)
Bing Wang (Penn State)
David A. Pearlman (UC San Francisco)
Mike Crowley (TSRI)
Scott Brozell (TSRI)
Vickie Tsui (TSRI)
Holger Gohlke (TSRI, J.W. Goethe-Universität)
John Mongan (UC San Diego)
Viktor Hornak (Stony Brook)
Guanglei Cui (Stony Brook)
Paul Beroza (Telik)
Christian Schafmeister (UC San Francisco)
James W. Caldwell (UC San Francisco, Stanford)
Wilson S. Ross (UC San Francisco)
Peter A. Kollman (UC San Francisco)

Additional key contributors to earlier versions:
Robert V. Stanton (UC San Francisco)
Jed Pitera (UC San Francisco)
Irina Massova (UC San Francisco)
Ailan Cheng (Penn State)
James J. Vincent (Penn State)
Randall Radmer (UC San Francisco)
George L. Seibel (UC San Francisco)
U. Chandra Singh (UC San Francisco)
Paul Weiner (UC San Francisco)

Additional key people involved in force field development:
Piotr Cieplak (Accerlys)
Yong Duan (University of Delaware)
Rob Woods (University of Georgia)
Karl Kirschner (University of Georgia)
Sarah M. Tschampel (University of Georgia)
Alexey Onufriev (Virginia Tech.)
Christopher Bayly (Merck-Frost)
Wendy Cornell (UC San Francisco, Novartis)
Scott Weiner (UC San Francisco)

Acknowledgments

We acknowledge the generous cooperation of Wilfred van Gunsteren, whose molecular dynamics code was used as the basis of the md modules in version 2.0. We are also pleased to acknowledge Rad Olson and Bill Swope at IBM Almaden Center, whose contributions were instrumental in developing the better vector optimized non-bonded routines first released in version 3, revision A. Research support from DARPA, NIH and NSF for Peter Kollman is gratefully acknowledged, as is support from NIH, NSF and DOE for David Case. Use of the facilities of the UCSF Computer Graphics Laboratory (Thomas Ferrin, PI) is appreciated. The pseudocontact shift code was provided by Ivano Bertini of the University of Florence. We thank Chris Bayly and Merck-Frosst, Canada for permission to include charge increments for the AM1-BCC charge scheme. Many people helped add features to various codes; these contributions are described in the documentation for the individual programs.

Recommended Citations:

When citing Amber Version 8 in the literature, the following citation should be used:

> D.A. Case, T.A. Darden, T.E. Cheatham, III, C.L. Simmerling, J. Wang, R.E. Duke, R. Luo, K.M. Merz, B. Wang, D.A. Pearlman, M. Crowley, S. Brozell, V. Tsui, H. Gohlke, J. Mongan, V. Hornak, G. Cui, P. Beroza, C. Schafmeister, J.W. Caldwell, W.S. Ross, and P.A. Kollman (2004), AMBER 8, University of California, San Francisco.

The history of the codes and a basic description of the methods can be found in:

> D.A. Pearlman, D.A. Case, J.W. Caldwell, W.S. Ross, T.E. Cheatham, III, S. DeBolt, D. Ferguson, G. Seibel, and P. Kollman. AMBER, a package of computer programs for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to simulate the structural and energetic properties of molecules. *Comp. Phys. Commun.* **91,** 1-41 (1995).

Peter Kollman died unexpectedly in May, 2001. We dedicate Amber to his memory.

Cover Illustration

The cover image shows the complex between duplex DNA and the formamidopyrimidine-DNA glycosylase (called Fpg or MutM) involved in repair of oxidatively damaged bases. The coordinates represent a covalently trapped complex following removal of 8-oxo-G, and were obtained from X-ray crystallography (pdb code 1K82). The enzyme is shown as a schematic and the DNA duplex is shown with solvent-accessible surface and heavy atoms for the bases. The image was produced by Asim Okur, Adrian Roitberg and Carlos Simmerling.

Table of Contents

1. Introduction

Amber is the collective name for a suite of programs that allow users to carry out molecular dynamics simulations, particularly on biomolecules. None of the individual programs carries this name, but the various parts work reasonably well together, and provide a powerful framework for many common calculations [1]. The term *amber* is also sometimes used to refer to the empirical force fields that are implemented here [2]. It should be recognized however, that the code and force field are separate: several other computer packages have implemented the *amber* force fields, and other force fields can be implemented with the *amber* programs. Further, the force fields are in the public domain, whereas the codes are distributed under a license agreement.

Amber 8 (2004) represents a significant change from the most recent previous version, *Amber 7*, which was released in March, 2002. Briefly, the major differences include:

(1) *Force fields:* Four new force fields are available: a re-parameterization of an all-atom protein force field (*ff03*), based on quantum calculations in a continuum solvent environment [3]; a major extension of the General Amber Force Field (*gaff*), that expands the range of applicable molecules, particularly for conjugated systems [4]; a new version of the "glycam" carbohydrate force field (*glycam04*) developed in Rob Woods' group [5-7]; and a "QM/MM" facility by which part of the system can use energies and forces derived from a semiempirical Hamiltonian such as AM1 or PM3.

(2) *Generalized Born models:* Amber 8 features a new parameterization with increased fidelity to Poisson-Boltzmann results [8], and a faster implementation. GB models can now use the locally-enhanced sampling (LES) method, and constant pH simulations can be carried out. Langevin dynamics are available to simulate solvent frictional effects.

(3) *Explicit solvent simulations:* Users can now perform non-periodic simulations (such as an enzyme in a spherical "shell" of water) with a reaction field (Poisson-Boltzmann) model of long-range electrostatic effects [9,10]. Explicit solvent periodic calculations are generally faster than in previous versions of Amber, and scale better in parallel environments.

(4) *Sampling and free energy calculations:* Replica exchange (or "multicanonical") simulations are now supported, and there are improved methods for potential of mean force and thermodynamic integration free energy simulations.

(5) *Connections to other programs:* Amber can use MD-GRAPE hardware from RIKEN, and can link into the "low-mode" (LMOD) methods from Istvan Kolossváry for docking and conformational analysis; (for the latter, see *amber8/doc/lmod.pdf*). The Amber suite now includes programs for semi-empirical quantum calculations, and for finite-difference Poisson-Boltzmann calculations.

1.1. What to read next

If you are installing this package see Section 1.3. New users should continue with this Chapter, and should consult the tutorial information in Section 1.4, and the scientific overviews in Chapter 12. Everyone should read Chapter 2, which contains information about force fields, and which is extensively revised from earlier versions of Amber. There are also tips and examples on the Amber Web pages at `http://amber.scripps.edu`. Although Amber may appear

dauntingly complex at first, it has become easier to use over the past few years, and overall is reasonably straightforward once you understand the basic architecture and option choices. In particular, we have worked hard on the tutorials to make them accessible to new users. Hundreds of people have learned to use Amber; don't be easily discouraged.

If you want to learn more about basic biochemical simulation techniques, there are a variety of good books to consult, ranging from introductory descriptions [11,12], to standard works on liquid state simulation methods [13,14], to multi-author compilations that cover many important aspects of biomolecular modelling [15-17]. Looking for "paradigm" papers that report simulations similar to ones you may want to undertake is also generally a good idea.

1.2. Information flow in Amber

Understanding where to begin in Amber is primarily a problem of managing the flow of information in this package--see Fig. 1. You first need to understand what information is needed by the simulation programs (*sander, pmemd* and *nmode*). You need to know where it comes from, and how it gets into the form that the energy programs require. This section is meant to orient the new user and is not a substitute for the individual program documentation.

Information that all the simulation programs need:

(1) Cartesian coordinates for each atom in the system. These usually come from Xray crystallography, NMR spectroscopy, or model-building. They should be in Protein Databank (PDB) format. The program *LEaP* provides a platform for carrying out many of these

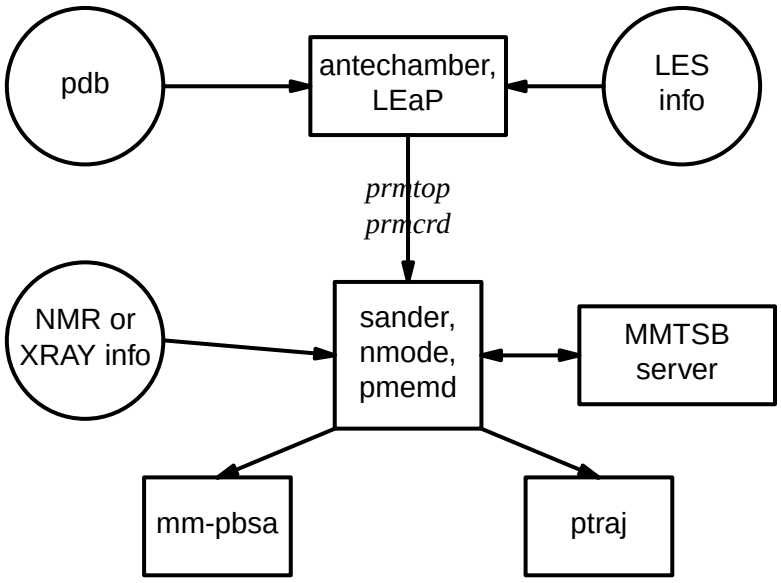


Fig. 1. Basic information flow in Amber

modeling tasks, but users may wish to consider other programs as well.

(2) "Topology": connectivity, atom names, atom types, residue names, and charges. This information comes from the database, which is found in the *amber8/dat/leap/prep* directory, and is described in Chapter 2. It contains topology for the standard amino acids as well as N- and C-terminal charged amino acids, DNA, RNA, and common sugars. The database contains default internal coordinates for these monomer units, but coordinate information is usually obtained from PDB files. Topology information for other molecules (not found in the standard database) is kept in user-generated "residue files", which are generally created using *antechamber*.

(3) Force field: Parameters for all of the bonds, angles, dihedrals, and atom types in the system. The standard parameters for several force fields are found in the *amber8/dat/leap/parm* directory; consult Chapter 2 for more information. These files may be used "as is" for proteins and nucleic acids, or users may prepare their own files that contain modifications to the standard force fields.

(4) Commands: The user specifies the procedural options and state parameters desired. These are specified in the input files (usually called *mdin*) to the sander, pmemd, or nmode programs.

1.2.1. Preparatory programs

LEaP is the primary program to create a new system in Amber, or to modify old systems. It combines the functionality of `prep, link, edit,` and `parm` from earlier versions.

ANTECHAMBER is the main program from the Antechamber suite. If your system contains more than just standard nucleic acids or proteins, this may help you prepare the input for LEaP.

1.2.2. Simulation programs

SANDER is the basic energy minimizer and molecular dynamics program. This program relaxes the structure by iteratively moving the atoms down the energy gradient until a sufficiently low average gradient is obtained. The molecular dynamics portion generates configurations of the system by integrating Newtonian equations of motion. MD will sample more configurational space than minimization, and will allow the structure to cross over small potential energy barriers. Configurations may be saved at regular intervals during the simulation for later analysis, and basic free energy calculations using thermodynamic integration may be performed.

More elaborate conformational searching and modeling MD studies can also be carried out using the SANDER module. This allows a variety of constraints to be added to the basic force field, and has been designed especially for the types of calculations involved in NMR structure refinement.

PMEMD is a version of *sander* that is optimized for speed and for parallel scaling. The name stands for "Particle Mesh Ewald Molecular Dynamics," and it is restricted

to this sort of simulation. The input and output have only a few changes from *sander*.

NMODE is both a quasi-Newton Raphson second derivative energy minimizer and vibrational analysis program. NMODE can calculate the normal modes of the system as well as numerous thermochemical properties. Other features include the ability to compute "Langevin modes" (normal modes including viscous coupling to a continuum solvent) and techniques to find transitions states as well as minima.

1.2.3. Analysis programs

PTRAJ is a general purpose utility for analyzing and processing trajectory or coordinate files created from MD simulations (or from various other sources), carrying out superpositions, extractions of coordinates, calculation of bond/angle/dihedral values, atomic positional fluctuations, correlation functions, analysis of hydrogen bonds, etc. The same executable, when named rdparm (from which the program evolved), can examine and modify prmtop files.

MM-PBSA is a script that automates energy analysis of snapshots from a molecular dynamics simulation using ideas generated from continuum solvent models.

1.3. Installation of Amber 8

To compile the basic AMBER distribution, do the following:

(1) Set up the `AMBERHOME` environment variable to point to where the Amber tree resides on your machine. For example

```
Using csh, tcsh, etc:          setenv AMBERHOME  /usr/local/amber8

Using bash, sh, zsh, etc:      set AMBERHOME=/usr/local/amber8
                               export AMBERHOME
```

NOTE: Be sure to replace the "/usr/local" part above with whatever path is appropriate for your machine. You should then add `$AMBERHOME/exe` to your PATH.

(2) Go to the `$AMBERHOME/src` directory, and create a configuration file for a serial version:

```
./configure -help
```

will show you the options available. Choose a machine/compiler name, for example:

```
./configure -p4 ifort
```

This will create a `config.h` file for a single-processor Pentium IV machine using the Intel Fortran 90 compiler. You can examine and edit this file to match your local environment, if necessary. Do not choose any parallel options (`-mpich, -lam,`...) at this point.

(3) Now compile everything:

```
make serial
```

Loader warnings (especially on SGI) can generally be ignored; compiler warnings should be considered, but most are innocuous. If a program that you don't need initially fails to compile, you should consider invoking "make" with the ignore errors option (`make -i`) or commenting out that line in the `Makefile`, and seeing if the rest of the suite can be compiled correctly.

(4) To test the basic AMBER distribution, do this:

```
cd $AMBERHOME/test
make test
```

You can also try "make test.<program-name>", to test individual programs. See $AMBERHOME/test/Makefile for the available options.

Where "possible FAILURE" messages are found, go to the indicated directory under $AMBERHOME/test, and look at the "*.dif" files. Differences should involve round-off in the final digit printed, or occasional messages that differ from machine to machine (see below for details). As with compilation, if you have trouble with individual tests, you may wish to invoke "make" with the ignore errors option (`make -i`) or comment out certain lines in the Makefile, and/or go directly to the $AMBERHOME/test subdirectories to examine the inputs and outputs in detail.

(5) To build a parallel version, do the following:

```
cd $AMBERHOME/src
make clean
./configure -mpi ifort        (as an example)
make parallel
```

To test parallel programs, you need first to set the DO_PARALLEL environment variable as follows:

```
cd $AMBERHOME/test
setenv DO_PARALLEL 'mpirun -np 4'
make test.parallel
```

The integer is the number of processors; if your command to run MPI jobs is something different than `mpirun` (e.g. it is `dmpirun` on Tru64 Unix systems), use the command appropriate for your machine.

(6) If you are planning to run periodic, explicit solvent simulations using the paricle-mesh Ewald (PME) method, you will probably want to compile the *pmemd* program. See Chapter 6 for information on this.

1.3.1. More information on parallel machines or clusters

This section contains notes about the various parallel implementations supplied in the current release. Only *sander* and *pmemd* are parallel programs; all others are single threaded. NOTE: Parallel machines and networks fail in unexpected ways. PLEASE check short parallel runs against a single-processor version of Amber before embarking on long parallel simulations!

The MPI (message passing) version was initially developed by James Vincent and Ken Merz, based on 4.0 and later an early prerelease 4.1 version [18]. This version was optimized, integrated and extended by James Vincent, Dave Case, Tom Cheatham, Scott Brozell, and Mike Crowley, with input from Thomas Huber, Asiri Nanyakkar, and Nathalie Godbout.

The bonds, angles, dihedrals, SHAKE (only on bonds involving hydrogen), nonbonded energies and forces, pairlist creation, and integration steps are parallelized. The code is pure SPMD (single program multiple data) using a master/slave, replicated data model. Basically, the master node does all of the initial set-up and performs all the I/O. Depending on the version and/or what particular input options are chosen, either all the non-master nodes execute *force()* in parallel, or all nodes do both the forces and the dynamics in parallel. Communication is done to accumulate partial forces, update coordinates, etc.

The MPI source code is generally wrapped with C preprocessor, `cpp`, directives:

```
#ifdef MPI

    ...parallel sections with calls to MPI library routines...

#endif
```

If you plan on running with an MPI version and there is no pre-made configuration file, then you will need to modify the `config.h` file as follows:

```
(1) Add '-DMPI' to the FPPFLAGS variable.

(2) Add the path for include file for the (implementation supplied)
    mpif.h file to the FPPFLAGS variable; for example:

    FPPFLAGS="-DMPI -I/usr/local/src/mpi/include"

(3) Reference any necessary MPI libraries in the LOADLIB variable.
```

For reasons we don't understand, some MPI implementations require a null file for stdin, even though *sander* doesn't take any input from there. This is true for some SGI and HP machines. If you have troubles getting going, try the following:

```
mpirun -np <num-proc> sander  [ options ]   < /dev/null
```

1.3.2. Installing Non-Standard Features

The source files of some Amber programs contain multiple code paths. These code paths are guarded by directives to the C preprocessor. All Amber programs regardless of source language use the C preprocessor. The activation of non-standard features in guarded code paths can be controlled at build time via the -D preprocessor option. For example, to enable the use of a Lennard-Jones 10-12 potential with the *sander* program the HAS_10_12 preprocessor guard must be activated with -DHAS_10_12.

To ease the installers burden we provide a hook into the build process. The hook is the environment variable `AMBERBUILDFLAGS`. For example, to build *sander* with -DHAS_10_12, assuming that a correct configuration file has already been created, do the following:

```
cd $AMBERHOME/src/sander
make clean
make AMBERBUILDFLAGS='-DHAS_10_12' sander
```

Note that AMBERBUILDFLAGS is accessed by all stages of the build process: preprocessing, compiling, and linking. In rare cases a stage may emit warnings for unknown options in AMBERBUILDFLAGS; these may usually be ignored.

1.3.3. Installing on Windows

All of Amber (including the X-windows parts) will compile and run on Windows using the Cygwin development tools: see *http://sources.redhat.com/cygwin*. You will need to obtain a Fortran 90 compiler because one is not distributed with cygwin. Compilers from Absoft and Lahey have been tested successfully, but other f90 compilers ought to work as well.

Some f90 compilers (such as Absoft) won't work with ptraj, giving errors at the load stage. If this happens to you, compile everything else, then reconfigure with g77 as the compiler, and compile ptraj in that environment.

Note that Cygwin provides a POSIX-compatible environment for Windows. Effective use of this environment requires a basic familiarity with the principles of Linux or Unix operating systems. Building the Windows version is thus somewhat more complex (not simpler) than building under other operating systems. The Windows version has only been tested in a single-cpu environment.

1.3.4. Testing

We have installed and tested AMBER 8 on a number of platforms, using UNIX machines from IBM, Sun, Hewlett-Packard, DEC(Compaq), and Silicon Graphics, and on Red Hat Linux and Windows 95/98/NT/2000 (running on Intel Pentium and Itanium machines). However, owing to time and access limitations, not all combinations of code, compilers, and operating systems have been tested. Therefore we recommend running the test suites.

The distribution contains a validation suite that can be used to help verify correctness. The nature of molecular dynamics, is such that the course of the calculation is very dependent on the order of arithmetical operations and the machine arithmetic implementation, *i.e.* the method used for roundoff. Because each step of the calculation depends on the results of the previous step, the slightest difference will eventually lead to a divergence in trajectories. As an initially

identical dynamics run progresses on two different machines, the trajectories will eventually become completely uncorrelated. Neither of them are "wrong;" they are just exploring different regions of phase space. Hence, states at the end of long simulations are not very useful for verifying correctness. Averages are meaningful, provided that normal statistical fluctuations are taken into account. "Different machines" in this context means any difference in floating point hardware, word size, or rounding modes, as well as any differences in compilers or libraries. Differences in the order of arithmetic operations will affect roundoff behavior; (a + b) + c is not necessarily the same as a + (b + c). Different optimization levels will affect operation order, and may therefore affect the course of the calculations.

All initial values reported as integers should be identical. The energies and temperatures on the first cycle should be identical. The RMS and MAX gradients reported in sander are often more precision sensitive than the energies, and may vary by 1 in the last figure on some machines. In minimization and dynamics calculations, it is not unusual to see small divergences in behavior after as little as 100-200 cycles.

1.3.5. Memory Requirements

The AMBER 8 programs mainly use dynamic memory allocation, and do not need to be compiled for any specific size of problem. Some sizes related to NMR refinements are defined in `nmr.h`, and some dimensioning information for QM/MM calculations is in `cp.h`. If you receive error messages directing you to look at these files, you may need to edit them, then recompile.

If you get a "Segmentation fault" immediately upon starting a program (particularly if this happens with no arguments), you may not have enough memory to run the program. The Unix "size" command will show you the size of the executable. Also check the limits of your shell; you may need to increase these (especially stacksize, which is sometimes set to quite small values).

1.3.6. Notes for users of previous versions of Amber

The comments here are not exhaustive; they just point out some features and compatibility changes that may be affect those who have used previous versions of this package.

(1) In LEaP, the way perturbed charges are read in (in the *pertCharge* variable) is different from earlier versions of Amber. See the discussion in Section 3.2.3.5. In particular, if you have "off" files from Amber 7 that use the perturbation scheme, these will have to be modified for Amber 8. Perturbation *prmtop* files from previous versions do not need to be re-made.

(2) In sander, the syntax for specifying the atoms involved when *ntr=1* or *ibelly=1* has changed; the old syntax will continue to work, but will almost certainly be removed in future versions.

(3) In sander, the default value for *taup* is now more reasonable (1 ps *vs.* 0.2 ps).

(4) The *igb=3,4* options, for alternative generalized Born implementations, have been removed.

(5) The *sander, nmode, pmemd, ambpdb* and *pbsa* programs use Fortran 90, and hence require an f90 compiler. This change will primarily affect those who used g77 to compile earlier versions of Amber.

(6) The *gibbs*, *anal* and *carnal* programs have been retired, in the sense that we are no longer developing them. Source and documentation is still here for users who still want these programs. Most of the functionality has been put into *sander* and *ptraj*.

(7) There is no separate *roar* program; many of the QM/MM features of *roar* are now incorporated in *sander*.

(8) The *nmanal* and *lmanal* programs have been retired; this functionality is now in *ptraj*.

(9) We now provide our own semiempirical program, *divcon*; it is no longer necessary to obtain *mopac* in order to use antechamber.

(10) We now provide our own finite-difference Poisson-Boltzmann solver; it is no longer necessary to obtain an external program like *MEAD* or *delphi* in order to run mm_pbsa calculations.

(11) Appendix F contains the names of retired namelist variables.

1.4. Basic tutorials

AMBER is a suite of programs for use in molecular modeling and molecular simulations. It consists of a substructure database, a force field parameter file, and a variety of useful programs. Here we give some commented sample runs to provide an overview of how things are carried out. The examples only cover a fraction of the things that it is possible to do with AMBER. The formats of the example files shown are described in detail later in the manual, in the chapters pertaining to the programs. Tom Cheatham, Bernie Brooks and Peter Kollman have prepared some detailed information on simulation protocols that should be also be consulted [19].

Additional tutorial examples are available at *http://amber.scripps.edu*. Because the web can provide a richer interface than one can get on the printed page (with screen shots, links to the actual input and output files, etc.), most of our recent efforts have been devoted to updating the tutorials on the web site. In particular, new users are advised to look at the following, which can be found at both the web site listed above, and on the distribution CD, under *amber8/tutorial*:

DNA	Basic introduction to *LEaP*, *sander*, and *ptraj*, to build, solvate, run MD and analyze trajectories.
Plastocyanin/ion/water	Basic tutorial for a protein, introducing non-standard residues, NMR restraints, and more complex modeling tasks.
Loop dynamics in HIV integrase	Show how a study of protein dynamical behavior was carried out, illustrating some more complex setups and analyses.
NMR refinement of DNA	Basic introduction to NMR refinement using LEaP and sander.
GB simulation	Carrying out a protein simulation using the generalized Born continuum solvent model.

We are in the process of creating additional tutorials; because of the lead time needed to print this manual, there may be new tutorials available, either in *amber8/tutorial* or at the web site listed

above. You should also look at the sample inputs in the chapters devoted to each program, especially for LEaP and sander.

As a basic example, we consider here the minimization of a protein in a simple solvent model. The procedure consists of three steps:

Step 1. Generate some starting coordinates.

The first step is to obtain starting coordinates. We begin with the bovine pancreatic trypsin inhibitor, and consider the file *6pti.pdb*, exactly as distributed by the Protein Data Bank. This file (as with most PDB files) needs some editing before it can be used by Amber. First, alternate conformations are provided for residues 39 and 50, so we need to figure out which one we want. For this example, we choose the "A" conformation, and manually edit the file to remove the alternate conformers. Second, coordinates are provided for a phosphate group and a variety of water molecules. These are not needed for the calculation we are pursuing here, so we also edit the file to remove these. Third, the cysteine residues are involved in disulfide bonds, and need to have their residue names changed in an editor from CYS to CYX to reflect this. Finally, since we removed the phosphate groups, some of the CONECT records now refer to non-existent atoms; if you are not sure that the CONECT records are all correct then it may be safest to remove all of them, as we do for this example. Let's call this modified file *6pti.mod.pdb.*

Although Amber tries hard to understand pdb-format files, it is typical to have to do some manual editing before proceeding. A general prescription is: "keep running the *loadPdb* step in LEaP (see step 2, below), and editing the pdb file, until there are no error messages."

Step 2. Run LEaP to generate the parameter and topology file.

This is a fairly straightforward exercise in loading in the pdb file, adding the disulfide cross links, and saving the resulting files. Typing the following commands should work in either *tleap* or *xleap*:

```
source leaprc.ff94
bpti = loadPdb 6pti.mod.pdb
bond bpti.5.SG bpti.55.SG
bond bpti.14.SG bpti.38.SG
bond bpti.30.SG bpti.51.SG
saveAmberParm bpti prmtop prmcrd
quit
```

Step 3. Perform some minimization.

Use this script:

Running minimization for BPTI

```
cat << eof > min.in
#  200 steps of minimization, generalized Born solvent model
 &cntrl
    maxcyc=200, imin=1, cut=12.0, igb=1, ntb=0, ntpr=10,
 /
eof
sander -i min.in -o 6pti.min1.out -c prmcrd -r 6pti.min1.xyz
/bin/rm min.in
```

This will perform minimization (`imin=1`) for 200 steps (`maxcyc`), using a nonbonded cut-off of 12 Å (`cut`), a generalized Born solvent model (`igb=1`), and no periodic boundary (`ntb=0`); intermediate results will be printed every 10 steps (`ntpr`). Text output will go to file *6pti.min1.out*, and the final coordinates to file *6pti.min1.xyz*. The "out" file is intended to be read by humans, and gives a summary of the input parameters and a history of the progress of the minimization.

Of course, Amber can do much more than the above minimization. This example illustrates the basic information flow in Amber: Cartesian coordinate preparation (*Step 1.*), topology and force field selection (*Step 2.*), and simulation program command specification (*Step 3.*). Typically the subsequent steps are several stages of equilibration, production molecular dynamics runs, and analyses of trajectories. The tutorials in *amber8/tutorial* should be consulted for examples of these latter steps.

2. Specifying a force field

Amber is designed to work with several simple types of force field, although it is most commonly used with parameterizations developed by Peter Kollman and his co-workers. There are now a variety of such parameterizations, with no obvious "default" value. The "traditional" parameterization uses fixed partial charges, centered on atoms. Examples of this are *ff94*, *ff99* and *ff03* (described below). The default in versions 5 and 6 of Amber was *ff94*; a comparable default now would probably be *ff03*, but users should consult the papers listed below to see a detailed discussion of the changes made.

Less extensively used, but very promising, recent modifications add polarizable dipoles to atoms, so that the charge description depends upon the environment; such potentials are called "polarizable" or "non-additive". Examples are *ff02* and *ff02EP*: the former has atom-based charges (as in the traditional parameterization), and the latter adds in off-center charges (or "extra points"), primarily to help describe better the angular dependence of hydrogen bonds. Again, users should consult the papers cited below to see details of how these new force fields have been developed.

In order to tell LEaP which force field is being used, the four types of information described below need to be provided. This is generally accomplished by selecting an appropriate *leaprc* file, which loads the information needed for a specific force field. (See section 2.2, below).

(1) A listing of the atom types, what elements they correspond to, and their hybridizations. This information is encoded as a set of LEaP commands, and is normally read from a *leaprc* file.

(2) Residue descriptions (or "topologies") that describe the chemical nature of amino acids, nucleotides, and so on. These files specify the connectivities, atom types, charges, and other information. These files have a "prep" format (a now-obsolete part of Amber) and have a ".in" extension. Standard libraries of residue descriptions are in the *amber8/dat/leap/prep* directory. The *antechamber* program may be used to generate prep files for other organic molecules.

(3) Parameter files give force constants, equilibrium bond lengths and angles, Lennard-Jones parameters, and the like. Standard files have a ".dat" extension, and are found in *amber8/dat/leap/parm*.

(4) Extensions or changes to the parameters can be included in *frcmod* files. The expectation is that the user will load a large, "standard" parameter file, and (if needed) a smaller frcmod file that keeps track of any changes to the default parameters that are needed. The *frcmod* files for changing the default water model (which is TIP3P) into other water models are in files like *amber8/dat/leap/parm/frcmod.tip4p*. The *parmchk* program (part of Antechamber) can also generate *frcmod* files.

2.1. Description of the database files

The following files are in the *amber8/dat/leap* directory. Files with a ".in" extension are in the *prep* subdirectory; those with a ".dat" extension are in the *parm* subdirectory, as are the "frcmod" files; files ending with ".off" or ."lib" are in the *lib* subdirectory.

Glycam 2004 (Woods et al.) force field

```
glycam04.dat        Parameters for oligosaccharides
glycam04EP.dat      Parameters for oligosaccharides, using extra points

glycam04.in         Topologies for glycosyl residues
glycam04EP.in       Topologies for glycosyl residues, using extra points
```

Amber 2003 (Duan et al.) force field

```
frcmod.ff03         For proteins: changes to parm99.dat, primarily in the
                    phi and psi torsions.
all_amino03.in      Charges and atom types for proteins.
```

Amber 1999 and 2002 force fields

```
parm99.dat          Force field, for amino acids and some organic molecules;
                       can be used with either additive or
                       non-additive treatment of electrostatics.
parm99EP.dat        Like parm99.dat, but with "extra-points": off-center
                       atomic charges, somewhat like lone-pairs.
gaff.dat            Force field for general organic molecules.
frcmod.mod_phipsi.1  Modified phi and psi torsions, described below.
frcmod.mod_phipsi.2  Modified phi and psi torsions, described below.

all_nuc02.in        Nucleic acid input for building database, for a non-
                    additive (polarizable) force field without extra points.
all_amino02.in      Amino acid input ...
all_aminoct02.in    COO- amino acid input ...
all_aminont02.in    NH3+ amino acid input ....

all_nuc02EP.in      Nucleic acid input for building database, for a non-
                    additive (polarizable) force field with extra points.
all_amino02EP.in      Amino acid input ...
all_aminoct02EP.in    COO- amino acid input ...
all_aminont02EP.in    NH3+ amino acid input ....
```

Amber 1994 (Cornell et al.) force field

```
all_nuc94.in        Nucleic acid input for building database.
all_amino94.in      Amino acid input for building database.
all_aminoct94.in    COO- amino acid input for database.
all_aminont94.in    NH3+ amino acid input for database.
nacl.in             Ion file.
```

```
parm94.dat          1994 force field file.
parm96.dat          Modified version of 1994 force field, for proteins.
parm98.dat          Modified version of 1994 force field, for nucleic acids.
```

Amber 1984, 1986 (Weiner et al.) force fields

```
all.in              All atom database input.
allct.in            All atom database input, COO- Amino acids.
allnt.in            All atom database input, NH3+ Amino acids.
uni.in              United atom database input.
unict.in            United atom database input, COO- Amino acids.
unint.in            United atom database input, NH3+ Amino acids.
parm91X.dat         Parameters for 1984, 1986 force fields.
```

Solvent models:

```
water.in            Topology definition for several water models.
meoh.in             Topology file for methanol.
chcl3.in            Topology file for chloroform.
nma.in              Topology file for N-methylacetamide.

tip3pbox.off        Solvation box for TIP3P water.
tip4pbox.off        Solvation box for TIP4P water.
pol3box.off         Solvation box for POL3 water.
spcebox.off         Solvation box for SPC/E water.
meohbox.off         Solvation box for methanol.
nmabox.off          Solvation box for N-methylacetamide.
chcl3box.off        Solvation box for chloroform.
8Mureabox.off       Solvation box for 8M urea/water mixture
                      (see 8Murea.readme for more information).

frcmod.tip4p        Parameter changes from TIP3P -> TIP4P.
frcmod.tip5p        Parameter changes from TIP3P -> TIP5P.
frcmod.spce         Parameter changes from TIP3P -> SPC/E.
frcmod.pol3         Parameter changes from TIP3P -> POL3.
frcmod.meoh         Parameters for methanol.
frcmod.chcl3        Parameters for chloroform.
frcmod.nma          Parameters for N-methyacetamide.
frcmod.urea         Parameters for urea (or urea-water mixtures).
```

Miscellaneous:

```
nucgen.dat          Nucgen nucleic acid conformations.
PROTON_INFO*        Files needed for protonate
map.DG-AMBER        Needed for NMR input generation.
```

2.2. Specifying which force field you want in LEaP

Various combinations of the above files make sense, and we have moved to an "ff" (force field) nomenclature to identify these; examples would then be *ff94* (which was the default in Amber 5 and 6), *ff99*, etc. The most straightforward way to specify which force field you want is to use one of the *leaprc* files in *$AMBERHOME/dat/leap/cmd*. The sytax is:

```
xleap -s -f <filename>
```

Here, the *-s* flag tells LEaP to ignore any *leaprc* file it might find, and the *-f* flag tells it to start with commands for some other file. Here are the combinations we support and recommend:

How to specify force fields in LEaP

filename	*topology*	*parameters*
leaprc.ff86	Weiner *et al.* 1986	parm91X.dat
leaprc.ff94	Cornell *et al.* 1994	parm94.dat
leaprc.ff96	"	parm96.dat
leaprc.ff98	"	parm98.dat
leaprc.ff99	"	parm99.dat
leaprc.ff03	Duan *et al.* 2003	parm99.dat+frcmod.ff03
leaprc.ff02	reduced (polarizable) charges	parm99.dat
leaprc.ff02EP	" + extra points	parm99EP.dat
leaprc.gaff	none	gaff.dat
leaprc.glycam04	Woods *et al.*	glycam04.dat
leaprc.glycam04EP	"	glycam04EP.dat

Notes:

(1) There is no default leaprc file. If you make a link from one of the files above to a file named `leaprc`, then that will become the default. For example:

```
cd $AMBERHOME/dat/leap/cmd
ln -s leaprc.ff03 leaprc
```

will provide a good default for many users; after this you could just invoke `tleap` or `xleap` without any arguments, and it would automatically load the *ff03* force field. If you put *leaprc.ff94* in the above link command, you would be making the Cornell *et al.* force field the default, which was the behavior of versions 5 and 6 of Amber. Note also that a *leaprc* file in the current directory overrides any other such files that might be present in the search path.

(2) The first six choices in the above table are for additive (non-polarizable) simulations; you should use saveAmberParm (or saveAmberParmPert) to save the *prmtop* file, and keep the default *ipol=0* in sander or gibbs.

(3) The *ff02* entries in the above table are for non-additive (polarizable) force fields. Use saveAmberParmPol to save the *prmtop* file, and set *ipol=1* in the sander or gibbs input file. Note that POL3 is a polarizable water model, so you need to use saveAmberParmPol for it as well.

(4) The files above assume that nucleic acids are DNA, if not explicitly specified. Use the files *leaprc.rna.ff98, leaprc.rna.ff99, leaprc.rna.ff02* or *leaprc.rna.ff00EP* to make the default RNA. If you have a mixture of DNA and RNA, you will need to edit your PDB file, or use the `loadPdbUsingSequence` command in LEaP (see that chapter) in order to specify which nucelotide is which.

(5) There is also a *leaprc.gaff* file, which sets you up for the "general" Amber force field. This is primarily for use with Antechamber (see that chapter), and does not load any topology files.

(6) The *leaprc.ff86* files gives the 1986 all-atom parameters; Amber no longer directly supports the 1984 united atom parameter set.

(7) Our experience with generalized Born simulations is mainly with *ff99* or *ff03*; the current GB models are not compatible with polarizable force fields. Replacing explicit water with a GB model is equivalent to specifying a different force field, and users should be aware that none of the GB options (in Amber or elsewhere) is as "mature" as simulations with explicit solvent; user discretion is advised!

2.3. The Duan et al. (2003) force field

The **ff03** force field [3] is a modified version of *ff99* (described below). The main changes are that charges are now derived from quantum calculations that use a continuum dielectric to mimic solvent polarization, and that the ϕ and ψ backbone torsions for proteins are modified, with the effect of decreasing the preference for helical configurations. The changes are just for proteins; nucleic acid parameters are the same as in *ff99*.

2.4. 1999 and 2002 force fields

The **ff99** force field [20] points toward a common force field for proteins for "general" organic and bioorganic systems. The atom types are mostly those of Cornell *et al.* (see below), but changes have been made in many torsional parameters, and this parameterization supports both additive and non-additive (polarizable) force fields. The topology and coordinate files for the small molecule test cases used in the development of this force field are in the *parm99_lib* subdirectory. The *ff99* force field uses these parameters, along with the topologies and charges from the Cornell *et al.* force field, to create an all-atom nonpolarizable force field for proteins and nucleic acids.

Several groups have noticed that *ff99* (and *ff94* as well) don't provide a good energy balance between helical and extended regions of peptide and protein backbones. The *frcmod.mod_phipsi.1* file (which you should load after loading *parm99.dat* or *parm94.dat*) is one attempt to improve this behavior, and was developed in the Simmerling group [21]. A second attempt is the *frcmod.mod_phipsi.2* file developed by Junmei Wang and Ray Luo (manuscript in preparation). A third alternative is to simply zero out the torsional terms for the ϕ and ψ backbone angles [22]. Research in this area is ongoing, and users interested in peptide and protein folding are urged to keep abreast of the current literature.

The **ff02** force field is a polarizable variant of *ff99*. Here, the charges were determined at the B3LYP/cc-pVTZ//HF/6-31G* level, and hence are more like "gas-phase" charges. During charge fitting the correction for intramolecular self polarization has been included [23]. Bond polarization arising from interactions with a condensed phase environment are achieved through polarizable dipoles attached to the atoms. These are determined from isotropic atomic

polarizabilities assigned to each atom, taken from experimental work of Applequist. The dipoles can either be determined at each step through an iterative scheme, or can be treated as additional dynamical variables, and propagated through dynamics along with the atomic positions, in a manner analogous Car-Parinello dynamics. Derivation of the polarizable force field required only minor changes in dihedral terms and a few modification of the van der Waals parameters.

The user also has a choice to use the polarizable force field with extra points on which additional point charges are located; this is called **ff02EP**. The additional points are located on electron donating atoms (e.g. O,N,S), which mimic the presence of electron lone pairs [24]. For nucleic acids we chose to use extra interacting points only on nucleic acid bases and not on sugars or phosphate groups.

There is not (yet) a full published description of this, but a good deal of preliminary work on small molecules is available [23,25]. Beyond small molecules, our initial tests have focussed on small proteins and double helical oligonucleotides, in additive TIP3P water solution. Such a simulation model, (using a polarizable solute in a non-polarizable solvent) gains some of the advantages of polarization at only a small extra cost, compared to a standard force field model. In particular, the polarizable force field appears better suited to reproduce intermolecular interactions and directionality of H-bonding in biological systems than the additive force field. Initial tests show *ff02EP* behaves slightly better than *ff02*, but it is not yet clear how significant or widespread these differences will be.

The **gaff.dat** ("general Amber force field") is yet a further step towards general purpose organic molecules [4]. It is primarily used in conjunction with the *antechamber* program, and users should consult that chapter for more information.

2.5. The Cornell et al. (1994) force field

Contained in **ff94** are parameters from the so-called "second generation" force field developed in the Kollman group in the early 1990's [26]. These parameters are especially derived for solvated systems, and when used with an appropriate 1-4 electrostatic scale factor, have been shown to perform well at modeling many organic molecules. The parameters in *parm94.dat* omit the hydrogen bonding terms of earlier force fields. This is an all-atom force field; no united-atom counterpart is provided. 1-4 electrostatic interactions are scaled by 1.2 instead of the value of 2.0 that had been used in earlier force fields.

Charges were derived using Hartree-Fock theory with the 6-31G* basis set, because this exaggerates the dipole moment of most residues by 10-20%. It thus "builds in" the amount of polarization which would be expected in aqueous solution. This is necessary for carrying out condensed phase simulations with an effective two-body force field which does not include explicit polarization. The charge-fitting procedure is described in Chapter 12.2.

The **ff96** force field [27] differs from *parm94.dat* in that the torsions for ϕ and ψ have been modified in response to *ab initio* calculations [28] which showed that the energy difference between conformations were quite different than calculated by Cornell *et al.* (using *parm94.dat*). To create parm96.dat, common V_1 and V_2 parameters were used for ϕ and ψ, which were empirically adjusted to reproduce the energy difference between extended and constrained alpha helical energies for the alanine tetrapeptide. This led to a significant improvement between molecular mechanical and quantum mechanical relative energies for the remaining members of the set of tetrapeptides studied by Beachy *et al.* Users should be aware that *parm96.dat* has not been as extensively used as *parm94.dat*, and that it almost certainly has its own biases and idiosyncrasies [29,30].

The **ff98** force field [31] differs from *parm94.dat* in torsion angle parameters involving the glycosidic torsion in nucleic acids. These serve to improve the predicted helical repeat and sugar pucker profiles.

2.6. The Weiner et al. (1984,1986) force fields

The **ff86** parameters are described in early papers from the Kollman and Case groups [32,33]. [The "parm91" designation is somewhat unfortunate: this file is really only a corrected version of the parameters described in the 1984 and 1986 papers listed above.] These parameters are not generally recommended any more, but may still be useful for vacuum simulations of nucleic acids and proteins using a distance-dependent dielectric, or for comparisons to earlier work. The material in *parm91X.dat* is the parameter set distributed with Amber 4.0. The *STUB* nonbonded set has been copied from *parmuni.dat*; these sets of parameters are appropriate for united atom calculations using the "larger" carbon radii referred to in the "note added in proof" of the 1984 JACS paper. If these values are used for a united atom calculation, the parameter *scnb* should be set to 8.0; for all-atom calculations use 2.0. The *scee* parameter should be set to 2.0 for both united atom and all-atom variants. *Note that the default value for scee is sander is now 1.2 (the value for 1994 and later force fields; users must explicitly change this in their inputs for the earlier force fields.*

parm91X.dat is not recommended. However, for historical completeness a number of terms in the non-bonded list of parm91X.dat should be noted. The non-bonded terms for I(iodine), CU(copper), and MG(magnesium) have not been carefully calibrated, but are given as approximate values. In the STUB set of non-bonded parameters, we have included parameters for a large hydrated monovalent cation (IP) that represent work by Singh *et al.* [34] on large hydrated counterions for DNA. Similar values are included for a hydrated anion (IM).

The non-bonded potentials for hydrogen-bond pairs in *ff86* use a Lennard-Jones 10-12 potential. If you want to run *sander* with *ff86* then you will need to recompile, adding -DHAS_10_12 to the Fortran preprocessor flags; see Chapter 1.3.2.

2.7. Glycam-04 force field for carbohydrates

As in previous versions of Glycam, the parameters are intended for explicitly solvated MD simulations of carbohydrates. Also, as in previous versions, the van der Waals parameters were borrowed from the Parm94 force field. However, in several other areas the development of the Glycam-04 parameters differs significantly from earlier versions of Glycam. This parameter set is unique from the other standard AMBER parameters in that is has been derived for use without the need for scaling 1-4 non-bonded interactions. Other major differences from earlier Glycam versions relate to the partial charge placements. Throughout the development of the parameters the 1-4 electrostatic (SCEE) and non-bonded (SCNB) scaling factors were set to unity. We have shown that this is essential in order to properly treat internal hydrogen bonds, particularly those associated with the hydroxymethyl group. With 1-4 scaling, it was not possible to correctly reproduce the rotamer populations for the C5-C6 bond. For studying carbohydrate-protein interactions, we suggest that the SCEE and SCNB scaling factors be set to the appropriate value according to the protein force field that is chosen. While this may degrade the accuracy of the rotational populations obtained with Glycam, it should not interfere with the stability or structure of protein-bound carbohydrates. Details of how the new parameters were developed are described elsewhere [5-7].

As in previous versions of Glycam, the atomic partial charges were determined using the RESP formalism, with a weighting factor of 0.01 from a wavefunction computed at the

HF/6-31G(d) level. However, to reduce artifactual fluctuations in the charges on saturated carbon atoms, charges on aliphatic hydrogens (types HC, H1, H2, & H3) were set to zero while the partial charges were fit to the remaining atoms. Due to the rotational freedom of hydroxyl groups, partial atomic charges for each sugar were determined by averaging charges obtained from 100 conformations selected evenly from 50 ns solvated MD simulations of the methyl glycoside of each monosaccharide, thus yielding an ensemble averaged charge set.

In order to extend glycam04 to simulations employing the TIP-5P water model, an additional set of carbohydrate prep files has been derived, in which lone pairs (or extra points, EPs) have been incorporated on the oxygen atoms. The optimal O-EP distance was located by obtaining the best fit to the HF/6-31g(d) electrostatic potential. In general, the best fit to the quantum potential coincided with a negligible charge on the oxygen nuclear position. The optimal O-EP distance for an sp3 oxygen atom was found to be 0.70Å; for an sp2 oxygen atom a shorter length of 0.45Å was optimal. When applied to water, this approach to locating the lone pair positions and assigning the partial charges yielded a model that was essentially indistinguishable from TIP-5P. Therefore, we believe this model is well suited for use with TIP-5P.

All valence parameters were determined by fitting to data computed at the B3LYP/6-31++G(2d,2p)//HF/6-31G(d) level of theory. This level was selected due to its inherently low level of basis set superposition error, which manifests itself in a reduction of rotational barrier heights. A major difference from earlier versions of Glycam is that here each parameter term was explicitly parameterized, using a molecular development suite of more than 75 molecules. The parameter test suite included carbohydrates and numerous smaller molecular fragments. Further, in this version of Glycam we have eliminated the atom types AC, EC, and OG. However, to avoid potential conflicts with the CT atom type in the protein parameters, we have employed a new atom type for tetrahedral carbons called CG (C-Glycam) in order to keep the Glycam parameters specific for carbohydrates.

The latest release of the Glycam parameters prep files can always be obtained from the Woods group at `http://glycam.ccrc.uga.edu`.

2.7.1. Notes on the naming of Prep files

Due to the structural diversity of carbohydrates, the prep file nomenclature requires some explanation. The naming of prep files is relatively straightforward. However, to be limited to a three-letter residue name requires some compromises in clarity. Nevertheless, an orthogonal set is presented that encodes the following details: core monosaccharide name (glucose, mannose, etc.), anomericity (α or β), configuration (D or L), sugar linkage position (2, 3, 4, etc.) and ring size (pyranose or furanose).

Here are some examples of pyranoses (linked at the 2-position):

-2)-α-D-mannose	prep file name: 2madp.prep	Residue name: 2MA (A = α)
-2)-β-D-mannose	prep file name: 2mbdp.prep	Residue name: 2MB (B = β)
-2)-α-L-mannose	prep file name: 2malp.prep	Residue name: M2A (mirror image of 2MA)
-2)-β-L-mannose	prep file name: 2mblp.prep	Residue name: M2B (mirror image of 2MA)

Here are some examples of furanoses (linked at the 2-position):

-2)-α-D-xylose	prep file name: 2xadf.prep	Residue name 2XD (D = α, as in Down)
-2)-β-D-xylose	prep file name: 2xbdf.prep	Residue name 2XU (U = β, as in Up)
-2)-α-L-xylose	prep file name: 2xalf.prep	Residue name X2D (mirror image of 2XD)
-2)-β-L-xylose	prep file name: 2xblf.prep	Residue name X2U (mirror image of 2XU)

Finally, here are some examples of terminal residues (thought of as linked at the 1-position):

α-D-mannopyranose-(1-	prep file name: 1madp.prep	Residue name: 1MA
α-D-xylofuranose-(1-	prep file name: 1xadf.prep	Residue name 1XD

2.7.2. Carbohydrate Naming Convention in Glycam-04

Here are the one-letter codes that form the core of the Glycam residue names for carbohydrates. (In the future we may employ lowercase letters to define L-sugars. There are also numerous complex residues that will not fit the rule for simple monosaccharides.)

One-letter codes for carbohydrates			
Carbohydrate	*One-letter*	*Common Abbr.*	*Classification*
Arabinose	A	Ara	Pentose
Lyxose	D	Lyx	Pentose
Ribose	R	Rib	Pentose
Xylose	X	Xyl	Pentose
Allose	N	All	Hexose
Altrose	E	Alt	Hexose
Galactose	L	Gal	Hexose
Glucose	G	Glc	Hexose
Gulose	K	Gul	Hexose
Idose	I	Gul	Hexose
Mannose	M	Man	Hexose
Talose	T	Tal	Hexose
Fructose	C	Fru	Hexulose
Psicose	P	Psi	Hexulose
Sorbose	B	Sor	Hexulose
Tagatose	J	Tag	Hexulose
Fucose	F	Fuc	6-DeoxyHexose
Quinovose	Q	Qui	6-DeoxyHexose
Rhamnose	H	Rha	6-DeoxyHexose
Galacturonic Acid	O	GalA	Hexuronic Acid
Glucuronic Acid	Z	GlcA	Hexuronic Acid
Iduronic Acida	U	IdoA	Hexuronic Acid
N-Acetylgalactosamine	V	GalNac	HexNAc
N-Acetylglucosamine	Y	GlcNAc	HexNAc
N-Acetylmannosamine	W	ManNAc	HexNAc
N-Acetyl-neuraminic acid	S	NeuNAc, Neu5Ac	9-Carbon Acid
KDN	KN	KDN	8-Carbon Acid
KDO	KO	KDO	8-Carbon Acid
N-Glycolyl-neuraminic acid	SG	NeuNGc, Neu5Gc	9-Carbon Acid

The following tables give details of choosing residue names.

Linkage position and anomeric configuration in D-hexopyranoses				
	α-D-Glucose		*β-D-Mannose*	
Linkage Position	*Prep File Prefix*	*Residue Name*	*Prep File Prefix*	*Residue Name*
Terminal	1GADP	1GA	1MBDP	1MB
2-	2GADP	2GA	2MBDP	2MB
3-	3GADP	3GA	3MBDP	3MB
4-	4GADP	4GA	4MBDP	4MB
6-	6GADP	6GA	6MBDP	6MB
2,3-	23GADP	ZGA	23MBDP	ZMB
2,4-	24GADP	YGA	24MBDP	YMB
2,6-	26GADP	XGA	26MBDP	XMB
3,4-	34GADP	WGA	34MBDP	WMB
3,6-	36GADP	VGA	36MBDP	VMB
4,6-	46GADP	UGA	46MBDP	UMB
2,3,4-	234GADP	TGA	234MBDP	TMB
2,3,6-	236GADP	SGA	236MBDP	SMB
2,4,6-	246GADP	RGA	246MBDP	RMB
3,4,6-	346GADP	QGA	346MBDP	QMB
2,3,4,6-	2346GADP	PGA	2346MBDP	PMB

Linkage position and anomeric configuration in L-hexopyranoses				
	α-L-Glucose		*β-L-Mannose*	
Linkage Position	*Prep File Prefix*	*Residue Name*	*Prep File Prefix*	*Residue Name*
Terminal	1GALP	G1A	1MBLP	M1B
2-	2GALP	G2A	2MBLP	M2B
3-	3GALP	G3A	3MBLP	M3B
4-	4GALP	G4A	4MBLP	M4B
6-	6GALP	G6A	6MBLP	M6B
2,3-	23GALP	GZA	23MBLP	MZB
2,4-	24GALP	GYA	24MBLP	MYB
2,6-	26GALP	GXA	26MBLP	MXB
3,4-	34GALP	GWA	34MBLP	MWB
3,6-	36GALP	GVA	36MBLP	MVB
4,6-	46GALP	GUA	46MBLP	MUB
2,3,4-	234GALP	GTA	234MBLP	MTB
2,3,6-	236GALP	GSA	236MBLP	MSB
2,4,6-	246GALP	GRA	246MBLP	MRB
3,4,6-	346GALP	GQA	346MBLP	MQB
2,3,4,6-	2346GALP	GPA	2346MBLP	MPB

A = α, B = β; The D-configuration (mirror image of the L-) is indicated in the three letter residue name by reversing the first two letters.

Linkage position and anomeric configuration in D-pentofuranoses				
	α-D-Arabinose		*β-D-Xylose*	
Linkage Position	*Prep File Prefix*	*Residue Name*	*Prep File Prefix*	*Residue Name*
Terminal	1AADF	1AD	1XBDF	1XU
2-	2AADF	2AD	2XBDF	2XU
3-	3AADF	3AD	3XBDF	3XU
4-	4AADF	4AD	4XBDF	4XU
2,3-	23AADF	ZAD	23XBDF	ZXU
2,4-	24AADF	YAD	24XBDF	YXU
3,4-	34AADF	WAD	34XBDF	WXU
2,3,4-	234AADF	TAD	234XBDF	TXU

Linkage position and anomeric configuration in L-pentofuranoses				
	α-L-Arabinose		*β-L-Xylose*	
Linkage Position	*Prep File Prefix*	*Residue Name*	*Prep File Prefix*	*Residue Name*
Terminal	1AALF	A1D	1XBLF	X1U
2-	2AALF	A2D	2XBLF	X2U
3-	3AALF	A3D	3XBLF	X3U
4-	4AALF	A4D	4XBLF	X4U
2,3-	23AALF	AZD	23XBLF	XZU
2,4-	24AALF	AYD	24XBLF	XYU
3,4-	34AALF	AWD	34XBLF	XWU
2,3,4-	234AALF	ATD	234XBLF	XTU

Linkage position in exceptional cases			
Carbohydrate	*Linkage Position*	*Prep File Prefix*	*Residue Name*
α-D-Sialic Acid	Terminal	1SA	1SA
	7-	7SA	7SA
	8-	8SA	8SA
	9-	9SA	9SA
α-D-N-Acetylglucosamine	Terminal	YA	YA
	3-	3YA	3YA
	4-	4YA	4YA
	6-	6YA	6YA
	3,4-	34YA	WYAa
	3,6-	36YA	VYA
	4,6-	46YA	UYA
	3,4,6-	346YA	RYA

(The single letter specification of linkage position follows the pattern of the earlier tables.)

2.7.3. Building a polysaccharaide in LEaP

Here is an example of how to construct a somewhat complex polysaccharide, using the *glycam04* parameters:

```
logFile leap.log
#
#
#          Making Branched Carbohydrates
#
# In general, the head atom is set to C1 in each sugar, where the tail atom
# varies depending on the sugar linkage.  For all sugars, the tail atom is
# specified as the glycosidic oxygen with the largest atomid, i.e. for the
# 2,3,4, linked sugars the tail atoms would be O4.  Therefore, it it possible
# to build the longest chain according to this convention and then connect
# the branched portions.  Simply changing the tail atom enables a different
# connection point for the unit, which can then be utilized to sequence to
# any branched residue/s.
#
#
# Example: Man9
#
# Mana1 - 2Mana1 \
#                   3
#                    Mana1
#                   6      \
# Mana1 - 2Mana1 /           6
#                             ManB-OMe
#                            3
# Mana1 - 2Mana1 - 2Mana1 /
#
#
# First, we must load the necessary parameters and prep files
#
source leaprc.glycam04EP
#
# Find the longest chain and use the sequence command to build it
# (See the manual for naming conventions)
#
part1 = sequence { OME VMB VMA 2MA 1MA }
#
# Now set the tail atom of part1 to O3 of VMB
#
set part1 tail part1.2.16
#
# Join the first branch to the long chain
#
part2 = sequence { part1 2MA 2MA 1MA }
#
# Set the tail atom of part2 to O3 of VMA
```

```
#
set part2 tail part2.3.11
#
# Add the last branch
#
man9 = sequence { part2 2MA 1MA }
#
# The basic structure has been built, but it clearly does not have the optimal
# glycosidic torsion angles.
#
# Set the psi(1-6) torsion to 180 (C1-O6-C6-C5)
#
impose man9 { 2 3 4 } { { C5 C6 O6 C1 180.0 } }
#         CHECK TORSIONS
measureGeom man9.4.1 man9.3.28 man9.3.25 man9.3.20
measureGeom man9.3.1 man9.2.28 man9.2.25 man9.2.6
#
# Set the omega angle of 6-linked sugars to 60.0 (O5-C5-C6-O6)
#
impose man9 { 2 3 }  {{ O6 C6 C5 O5 60.0 } }
#         CHECK TORSIONS
measureGeom man9.2.3 man9.2.6 man9.2.25 man9.2.28
measureGeom man9.3.22 man9.3.20 man9.3.25 man9.3.28
#
# Set the phi angle of all a-linked (not 1-6 linked) sugars to -60.0
#
impose man9 { 4 5 6 7 8 9 10 } { { H1 C1 O2 C2 -60.0 } }
impose man9 { 6 2 } { { H1 C1 O3 C3 -60.0 } }
impose man9 { 9 3 } { { H1 C1 O3 C3 -60.0 } }
#         CHECK TORSIONS
measureGeom man9.5.2 man9.5.1 man9.4.29 man9.4.3
measureGeom man9.6.2 man9.6.1 man9.2.16 man9.2.14
measureGeom man9.7.2 man9.7.1 man9.6.29 man9.6.3
measureGeom man9.8.2 man9.8.1 man9.7.29 man9.7.3
measureGeom man9.9.2 man9.9.1 man9.3.11 man9.3.9
measureGeom man9.10.2 man9.10.1 man9.9.29 man9.9.3
#
# Set the psi angle of all sugars to 0.0
#
impose man9 { 4 5 6 7 8 9 10 } { { C1 O2 C2 H2 0.0 } }
impose man9 { 6 2 } { { C1 O3 C3 H3 0.0 } }
impose man9 { 9 3 } { { C1 O3 C3 H3 0.0 } }
#         CHECK TORSIONS
measureGeom man9.5.1 man9.4.29 man9.4.3 man9.4.4
measureGeom man9.6.1 man9.1.16 man9.1.14 man9.1.15
measureGeom man9.7.1 man9.6.29 man9.6.3 man9.6.4
measureGeom man9.8.1 man9.7.29 man9.7.3 man9.7.4
measureGeom man9.9.1 man9.3.11 man9.3.9 man9.3.10
measureGeom man9.10.1 man9.9.29 man9.9.3 man9.9.4
```

```
#
# Now we have a reasonable starting structure for AMBER min/md, but we need
# to save a topology and coordinate file as well as a pdb for later use.
#
saveamberparm man9 man9.top man9.crd
savepdb man9 man9.pdb
#
# Exit the program
#
quit
#
# For more info visit the Glycam website www.glycam.ccrc.uga.edu
# February 22, 2004 - Tschampel
#
```

2.8. Ions

For alkali ions with TIP3P waters, we have provided the values of Åqvist [35], which are adjusted for Amber's nonbonded atom pair combining rules to give the same ion-OW potentials as in the original (which were designed for SPC water); these values reproduce the first peak of the radial distribution for ion-OW and the relative free energies of solvation in water of the various ions. Note that these values would have to be changed if a water model other than TIP3P were to be used. These potentials may also need modification if absolute free energies of solvation are important [36].

2.9. Solvent models

Amber now provides direct support for several water models. The default water model is TIP3P [37]. This model will be used for residues with names HOH or WAT. If you want to use other water models, execute the following leap commands after loading your leaprc file:

```
WAT = PL3                       (residues named WAT in pdb file will be POL3)
loadAmberParams frcmod.pol3     (sets the HW,OW parameters to POL3)
```

(The above is obviously for the POL3 model.) The *solvents.lib* file contains TIP3P [37], TIP4P [37,38], TIP5P [39], POL3 [40] and SPC/E [41] models for water; these are called TP3, T4P, T5P, PL3 and SPC, respectively. By default, the residue name in the *prmtop* file will be WAT, regardless of which water model is used. If you want to change this (for example, to keep track of which water model you are using), you can change the residue name to whatever you like. For example,

```
WAT = TP4
set WAT.1 name "TP4"
```

would make a special label in PDB and *prtmop* files for TIP4P water. Note that Brookhaven format files allow at most three characters for the residue label.

In addition, non-polarizable models for the organic solvents methanol, chloroform and N-methylacetamide are provided, along with a box for an 8M urea-water mixture. The input files for a single molecule are in *amber8/dat/leap/prep*, and the corresponding frcmod files are in

amber8/dat/leap/parm. Pre-equilibrated boxes are in *amber8/dat/leap/lib*. For example, to solvate a simple peptide in methanol, you could do the following:

```
source leaprc.ff99                      (get a standard force field)
loadAmberParams frcmod.meoh             (get methanol parameters)
peptide = sequence { ACE VAL NME }      (construct a simple peptide)
solvateBox peptide MEOHBOX 12.0 0.8     (solvate the peptide with meoh)
saveAmberParm peptide prmtop prmcrd
quit
```

Similar commands will work for other solvent models.

3. LEaP

3.1. Introduction

LEaP is the generic name given to the programs teLeap and xaLeap, which are generally run *via* the *tleap* and *xleap* shell scripts. These two programs share a common command language but the xleap program has been enhanced through the addition of an X-windows graphical interface. The name LEaP is an acronym constructed from the names of the older AMBER software modules it replaces: link, edit, and parm. Thus, LEaP can be used to prepare input for the AMBER molecular mechanics programs.

Both *tleap* and *xleap* are written in ANSI C; the former does not support graphics and therefore, it will run in a text window or from a script. The *xleap* script is meant to run on any machine that supports X-windows (Version 11 Revision 4 and latter versions); it does all of its graphics manipulations in generic X-windows. It does not depend on any system-dependent graphics to do 3D transformations or page-flipping. All of the user interface was written using David E. Smyth's Widget Creation Library (Wcl-1.05). This library is included in the LEaP distribution, as is the Xraw 3D widget set by Vladimir Romanovski (modeled on the ATHENA 3D widget set by Kaleb Keithley).

Using *tleap*, the user can:

```
Read AMBER PREP input files
Read AMBER PARM format parameter sets
Read and write Object File Format files (OFF)
Read and write PDB files
Construct new residues and molecules using simple commands
Link together residues and create nonbonded complexes of molecules
Place counterions around a molecule
Solvate molecules in arbitrary solvents
Modify internal coordinates within a molecule
Generate files that contain topology and parameters for AMBER.
```

In addition, with *xleap* the user can:

```
Access commands using a simple point and click interface
Draw new residues and molecules in a graphical environment
View structures graphically
Graphically dock molecules
Modify the properties of atoms, residues, and molecules using a
     spreadsheet editor
Input or alter molecular mechanics parameters using a spreadsheet editor.
```

3.2. Concepts

In order to effectively use LEaP it is necessary to understand the philosophy behind the program, especially the concepts of LEaP *commands*, *variables*, and *objects*. In addition to exploring these concepts, this section also addresses the use of external files and libraries with the program.

3.2.1. Commands

A researcher uses LEaP by entering commands that manipulate objects. An object is just a basic building block; some examples of objects are ATOMs, RESIDUEs, UNITs, and PARMSETs. The commands that are supported within LEaP are described throughout the manual and are defined in detail in the "Command Reference" section.

The heart of LEaP is a command-line interface that accepts text commands which direct the program to perform operations on objects. All LEaP commands have one of the following two forms:

```
command argument1 argument2 argument3 ...
variable = command argument1 argument2 ...
```

For example:

```
edit ALA
trypsin = loadPdb trypsin.pdb
```

Each command is followed by zero or more arguments that are separated by whitespace. Some commands return objects which are then associated with a variable using an assignment (=) statement. Each command acts upon its arguments, and some of the commands modify their arguments' contents. The commands themselves are case- insensitive. That is, in the above example, `edit` could have been entered as `Edit`, `eDiT`, or any combination of upper and lower case characters. Similarly, `loadPdb` could have been entered a number of different ways, including `loadpdb`. In this manual, we frequently use a mixed case for commands. We do this to enhance the differences between commands and as a mnemonic device. Thus, while we write `createAtom`, `createResidue`, and `createUnit` in the manual, the user can use any case when entering these commands into the program.

The arguments in the command text may be *objects* such as NUMBERs, STRINGs, or LISTs or they may be *variables*. These two subjects are discussed next.

3.2.2. Variables

A *variable* is a handle for accessing an object. A variable name can be any alphanumeric string whose first character is an alphabetic character. (Alphanumeric means that the characters of the name may be letters, numbers, or special symbols such as "*". The following special symbols should not be used in variable names: dollar sign, comma, period, pound sign, equal sign, space, semicolon, double quote, or list open or close characters { and }. LEaP commands should not be used as variable names. Variable names are case-sensitive: "ARG" and "arg" are different variables. Variables are associated with objects using an assignment statement not unlike regular computer languages such as Fortran or C.

```
mole = 6.02E23
MOLE = 6.02E23
myName = "Joe Smith"
listOf7Numbers = { 1.2 2.3 3.4 4.5 6 7 8 }
```

In the above examples, both `mole` and `MOLE` are variable names, whose contents are the same (6.02E23). Despite the fact that both `mole` and `MOLE` have the same contents, they are *not* the same variable. This is due to the fact that variable names are case-sensitive. LEaP maintains a

list of variables that are currently defined and this list can be displayed using the `list` command. The contents of a variable can be printed using the `desc` command.

3.2.3. Objects

The *object* is the fundamental entity in LEaP. Objects range from the simple objects NUMBERs and STRINGs to the complex objects UNITs, RESIDUEs, and ATOMs. Complex objects have properties that can be altered using the `set` command and some complex objects can contain other objects. For example, RESIDUEs are complex objects that can contain ATOMs and have the properties: residue name, connect atoms, and residue type.

3.2.3.1. NUMBERs

NUMBERs are simple objects and they are identical to double precision variables in Fortran and double variables in C.

3.2.3.2. STRINGs

STRINGs are simple objects that are identical to character arrays in C and similar to character strings in Fortran. STRINGs are represented by sequences of characters which may be delimited by double quote characters. Example strings are:

```
"Hello there"
"String with a "" (quote) character"
"Strings contain letters and numbers:1231232"
```

3.2.3.3. LISTs

LISTs are composed of sequences of other objects delimited by LIST open and close characters. The LIST open character is an open curly bracket ({) and the LIST close character is a close curly bracket (}). LISTs can contain other LISTs and be nested arbitrarily deep. Example LISTs are:

```
{ 1 2 3 4 }
{ 1.2 "string" }
{ 1 2 3 { 1 2 } { 3 4 } }
```

LISTs are used by many commands to provide a more flexible way of passing data to the commands. The `zMatrix` command has two arguments, one of which is a LIST of LISTs where each subLIST contains between three and eight objects.

3.2.3.4. PARMSETs (Parameter Sets)

PARMSETs are objects that contain bond, angle, torsion, and nonbond parameters for AMBER force field calculations. They are normally loaded from *e.g.* `parm94.dat` and `frcmod` files.

3.2.3.5. ATOMs

ATOMs are complex objects that do not contain any other objects. The ATOM object is similar to the chemical concept of atoms. Thus, it is a single entity that may be bonded to other ATOMs and it may be used as a building block for creating molecules. ATOMs have many properties that can be changed using the `set` command. These properties are defined below.

`name`

This is a case-sensitive STRING property and it is the ATOM's name. The `name`s for all ATOMs in a RESIDUE should be unique. The `name` has no relevance to molecular mechanics force field parameters; it is chosen arbitrarily as a means to identify ATOMs. Ideally, the `name` should correspond to the PDB standard, being 3 characters long except for hydrogens, which can have an extra digit as a 4th character.

`type`

This is a STRING property. It defines the AMBER force field atom type. It is important that the character case match the canonical type definition used in the appropriate "parm.dat" or "frcmod" file. For smooth operation, all atom types need to have element and hybridization defined by the `addAtomTypes` command. The standard AMBER force field atom types are added by the default "leaprc" file.

`charge`

The `charge` property is a NUMBER that represents the ATOM's electrostatic point charge to be used in a molecular mechanics force field.

`element`

The atomic element provides a simpler description of the atom than the `type`, and is used only for LEaP's internal purposes (typically when force field information is not available). The element names correspond to standard nomenclature; the character "?" is used for special cases.

`position`

This property is a LIST of NUMBERs. The LIST must contain three values: the (X, Y, Z) Cartesian coordinates of the ATOM.

AMBER also supports free energy calculations, where one chemical species is transformed into another and the (free) energy change associated with the transformation is measured. In order to perform these, the properties of the perturbed ATOMs must also be set. These properties correspond to the ATOM properties described above, but the values represent the final state of the perturbed species, as described below. If a free energy calculation is not to be performed, the following properties can be ignored.

`pertName`

This property can either be `null` or a case sensitive STRING. The property is a unique identifier for an ATOM in its final state during a free energy calculation. If it is `null` then the perturbed ATOM will inherit the unperturbed name. The `pertName` has no effect on calculations and is mainly useful as a reminder of what was intended.

`pertType`

This property can either be `null` or a STRING. If the value is `null`, or if it is the same as the ATOM's `type`, then the ATOM will not be marked as "perturbed". Otherwise, `pertType` is the AMBER force field atom type of the perturbed ATOM. This property is case-sensitive.

`pertCharge`

The `pertCharge` property is a NUMBER. In spite of its name, it actually represents the *difference* between the perturbed charge and the regular charge on an ATOM. Hence, if this value is zero (its default value), then the perturbed charge is the same as its regular charge. (Note that the meaning of this property is

changed from Amber versions prior to 8. If you have OFF files with perturbed charges in them from earlier versions, you will need to edit these to the new standard--either use a text editor or the edit table in xleap.)

(There is another important change between Amber version 8 and previous versions. In earlier versions, an atom was marked as "perturbed" only if the "PERTURB" column in the atom table (in xleap) was set to "true", or if a "set" command (with attribute Pert) was issued from the command line. In version 8, the "PERTURB" column (now named "unused") is ignored. An atom is considered to be perturbed if its PertType is different than its Type. The upshot of this is that you can change the charge of an atom without making the atom become "perturbed". This fits how sander works: energies with the perturbed charges are always computed (if needed) independent of which atoms (if any) are marked as "perturbed". However, if you want to use these perturbed prmtop files with the old *gibbs* program, this won't work; any atom with a changed charge needs to be explicitly marked as perturbed. To recover this older behavior, issue a "`set default gibbs on`" command before the `saveAmberParmPert` command.)

3.2.3.6. RESIDUEs

RESIDUEs are complex objects that contain ATOMs. RESIDUEs are collections of ATOMs, and are either molecules (e.g. formaldehyde) or are linked together to form molecules (e.g. amino acid monomers). RESIDUEs have several properties that can be changed using the `set` command. (Note that database RESIDUEs are each contained within a UNIT having the same name; the residue GLY is referred to as GLY.1 when setting properties. When two of these single-UNIT residues are joined, the result is a single UNIT containing the two RESIDUEs.)

One property of RESIDUEs is connection ATOMs. Connection ATOMs are ATOMs that are used to make linkages between RESIDUEs. For example, in order to create a protein, the N-terminus of one amino acid residue must be linked to the C-terminus of the next residue. This linkage can be made within LEaP by setting the N ATOM to be a connection ATOM at the N-terminus and the C ATOM to be a connection ATOM at the C-terminus. As another example, two CYX amino acid residues may form a disulfide bridge by crosslinking a connection atom on each residue.

There are several properties of RESIDUEs that can be modified using the `set` command. The properties are described below:

`connect0`
: This defines an ATOM that is used in making links to other RESIDUEs. In UNITs containing single RESIDUEs, the RESIDUE's `connect0` ATOM is usually defined as the UNIT's `head` ATOM. (This is how the standard library UNITs are defined.) For amino acids, the convention is to make the N-terminal nitrogen the `connect0` ATOM.

`connect1`
: This defines an ATOM that is used in making links to other RESIDUEs. In UNITs containing single RESIDUEs, the RESIDUE's `connect1` ATOM is usually defined as the UNIT's `tail` ATOM. (This is done in the standard library UNITs.) For amino acids, the convention is to make the C-terminal oxygen the `connect1` ATOM.

`connect2`
: This is an ATOM property which defines an ATOM that can be used in making links to other RESIDUEs. In amino acids, the convention is that this is the ATOM to which disulfide bridges are made.

`restype`
: This property is a STRING that represents the type of the RESIDUE. Currently, it can have one of the following values: `"undefined"`, `"solvent"`,

"protein", "nucleic", or "saccharide". Some of the LEaP commands behave in different ways depending on the type of a residue. For example, the solvate commands require that the solvent residues be of type "solvent". It is important that the proper character case be used when defining this property.

name — The RESIDUE name is a STRING property. It is important that the proper character case be used when defining this property.

3.2.3.7. UNITs

UNITs are the most complex objects within LEaP, and the most important. UNITs, when paired with one or more PARMSETs, contain all of the information required to perform a calculation using AMBER. UNITs have the following properties which can be changed using the set command:

head

tail — These define the ATOMs within the UNIT that are connected when UNITs are joined together using the sequence command or when UNITs are joined together with the PDB or PREP file reading commands. The tail ATOM of one UNIT is connected to the head ATOM of the next UNIT in any sequence. (Note: a "TER card" in a PDB file causes a new UNIT to be started.)

box — This property can either be null, a NUMBER, or a LIST. The property defines the bounding box of the UNIT. If it is defined as null then no bounding box is defined. If the value is a single NUMBER then the bounding box will be defined to be a cube with each side being NUMBER of angstroms across. If the value is a LIST then it must be a LIST containing three numbers, the lengths of the three sides of the bounding box.

cap — This property can either be null or a LIST. The property defines the solvent cap of the UNIT. If it is defined as null then no solvent cap is defined. If the value is a LIST then it must contain four numbers, the first three define the Cartesian coordinates (X, Y, Z) of the origin of the solvent cap in angstroms, the fourth NUMBER defines the radius of the solvent cap in angstroms.

Examples of setting the above properties are:

```
set dipeptide head dipeptide.1.N
set dipeptide box { 5.0 10.0 15.0 }
set dipeptide cap { 15.0 10.0 5.0 8.0 }
```

The first example makes the amide nitrogen in the first RESIDUE within "dipeptide" the head ATOM. The second example places a rectangular bounding box around the origin with the (X, Y, Z) dimensions of (5.0, 10.0, 15.0) in angstroms. The third example defines a solvent cap centered at (15.0, 10.0, 5.0) angstroms with a radius of 8.0 Å. **Note:** the "set cap" command does not actually solvate, it just sets an attribute. See the solvateCap command for a more practical case.

UNITs are complex objects that can contain RESIDUEs and ATOMs. UNITs can be created using the createUnit command and modified using the set commands. The contents of a UNIT can be modified using the add and remove commands.

3.2.3.8. Complex objects and accessing subobjects

UNITs and RESIDUEs are complex objects. Among other things, this means that they can contain other objects. There is a loose hierarchy of complex objects and what they are allowed to contain. The hierarchy is as follows:

- UNITs can contain RESIDUEs and ATOMs.
- RESIDUEs can contain ATOMs.

The hierarchy is loose because it does not forbid UNITs from containing ATOMs directly. However, the convention that has evolved within LEaP is to have UNITs directly contain RESIDUEs which directly contain ATOMs.

Objects that are contained within other objects can be accessed using dot "." notation. An example would be a UNIT which describes a dipeptide ALA-PHE. The UNIT contains two RESIDUEs each of which contain several ATOMs. If the UNIT is referenced (named) by the variable `dipeptide`, then the RESIDUE named ALA can be accessed in two ways. The user may type one of the following commands to display the contents of the RESIDUE:

```
desc dipeptide.ALA
desc dipeptide.1
```

The first translates to "some RESIDUE named `ALA` within the UNIT named `dipeptide`". The second form translates as "the RESIDUE with sequence number `1` within the UNIT named `dipeptide`". The second form is more useful because every subobject within an object is guaranteed to have a unique sequence number. If the first form is used and there is more than one RESIDUE with the name `ALA`, then an arbitrary residue with the name `ALA` is returned. To access ATOMs within RESIDUEs, the notation to use is as follows:

```
desc dipeptide.1.CA
desc dipeptide.1.3
```

Assuming that the ATOM with the name `CA` has a sequence number `3`, then both of the above commands will print a description of the $alpha$−carbon of RESIDUE `dipeptide.ALA` or `dipeptide.1`. The reader should keep in mind that `dipeptide.1.CA` is the ATOM, an object, contained within the RESIDUE named `ALA` within the variable `dipeptide`. This means that `dipeptide.1.CA` can be used as an argument to any command that requires an ATOM as an argument. However `dipeptide.1.CA` is not a variable and cannot be used on the left hand side of an assignment statement.

In order to further illustrate the concepts of UNITs, RESIDUEs, and ATOMs, we can examine the log file from a LEaP session. Part of this log file is printed below.

```
> loadOff all_amino94.lib
> desc GLY
UNIT name: GLY
Head atom: .R<GLY 1>.A<N 1>
Tail atom: .R<GLY 1>.A<C 6>
Contents:
R<GLY 1>
> desc GLY.1
RESIDUE name: GLY
```

```
RESIDUE sequence number: 1
RESIDUE PDB sequence number: 0
Type: protein
Connection atoms:
 Connect atom 0: A<N 1>
 Connect atom 1: A<C 6>
Contents:
A<N 1>
A<HN 2>
A<CA 3>
A<HA2 4>
A<HA3 5>
A<C 6>
A<O 7>
> desc GLY.1.3
ATOM
Normal                  Perturbed
Name:    CA             CA
Type:    CT             CT
Charge:  -0.025         0.000
Element: C              (not affected by pert)
Atom position: 3.970048, 2.845795, 0.000000
Atom velocity: 0.000000, 0.000000, 0.000000
  Bonded to .R<GLY 1>.A<N 1> by a single bond.
  Bonded to .R<GLY 1>.A<HA2 4> by a single bond.
  Bonded to .R<GLY 1>.A<HA3 5> by a single bond.
  Bonded to .R<GLY 1>.A<C 6> by a single bond.
```

In this example, command lines are prefaced by ">" and the LEaP program output has no such character preface. The first command,

```
> loadOff all_amino94.lib
```

loads an OFF library containing amino acids. The second command,

```
> desc GLY
```

allows us to examine the contents of the amino acid UNIT, GLY. The UNIT contains one RESIDUE which is named GLY and this RESIDUE is the first residue in the UNIT (`R<GLY 1>`). In fact, it is also the only RESIDUE in the UNIT. The `head` and `tail` ATOMs of the UNIT are defined as the N- and C-termini, respectively. The `box` and `cap` UNIT properties are defined as `null`. If these latter two properties had values other than `null`, the information would have been included in the output of the `desc` command.

The next command line in the session,

```
> desc GLY.1
```

enables us to examine the first residue in the GLY UNIT. This RESIDUE is named GLY and its residue type is that of a `protein`. The `connect0` ATOM (N) is the same as the UNITs' `head` ATOM and the `connect1` ATOM (C) is the same as the UNITs' `tail` ATOM. There are seven ATOM objects contained within the RESIDUE GLY in the UNIT GLY.

Finally, let us look at one of the ATOMs in the GLY RESIDUE.

```
> desc GLY.1.3
```

The ATOM has a name (CA) that is unique among the atoms of the residue. The AMBER force field atom type for CA is CT. The type of element, atomic point charge, and Cartesian coordinates for this ATOM have been defined along with its bonding attributes. Other force field parameters, such as the van der Waals well depth, are obtained from PARMSETs.

3.3. Starting LEaP

```
% xleap [-h] [-I dir] [-f file] [-s]
```

```
% tleap [-h] [-I dir] [-f file] [-s]
```

The user may enter several options when starting the LEaP program. If the option "`-h`" is used (e.g., `xleap -h`), then the program will print a list of start-up options and then exit. A directory may be added to the program's search path by using the option: "`-I dir`". This will cause the program to search `dir` whenever a file is requested. If the user would like to execute LEaP commands at start-up, they should use the option: "`-f file`". Finally, if the user enters the command option "`-s`", the "leaprc" file will not be executed at start-up.

A file called "leaprc" is executed as a script file at the start of the LEaP session unless the user suppresses it with a command line option. Sample files are in *$AMBERHOME/dat/leap/cmd*, and you may wish to copy one of these to become "your" default file. LEaP will look first for a *learpc* file in the user's current directory, then in any directories included with `-I` flags.

3.3.1. Verbosity

The `verbosity` command is used to control how much output LEaP displays to the user. A verbosity level of `0` tells LEaP to print the minimum amount of information. A verbosity level of `1` tells LEaP to print all information it can, and a verbosity level of `2` tells LEaP to print all information and to display each line read from source files executed using the `source` command.

3.3.2. Log File

The command line interface allows the user to specify a log file that is used to log all input and output within the command line environment. The log file is named using the `logFile` command. The file has two purposes: to allow the user to see a complete record of operations performed by LEaP, and to help recover from (and recreate) program crashes. Output from LEaP commands is written to the log file at a verbosity level of `2` regardless of the verbosity level set by the user using the `verbosity` command. Each line in the log file that was typed in by the user begins with the two characters "`> `" (a greater-than sign followed by a space). This allows the user to extract the commands typed into LEaP from the log file to create a script file that can be

executed using the `source` command. This provides a type of insurance against program crashes by allowing the user to regenerate their interactive sessions. An example of a command that works on UNIX systems and that will create a script to reenact a LEaP session is:

```
% cat LOGFILE | grep "^> " | sed "s/^> //" > SOURCEFILE.x
```

Note that changes via graphical and table interfaces (xleap) are not captured by command-line traces.

3.4. Using LEaP

In this section, we describe how to use the tleap and xleap user interfaces. Strategies for using LEaP in research are discussed in the subsequent section on using LEaP with AMBER.

tleap (terminal LEaP) is the non-graphical, command-line-only interface to LEaP. It has the same functionality as the *xleap* main window (Universe Editor Command Window, described below), and uses standard text control keys.

xleap is a windowing interface to LEaP. In addition to the command-line interface contained in the Universe Editor window, it has a Unit Editor (graphical molecule editor), an Atom Properties Editor, and a Parmset Editor. These editors are discussed in subsequent subsections.

3.4.1. Universe Editor

The window that first appears when the user starts xleap is called the Universe Editor. The Universe Editor is the most basic way in which users can interact with xleap. It has two parts, the "command window," which corresponds to the tleap command interface, and the "pulldown" items above the window, which provide mouse-driven methods to generate specific commands for the command window, either directly or via popped-up dialog boxes.

The items in the pulldowns allow the user to generate commands using dialog boxes. To display the "File" pulldown, for example, press the left mouse button on "File;" to select an item in the pulldown, keep the button down, move the mouse to highlight the item, then release the mouse button. A dialog box will then pop up containing fields which the user can fill in, and lists from which values can be chosen; these will be used to generate commands for the command window interface.

Currently, the following pulldown/popup commands are defined:

`loadOff` The dialog box contains a single file list. The user can move about the subdirectories and select the desired LEaP OFF library to load. Alternately, the name of the file to be loaded may be typed. The user should press the "Accept" button after selecting a file in order to execute the command.

`saveOff` This dialog box contains a list of UNITs/PARMSETs and a file list. The user must choose the UNIT or PARMSET to save, and choose the file to which to write. If the file to be written to does not exist, the user may type the name of a new file into the file name text box. The user can enter the command by pressing the "Accept" button.

`loadAmberPrep`
: The dialog box contains a single file list. The user can move about the subdirectories, and select the AMBER PREP file to load. Alternately, they may type the name of the file to be loaded. The user should press the "Accept" dialog box button after selecting a file.

`loadPdb`
: There are two parts to this dialog box. The PDB file will be read into a UNIT and that UNIT will be associated with a variable. The variable name to associate with the UNIT is entered into the first text field. The name of the PDB file is either selected from the file list or the file name is typed into the dialog box. The user can execute the command by pressing the "Accept" button.

`impose`
: This dialog box has three parts. The first part is a list of UNITs from which the user can select the UNIT which is to be changed. The second part is a list of STRING objects that may or may not contain internal coordinates. The third part is a text field for entry of RESIDUE sequence numbers, or ranges of sequence numbers. The user executes the command by pressing the "Accept" button.

`edit`
: A list of UNITs and PARMSETs that can be edited is presented to the user. The user may select one or type in the name of a UNIT. The user may "Accept" or "Cancel" the command by pressing one of these two buttons.

`source`
: The user can select a file which is to be used in a `source` command from the file list. Alternately, they may type the name of the file to be loaded. The user should press the "Accept" button after selecting a file.

`verbosity`
: The user is presented with three levels of `verbosity` in order to regulate the amount of output to be displayed during the LEaP session. The user should select one of these `verbosity` level buttons and press the "Accept" button to enter the command.

`quit`
: The user may "Accept" or "Cancel" the `quit` command.

3.4.2. Unit Editor

When the user enters the `edit` command from the Universe Editor Command Window, the Unit Editor will be displayed if the argument to the `edit` command is an existing UNIT or a nonexistent (i.e. new) object. The Parmset Editor will be activated if the argument is a PARMSET. The Parmset Editor is discussed later in this subsection.

The Unit Editor has five parts. At the top of the window is a pulldown menu bar; below it is a set of buttons titled "Manipulation" that define the mode of mouse activity in the graphics window, and below that, a list of elements to select for the manipulation "Draw" mode (selecting one automatically selects "Draw" mode). Then comes the graphical molecule-editing ("viewing") window itself, and at the very bottom a text window where status and errors are reported.

3.4.2.1. Unit Editor Menu Bar

The menu bar has three pulldowns: "Unit," "Edit," and "Display."

`Unit`
: The Unit pulldown contains commands affecting the whole UNIT.

  "Check unit" − checks the UNIT in the viewing window for improbable bond lengths, missing force field atom types, close nonbonded contacts, and a non-integral and non-zero total charge. Information is printed in the text window at the bottom of the Unit Editor.

"Calculate charge" − the total electrostatic charge for the UNIT is displayed in the text window at the bottom of the Unit Editor.

"Build," "Add H & Build" − the coordinates of new atoms are adjusted according to hybridization (inferred from bonds) and standard geometries. (See also the `Edit` pulldown's "Relax selection.) Newly-drawn ATOMs are marked as "unbuilt" until they are marked otherwise by one of the Build commands or by the `Edit` pulldown's "Mark selection (un)built." The builder *only* builds coordinates for unbuilt ATOMs. This allows users to draw molecules piecemeal and make adjustments as they draw, without worrying that the builder is going to undo their work. "Add H & Build" adds hydrogens to the ATOMs that do not have a full valence and builds coordinates for the hydrogens and any other ATOMs that are marked "unbuilt." The number of hydrogens added to each ATOM is determined by the hybridization and element type of each ATOM.

"Import unit" − a selection window pops up for the user to incorporate a copy of another unit in the current one. The imported unit will generally superimpose on the existing one. (Hint: select all atoms in the current unit before doing this to simplify dragging them apart using the Manipulation `Move` mode.)

"Close" − Exit the Editor.

`Edit` The Edit pulldown contains commands relating to the currently- selected ATOMs in the viewer window. Selection is described below in the "Manipulation buttons" section.

"Relax selection" − performs a limited energy minimization of all selected ATOMs, leaving unselected ATOMs fixed in place, by relaxing strained bonds, angles, and torsions. If atom types have been assigned and can be found in the currently-loaded force field, force field parameters are used. If no types are available then default parameters are used that are based on ATOM hybridization. This command invokes an iterative algorithm that can take some time to converge for large systems. As the algorithm proceeds, the modified UNIT will be continuously updated within the viewing window. The user can stop the process at any time by placing the mouse pointer within the viewing window and typing `control-C`. Since only internal coordinates are energy minimized, steric overlap can result.

"Edit selected atoms" − pops up an Atom Properties Editor, a tool for examining/setting the properties of the selected ATOMs. The Atom Properties Editor allows the user to edit the ATOM names, types, charges, perturbed names, etc. in a convenient table format. It is described in a separate subsection below.

"Flip chirality" This command inverts the chirality of all selected ATOMs. In order for the chirality to be inverted, the ATOM cannot be in more than one ring. The operation causes the lightest chains leaving the ATOM to be moved so as to invert the chirality. If the ATOM has only three chains attached to it, then only one of the chains will be moved. **Note:** this command is rather apt to crash LEaP.

"Select Rings/Residues/Molecules" − expands the currently selected group of atoms to include all partially-contained rings, residues, or molecules.

"Show everything" − causes all ATOMs to become visible.

"Hide selection" − makes all selected ATOMs invisible.

"Show selection only" − makes only selected ATOMs visible.

"Mark selection unbuilt/built" − see "Unit/Build," above.

`Display` The Display pulldown contains commands that determine what information is displayed within the viewing window.

"Names" − toggles display of ATOM names at each ATOM position.

"Perturbed names" − toggles display of perturbed ATOM names. The perturbed names are displayed immediately after the unperturbed names and are prefixed with a forward slash "/". (See the "Concepts" section for a discussion of Perturbed ATOM names.)

"Types" − toggles display of molecular mechanics atom types. The ATOM types are displayed within parentheses "()".

"Charges" − toggles display of the atomic charges.

"Perturbed types" − toggles display of perturbed atom types of the ATOMs. Perturbed types are displayed within the same parentheses as the unperturbed types, immediately after the unperturbed types, and are prefixed by a forward slash "/".

"Residue names" − toggles display of residue names. These are displayed at the position of the first ATOM, before any of that ATOM's information that may be displayed. The residue names are displayed within angled brackets "<>".

"Axes" − toggles display of the Cartesian coordinate axes. The origin of the axes coincides with the origin of Cartesian space.

"Periodic box" − toggles display of the periodic box, if the UNIT has one.

3.4.2.2. Unit Editor Manipulation Buttons

The Manipulation buttons are Select, Twist, Move, Erase, and Draw. They determine the behavior of the mouse left-button when the mouse pointer is in the Viewing Window.

`Select` This button allows one to select part or all of a UNIT in anticipation of a subsequent operation or action. In the `Select` mode, the user can highlight ATOMs within the viewing window for special operations. The mouse pointer becomes a pointing hand in the viewing window in this mode. Selected ATOMs are displayed in a different color (or different line styles on monochrome systems) from all other ATOMs. Atoms can be selected with the left-button in several ways: first, clicking on an atom and releasing selects that atom. Clicking twice in a row on an atom (at any speed) selects all atoms (this is a bug - only the residue should be selected). Keeping the button down and moving to release on another atom selects all ATOMs in the shortest chain between the two ATOMs, if such a chain exists. Finally, by first pressing the button in empty space, and holding it down as the mouse is moved, one can "drag a box" enclosing atoms of interest. Note that a current selection can be expanded by using the "Edit" menubar pulldown select option to complete any partial selection of rings, residues or molecules.

If the user holds down the SHIFT key while performing any of the above actions, the same effect will be seen, except ATOMs will be unselected.

`Twist` `Twist` mode operates on previously-`Select`ed atoms. The intention is to allow rotation about dihedrals; if too many atoms are selected, odd transformations can occur. While in the `Twist` mode, the mouse pointer looks like a curved arrow. Twisting is driven by holding down the left-button anywhere in the

viewing window and moving the mouse up and down. It is important to select a complete torsion (all four atoms) before trying to "twist" it.

`Move` Like `Twist`, `Move` mode operates on previously-`Selected` atoms. While in the `Move` mode, the mouse pointer looks like four arrows coming out of one central point. Holding down the left-button anywhere allows movement of these atoms by dragging in any direction in the viewing plane. (The view can be rotated by holding down the middle-button to allow any movement desired.) This option allows the user to move the selected ATOMs relative to the unselected ATOMs.

To *rotate* the selected ATOMs relative to the unselected ones, press and drag the mode (left) button while holding down the SHIFT key. The selected ATOMs will rotate around a central ATOM on a "virtual sphere" (see the subsubsection below on the rotate (middle) button for more information on the "virtual sphere"). The user can change which ATOM is used as the center of rotation by clicking the mode (left) button on any of the ATOMs in the window.

`Erase` `Erase` mode causes the mouse pointer to resemble a chalkboard eraser when it is in the viewing window. Clicking the left-button will delete any atoms or bonds under this mouse pointer, one atom or bond per click.

`Draw` Choosing `Draw` is equivalent to choosing the default "Elements" atom in the next array of buttons; the initial default is carbon. While in the `Draw` mode, the mouse pointer is a pencil when in the viewing window. Clicking the left-button deposits an atom of the current element, while dragging the mouse pointer with the left-button held down draws a bond: if no atom is found where the button is released, one is created.

When the mouse pointer approaches an ATOM, the end of the line connected to the pointer will "snap" to the nearest ATOM. This is to facilitate drawing of bonds between ATOMs. Any bonds that are drawn will by default be single bonds. To change the order of a bond, the user would move the mouse to any point along the bond and click the mode (left) button. This will cause the order of the bond to increase until it is reset back to a single bond. The user can cycle through the following bond order choices: single, double, triple, and aromatic.

If the user rotates a structure as it is being drawn, she will notice that all of the ATOMs that have been drawn lie in the same plane. New ATOMs are automatically placed in the plane of the screen. The fact that LEaP places the new ATOMs in the same plane is not a handicap because once a rough sketch of part of the structure is compete, the user can invoke one of LEaP's two model building facilities ("Unit/Build" and "Edit/Relax Selection" in the Unit Editor Menu bar) to build full three dimensional coordinates.

3.4.2.3. Unit Editor Elements Buttons

`C, H, O, ...`

These buttons put the viewing window in `Draw` mode if it is not in that mode already, and select the drawing element. The more common elements have their own buttons, and all elements are also found by pulling down the `other elements` button.

3.4.2.4. Unit Editor Viewing Window

The viewing window displays a projection of the UNIT currently being edited. The user can manipulate the structure within the viewing window with the mouse. By moving the mouse and holding down the mouse buttons, the user can rotate, scale, and translate the UNIT within the window. The functions attached to the mouse buttons are:

`Rotate (Middle button)`
: By pressing the rotate (middle) button within the viewing window and dragging the mouse, the user can rotate the UNIT around the center of the viewing window. While the rotate (middle) button is down, a circle appears within the viewing window, representing a "virtual sphere trackball." As the user drags the mouse around the outside of the circle, the UNIT will spin around the axis normal to the screen. As the user drags the mouse within the circle, the UNIT will spin around the axis in the screen, perpendicular to the movement of the mouse. The structures that are being viewed can be considered to be embedded within a sphere of glass. The circle is the projection of the edge of the sphere onto the screen. Rotating a UNIT while the mouse is within the circle is akin to placing a hand on a glass sphere and turning the sphere by pulling the hand. The rotate operation does not modify the coordinates of the ATOMs; rather, it simply changes the user's point of view.

`Translate (Right button)`
: By pressing the translate (right) button within the viewing window and dragging the mouse around the viewing window, the user can translate the UNIT within the plane of the screen. The structures will follow the mouse as it moves around the window. This operation does not modify the coordinates of the UNIT.

`Scale (middle plus right button)`
: If the scale "button" (holding the middle and right buttons down at the same time) is depressed, the user will change the size of the structures within the viewing window. Pressing the scale (middle plus right) button and dragging the mouse up and down the screen will increase and decrease the scale of the structures. This operation does not modify the coordinates of the UNIT.

`Mode button (left button) and the viewing window mode`
: The function of the left button is determined by the current mode of the viewing window as described in the "Manipulation" section, above. When the mouse enters the viewing window it changes shape to reflect the current mode of the viewing window.

`Spacebar`
: Another always-available operation when the mouse pointer is in the viewing window is the keyboard spacebar. It centers and normalizes the size of the molecule in the viewing window. This is especially useful if the UNIT becomes "lost" due to some operation.

The functions of the middle and right buttons are fixed and always available to the user. This allows the user to change the viewpoint of the UNIT within the viewing window regardless of its current mode. The user might ask why there are controls to translate in the plane of the screen, but not out of the plane of the screen. This is because LEaP does not have depth-cueing or stereo projection and this makes it difficult for users to perceive changes in the depth of a structure. However, the user can rotate the entire UNIT by 90 degrees which will orient everything so that the direction that was coming out of the screen becomes a direction lying in the plane of the screen. Once the UNIT has been rotated using the rotate (middle) button, the user can translate

the structure anywhere in space. While it does take some getting used to, users can become very adept at the combination of rotations and translations.

3.4.3. Atom Properties Editor

The Atom Properties Editor is popped up by the Unit Editor when the user selects the `Edit selected atoms` command from the `Edit` pulldown. The Atom Properties Editor allows the user to edit the properties of ATOMs using a convenient table format. ATOM properties are: name, type, charge, element, perturbed name, perturbed type, and perturbed charge. (Mass is not perturbed.)

3.4.4. Parmset Editor

If the user enters the command `edit Foo` in the Universe Editor and `Foo` is a PARMSET, then a Parmset Editor is popped up. First, a window appears which contains a number of buttons. The buttons list the parameters that can be edited − Atom, Bond, Angle, Proper Torsion, Improper Torsion, and Hydrogen Bond − and an option to close the editor. Choosing one of the parameter buttons will pop up a Table Editor. This editor resembles that of the Atom Properties Editor, having three parts: the Menu Bar, Status Window, and Table Window.

3.5. Basic instructions for using LEaP with AMBER

This section gives an overview of how LEaP is most commonly used. Detailed descriptions of all the commands are given in the following section.

3.5.1. Building a Molecule For Molecular Mechanics

In order to prepare a molecule within LEaP for AMBER, three basic tasks need to be completed.

(1) Any needed UNIT or PARMSET objects must be loaded;

(2) The molecule must be constructed within LEaP;

(3) The user must output topology and coordinate files from LEaP to use in AMBER.

The most typical command sequence is the following:

```
source leaprc.ff94              load a force field
x = loadPdb trypsin.pdb         load in a structure
   ....                         add in cross-links, solvate, etc.
saveAmberParm x prmtop prmcrd   save files for sander or gibbs
```

There are a number of variants of this:

(1) Although `loadPdb` is by far the most common way to enter a structure, one might use `loadOff`, `loadMol2`, or `loadAmberPrep`, or use the `zMatrix` command to build a molecule from a z-matrix. See the Commands section below for descriptions of these options. For situations where you do not have a starting structure (in the form of a pdb file) LEaP can be used to build the molecule; you may find, however, that this is not always as easy as it might be. Many experienced Amber users turn to other (commercial and non-commercial) programs to create their initial structures.

(2) Be very attentive to any errors produced in the `loadPdb` step; these generally mean that LEaP has mis-read the file. A general rule of thumb is to keep editing your input pdb file until LEaP stops complaining. It is often convenient to use the `addPdbAtomMap` or `addPdbResMap` commands to make systematic changes from the names in your pdb files to those in the Amber topology files; see the *leaprc* files for examples of this.

(3) The `saveAmberParm` command cited above is appropriate for calculations that do not compute free energies; for the latter you will need to use `saveAmberParmPert`. For polarizable force fields, you will need to add `Pol` to the above commands (see the Commands section, below.)

If you do want to build your own molecule, here is a brief description of how one would make a water molecule: After the xleap program is started and a PARMSET is loaded, the user can enter the Unit Editor with the `edit` command. If the command argument (WAT) is not an existing UNIT, a new RESIDUE and UNIT will be created and the program will display a Unit Editor for WAT.

The first objective is to draw and build the molecule. In the Control Window is a button named `draw`. The user should select this button with the left mouse button. The Viewing Window will now be set to the Draw mode. The user should then select the O (oxygen) element button in the Control Window. This will set the drawing element type to oxygen. The `Draw mode` mouse button (left button) is depressed and clicked anywhere on the screen. The user can then release the mouse button. Now the user can select the `Unit pulldown` command: `Add H & Build`. Two hydrogen ATOMs will be added to the oxygen and the molecular structure will be generated using the geometry builder rules. The user may want to rotate the molecule, using the middle mouse button, to confirm that the geometry is correct.

Next, the user needs to edit the ATOMs. The entire molecule should be selected by pressing the Manipulation `Select` option and then pressing the `Select mode` mouse button (left button) anywhere in the Viewing Window background and dragging the mouse until the select box encompasses the molecule. The mouse button can then be released. The user should then choose the `Edit selected atoms` command from the `Edit` pulldown. An Atom Properties Editor will appear.

The Unit Editor has already assigned names to the ATOMs and if desired, the user can change the names. In order for correct AMBER force field parameters to be assigned, the user must define the oxygen and hydrogen ATOM types as "OW" and "HW", respectively. The user should also assign electrostatic point charges to each ATOM. The Atom Properties Editor can then be closed by choosing the `Save and quit` command in the `Table` pulldown. The UNIT has been created and the user can return to the xleap Universe Editor.

```
> #
> #  Load the main parameter set:
> #
> parm94 = loadAmberParams parm94.dat
Loading parameters: parm94.dat
> #
> #  Graphically create a water molecule within
> #  the Unit Editor:
> #
> edit WAT
> #
```

```
> #  If necessary, load a PDB file to obtain correct
> #  Cartesian coordinates:
> #
> wat = loadPdb Wat.pdb
Loading PDB file: ./Wat.pdb
  total atoms in file: 3
```

3.5.2. Amino Acid Residues

The accompanying table shows the amino acid UNITs and their aliases as defined in the LEaP libraries.

For each of the amino acids found in the LEaP libraries, there has been created an N-terminal and a C-terminal analog. The N-terminal amino acid UNIT/RESIDUE names and aliases are prefaced by the letter N (e.g. NALA) and the C-terminal amino acids by the letter C (e.g. CALA}. If the user models a peptide or protein within LEaP, they may choose one of three ways

Group or residue	Residue Name, Alias
Acetyl beginning group	ACE
Amine ending group	NHE
N-methylamine ending group	NME
Alanine	ALA
Arginine	ARG
Asparagine	ASN
Aspartic acid	ASP
Aspartic acid--protonated	ASH
Cysteine	CYS
Cystine, S--S crosslink	CYX
Glutamic acid	GLU
Glutamic acid--protonated	GLH
Glutamine	GLN
Glycine	GLY
Histidine, delta H	HID
Histidine, epsilon H	HIE
Histidine, protonated	HIP
Isoleucine	ILE
Leucine	LEU
Lysine	LYS
Methionine	MET
Phenylalanine	PHE
Proline	PRO
Serine	SER
Threonine	THR
Tryptophan	TRP
Tyrosine	TYR
Valine	VAL

to represent the terminal amino acids. The user may use 1) standard amino acids, 2) protecting groups (ACE/NME), or 3) the charged C- and N-terminal amino acid UNITs/RESIDUEs. If the standard amino acids are used for the terminal residues, then these residues will have incomplete valences. These three options are illustrated below:

```
{ ALA VAL SER PHE }
{ ACE ALA VAL SER PHE NME }
{ NALA VAL SER CPHE }
```

The default for loading from PDB files is to use N- and C-terminal residues; this is established by the `addPdbResMap` command in the default `leaprc` files. To force incomplete valences with the standard residues, one would have to define a sequence ("`x = { ALA VAL SER PHE }`") and use `loadPdbUsingSeq`, or use `clearPdbResMap` to completely remove the mapping feature.

Histidine can exist either as the protonated species or as a neutral species with a hydrogen at the delta or epsilon position. For this reason, the histidine UNIT/RESIDUE name is either HIP, HID, or HIE (but not HIS). The default "leaprc" file assigns the name HIS to HID. Thus, if a PDB file is read that contains the residue HIS, the residue will be assigned to the HID UNIT object. This feature can be changed within one's own "leaprc" file.

The AMBER force fields also differentiate between the residue cysteine (CYS) and the similar residue that participates in disulfide bridges, cystine (CYX). The user will need to load the PDB file using the `loadPdbUsingSeq` command, substituting CYX for CYS in the sequence wherever a disulfide bond will be created. (Or alternatively, the PDB file can be manually edited to change CYS to CYX.) Then the user will have to explicitly define, using the `bond` command, the disulfide bond for a pair of cystines, as this information is not read from the PDB file.

3.5.3. Nucleic Acid Residues

The following are defined for the 1994 force field.

Group or residue	Residue Name, Alias
Adenine	DA,RA
Thymine	DT
Uracil	RU
Cytosine	DC,RC
Guanine	DG,RG

The "D" or "R" prefix can be used to distinguish between deoxyribose and ribose units; with the default `leaprc` file, ambiguous residues are assumed to be deoxy. Residue names like "DA" can be followed by a "5" or "3" ("DA5", "DA3") for residues at the ends of chains; this is also the default established by `addPdbResMap`, even if the "5" or "3" are not added in the PDB file. The "5" and "3" residues are "capped" by a hydrogen; the plain and "3" residues include a "leading" phosphate group. Neutral residues capped by hydrogens end in "N," such as "DAN."

3.5.4. Miscellaneous Residues

Miscellaneous Residue	unit/residue name
TIP3P water molecule	TP3
TIP4P water model	TP4
TIP5P water model	TP5
SPC/E water model	SPC
Cesium cation	Cs+
Potassium cation	K+
Rubidium cation	Rb+
Lithium cation	Li+
Sodium cation	Na+ or IP
Chlorine	Cl- or IM
Large cation	IB

"IB" represents a solvated monovalent cation (say, sodium) for use in vacuum simulations. The cation UNITs are found in the files "ions91.lib" and "ions94.lib", while the water UNITs are in the file "solvents.lib". The `leaprc` files assign the variables WAT and HOH to the TP3 UNIT found in the OFF library file. Thus, if a PDB file is read and that file contains either the residue name HOH or WAT, the TP3 UNIT will be substituted. See the `solvate` commands and Chapter 2.9 for a discussion of how to use other water models.

A periodic box of 216 TIP3P waters (TIP3PBOX) is provided in the file "solvents.lib". The box measures 18.774 angstroms on a side. This box of waters has been equilibrated by a Monte Carlo simulation. It is the UNIT that should be used to solvate systems with TIP3P water molecules within LEaP. It has been provided by W. L. Jorgensen. Boxes are also available for other water models (TIP4P, TIP5P, POL3, SPC/E), and for chloroform, methanol, and N-methylacetamide; these are described in Chapter 2.

3.6. Commands

The following is a description of the commands that can be accessed using the command line interface in *tleap*, or through the command line editor in *xleap*. Whenever an argument in a command line definition is enclosed in brackets ([arg]), then that argument is optional. When examples are shown, the command line is prefaced by "> ", and the program output is shown without this character preface.

Some commands that are almost never used have been removed from this description to save space. You can use the "help" facility to obtain information about these commands; most only make sense if you understand what the program is doing behind the scenes.

3.6.1. add

`add a b`

`UNIT/RESIDUE/ATOM` *a,b*

Add the object *b* to the object *a*. This command is used to place ATOMs within RESIDUEs, and RESIDUEs within UNITs. This command will work only if *b* is not contained by any other object.

The following example illustrates both the `add` command and the way the tip3p water molecule is created for the LEaP distribution tape.

```
> h1 = createAtom H1 HW  0.417
> h2 = createAtom H2 HW  0.417
> o  = createAtom O  OW -0.834
>
> set h1 element H
> set h2 element H
> set o  element O
>
> r = createResidue TIP3
> add r h1
> add r h2
> add r o
>
> bond h1 o
> bond h2 o
> bond h1 h2
>
> TIP3 = createUnit TIP3
>
> add TIP3 r
> set TIP3.1 restype solvent
> set TIP3.1 imagingAtom TIP3.1.O
>
> zMatrix TIP3 {
>       { H1 O 0.9572 }
>       { H2 O H1 0.9572 104.52 }
> }
>
> saveOff TIP3 water.lib
Saving TIP3.
Building topology.
Building atom parameters.
```

3.6.2. addAtomTypes

`addAtomTypes { { type element hybrid } { ... } ... }`

```
STRING  type
STRING  element
STRING  hybrid
```

Define element and hybridization for force field atom types. This command for the standard force fields can be seen in the default `leaprc` files. The STRINGs are most safely rendered using quotation marks. If atom types are not defined, confusing messages about hybridization can result when loading PDB files.

3.6.3. addIons

```
addIons unit ion1 numIon1 [ion2 numIon2]
```

```
UNIT    unit
UNIT    ion1
NUMBER  numIon1
UNIT    ion2
NUMBER  numIon2
```

Adds counterions in a shell around *unit* using a Coulombic potential on a grid. If *numIon1* is 0, then the *unit* is neutralized. In this case, *numIon1* must be opposite in charge to *unit* and *numIon2* cannot be specified. If solvent is present, it is ignored in the charge and steric calculations, and if an ion has a steric conflict with a solvent molecule, the ion is moved to the center of said molecule, and the latter is deleted. (To avoid this behavior, either solvate _after_ addions, or use addIons2.) Ions must be monoatomic. This procedure is not guaranteed to globally minimize the electrostatic energy. When neutralizing regular-backbone nucleic acids, the first cations will generally be placed between phosphates, leaving the final two ions to be placed somewhere around the middle of the molecule.The default grid resolution is 1 Å, extending from an inner radius of (maxIonVdwRadius + maxSoluteAtomVdwRadius) to an outer radius 4 Å beyond. A distance-dependent dielectric is used for speed.

3.6.4. addIons2

```
addIons2 unit ion1 numIon1 [ion2 numIon2]
```

```
UNIT    unit
UNIT    ion1
NUMBER  numIon1
UNIT    ion2
NUMBER  numIon2
```

Same as addIons, except solvent and solute are treated the same.

3.6.5. addPath

```
addPath path
```

```
STRING  path
```

Add the directory in *path* to the list of directories that are searched for files specified by other commands. The following example illustrates this command.

```
> addPath /disk/howard
/disk/howard added to file search path.
```

After the above command is entered, the program will search for a file in this directory if a file is specified in a command. Thus, if a user has a library named "/disk/howard/rings.lib" and the user wants to load that library, one only needs to enter load rings.lib and not load /disk/howard/rings.lib.

3.6.6. addPdbAtomMap

`addPdbAtomMap list`

```
LIST    list
```

The atom Name Map is used to try to map atom names read from PDB files to atoms within residue UNITs when the atom name in the PDB file does not match an atom in the residue. This enables PDB files to be read in without extensive editing of atom names. Typically, this command is placed in the LEaP start-up file, "leaprc", so that assignments are made at the beginning of the session. The LIST is a LIST of LISTs. Each sublist contains two entries to add to the Name Map. Each entry has the form:

```
{ string string }
```

where the first *string* is the name within the PDB file, and the second *string is the name in the residue UNIT.*

3.6.7. addPdbResMap

`addPdbResMap list`

```
LIST    list
```

The Name Map is used to map RESIDUE names read from PDB files to variable names within LEaP. Typically, this command is placed in the LEaP start-up file, "leaprc", so that assignments are made at the beginning of the session. The LIST is a LIST of LISTs. Each sublist contains two or three entries to add to the Name Map. Each entry has the form:

```
{ double string string }
```

where *double* can be 0 or 1, the first string is the name within the PDB file, and the second string is the variable name to which the first string will be mapped. To illustrate, the following is part of the Name Map that exists when LEaP is started from the "leaprc" file included in the distribution tape:

```
 ADE  -->  DADE
 :  :
0 ALA  -->  NALA
0 ARG  -->  NARG
   :  :
1 ALA  -->  CALA
1 ARG  -->  CARG
   :  :
1 VAL  -->  CVAL
```

Thus, the residue `ALA` will be mapped to `NALA` if it is the N-terminal residue and `CALA` if it is found at the C-terminus. The above Name Map was produced using the following (edited) command line:

```
> addPdbResMap {
> { 0 ALA NALA }  { 1 ALA CALA }
> { 0 ARG NARG } { 1 ARG CARG }
            :    :
> { 0 VAL NVAL }  { 1 VAL CVAL }
>
            :  :
> { ADE DADE }
            :  :
> }
```

3.6.8. alias

`alias [ string1 [ string2 ] ]`

```
STRING  string1
STRING  string2
```

This command will add or remove an entry to the Alias Table or list entries in the Alias Table. If both strings are present, then string1 becomes the alias to string2, the original command. If only one string is used as an argument, then this string is removed from the Alias Table. If no arguments are given with the command, the current aliases stored in the Alias Table will be listed.

The proposed alias is first checked for conflict with the LEaP commands and it is rejected if a conflict is found. A proposed alias will replace an existing alias with a warning being issued. The alias can stand for more than a single word, but also as an entire string so the user can quickly repeat entire lines of input.

3.6.9. bond

`bond atom1 atom2 [ order ]`

```
ATOM    atom1
ATOM    atom2
STRING  order
```

Create a bond between atom1 and atom2. Both of these ATOMs must be contained by the same UNIT. By specifying "-", "=", "#", or ":" as the optional argument, *order*, the user can specify a single, double, triple, or aromatic bond, respectively. Example:

```
bond trx.32.SG trx.35.SG
```

3.6.10. bondByDistance

`bondByDistance container [ maxBond ]`

```
CONT    container
NUMBER  maxBond
```

Create single bonds between all ATOMs in container that are within maxBond angstroms of each other. If maxBond is not specified then a default distance will be used. This command is especially useful in building molecules. Example:

```
bondByDistance alkylChain
```

3.6.11. center

`center container`

```
UNIT/RESIDUE/ATOM   container
```

Display the coordinates of the geometric center of the ATOMs within container. In the following example, the alanine UNIT found in the amino acid library has been examined by the center command:

```
> center ALA
The center is at: 4.04, 2.80, 0.49
```

3.6.12. charge

`charge container`

```
UNIT/RESIDUE/ATOM   container
```

This command calculates the total charge of the ATOMs within container. The total charges for both standard and, where applicable, perturbed systems are displayed. In the following example, the alanine UNIT found in the amino acid library has been examined by the charge command:

```
> charge ALA
Total unperturbed charge: 0.00
Total perturbed charge:   0.00
```

3.6.13. check

`check unit [ parms ]`

```
UNIT    unit
PARMSET parms
```

This command can be used to check the UNIT for internal inconsistencies that could cause problems when performing calculations. This is a very useful command that should be used before a UNIT is saved with *saveAmberParm or its variants. Currently it checks for the following possible problems:*

- `long bonds`
- `short bonds`

- non-integral total charge of the UNIT.
- missing force field atom types
- close contacts (< 1.5 Å) between nonbonded ATOMs.

The user may collect any missing molecular mechanics parameters in a PARMSET for subsequent editing. In the following example, the alanine UNIT found in the amino acid library has been examined by the *check command:*

```
> check ALA
Checking 'ALA'....
Checking parameters for unit 'ALA'.
Checking for bond parameters.
Checking for angle parameters.
Unit is OK.
```

3.6.14. combine

```
variable = combine  list
```

```
object  variable
LIST    list
```

Combine the contents of the UNITs within list into a single UNIT. The new UNIT is placed in variable. This command is similar to the *sequence* command except it does not link the ATOMs of the UNITs together. In the following example, the input and output should be compared with the example given for the *sequence* command.

```
> tripeptide = combine { ALA GLY PRO }
Sequence: ALA
Sequence: GLY
Sequence: PRO
> desc tripeptide
UNIT name: ALA     !! bug: this should be tripeptide!
Head atom: .R<ALA 1>.A<N 1>
Tail atom: .R<PRO 3>.A<C 13>
Contents:
R<ALA 1>
R<GLY 2>
R<PRO 3>
```

3.6.15. copy

```
newvariable = copy  variable
```

```
object  newvariable
object  variable
```

Creates an exact duplicate of the object variable. Since newvariable is not pointing to the same object as variable, changing the contents of one object will not alter the other

object. Example:

```
> tripeptide = sequence { ALA GLY PRO }
> tripeptideSol = copy tripeptide
> solvateBox tripeptideSol TIP3PBOX 8 2
```

In the above example, tripeptide is a separate object from tripeptideSol and is not solvated. Had the user instead entered

```
> tripeptide = sequence { ALA GLY PRO }
> tripeptideSol = tripeptide
> solvateBox tripeptideSol TIP3PBOX 8 2
```

then both tripeptide and tripeptideSol would be solvated since they would both point to the same object.

3.6.16. createAtom

```
variable = createAtom  name type charge
```

```
ATOM    variable
STRING  name
STRING  type
NUMBER  charge
```

Return a new and empty ATOM with name, type, and charge as its atom name, atom type, and electrostatic point charge. (See the *add* command for an example of the *createAtom* command.)

3.6.17. createParmset

```
variable = createParmset  name
```

```
PARMSET variable
STRING  name
```

Return a new and empty PARMSET with the name "name".

```
> newparms = createParmset pertParms
```

3.6.18. createResidue

```
variable = createResidue  name
```

```
RESIDUE variable
STRING  name
```

Return a new and empty RESIDUE with the name "name". (See the *add* command for an example of the *createResidue* command.)

3.6.19. createUnit

```
variable = createUnit  name
```

```
UNIT    variable
STRING  name
```

Return a new and empty UNIT with the name "name". (See the *add* command for an example of the *createUnit* command.)

3.6.20. deleteBond

```
deleteBond atom1 atom2
```

```
ATOM    atom1
ATOM    atom2
```

Delete the bond between the ATOMs atom1 and atom2. If no bond exists, an error will be displayed.

3.6.21. desc

```
desc variable
```

```
object  variable
```

Print a description of the object. In the following example, the alanine UNIT found in the amino acid library has been examined by the *desc* command:

```
> desc ALA
UNIT name: ALA
Head atom: .R<ALA 1>.A<N 1>
Tail atom: .R<ALA 1>.A<C 9>
Contents:
R<ALA 1>
```

Now, the *desc* command is used to examine the first residue (1) of the alanine UNIT:

```
> desc ALA.1
RESIDUE name: ALA
RESIDUE sequence number: 1
Type: protein
Connection atoms:
Connect atom 0: A<N 1>
Connect atom 1: A<C 9>
Contents:
A<N 1>
A<HN 2>
A<CA 3>
A<HA 4>
A<CB 5>
```

```
A<HB1 6>
A<HB2 7>
A<HB3 8>
A<C 9>
A<O 10>
```

Next, we illustrate the desc command by examining the ATOM *N* of the first residue (1) of the alanine UNIT:

```
> desc ALA.1.N
ATOM
Name:    N
Type:    N
Charge:  -0.463
Element: N
Atom flags: 20000|posfxd- posblt- posdrn- sel- pert-
notdisp- tchd- posknwn+ int - nmin- nbld-
Atom position: 3.325770, 1.547909, -0.000002
Atom velocity: 0.000000, 0.000000, 0.000000
Bonded to .R<ALA 1>.A<HN 2> by a single bond.
Bonded to .R<ALA 1>.A<CA 3> by a single bond.
```

Since the N ATOM is also the first atom of the ALA residue, the following command will give the same output as the previous example:

```
> desc ALA.1.1
```

3.6.22. edit

`edit unit`

```
UNIT    unit
```

In xleap this command creates a Unit Editor that contains the UNIT unit. The user can view and edit the contents of the UNIT using the mouse. The command causes a copy of the object to be edited. If the object that the user wants to edit is `null`, then the edit command assumes that the user wants to edit a new UNIT with a single RESIDUE within it. PARMSETs can also be edited. In tleap this command prints an error message.

3.6.23. groupSelectedAtoms

`groupSelectedAtoms unit name`

```
UNIT    unit
STRING  name
```

Create a group within unit with the name, "name", using all of the ATOMs within the UNIT that are selected. If the group has already been defined then overwrite the old group. The *desc* command can be used to list groups. Example:

```
groupSelectedAtoms TRP sideChain
```

An expression like "TRP@sideChain" returns a LIST, so any commands that require LIST ’s can take advantage of this notation. After assignment, one can access groups using the "@" notation. Examples:

```
select TRP@sideChain


center TRP@sideChain
```

The latter example will calculate the center of the atoms in the "sideChain" group. (see the *select* command for a more detailed example.)

3.6.24. help

```
help [string]

STRING  string
```

This command prints a description of the command in string. If the STRING is not given then a list of help topics is provided.

3.6.25. impose

```
impose unit seqlist internals

UNIT    unit
LIST    seqlist
LIST    internals
```

The impose command allows the user to impose internal coordinates on the UNIT. The list of RESIDUEs to impose the internal coordinates upon is in seqlist. The internal coordinates to impose are in the LIST internals.

The command works by looking into each RESIDUE within the UNIT that is listed in the seqlist argument and attempts to apply each of the internal coordinates within internals. The seqlist argument is a LIST of NUMBERs that represent sequence numbers or ranges of sequence numbers. Ranges of sequence numbers are represented by two element LISTs that contain the first and last sequence number in the range. The user can specify sequence number ranges that are larger than what is found in the UNIT. For example, the range { 1 999 } represents all RESIDUEs in a 200 RESIDUE UNIT.

The internals argument is a LIST of LISTs. Each sublist contains a sequence of ATOM names which are of type STRING followed by the value of the internal coordinate. An example of the impose command would be:

```
impose peptide { 1 2 3 } {
{ N CA C N -40.0 }
{ C N CA C -60.0 }
}
```

This would cause the RESIDUE with sequence numbers 1, 2, and 3 within the UNIT peptide to assume an alpha helical conformation. The command

```
impose peptide { 1 2 { 5 10 } 12 } {
 { CA CB 5.0 } }
```

will impose on the residues with sequence numbers 1, 2, 5, 6, 7, 8, 9, 10, and 12 within the UNIT peptide a bond length of 5.0 angstroms between the alpha and beta carbons. RESIDUEs without an ATOM named CB (like glycine) will be unaffected.

Three types of conformational change are supported: bond length changes, bond angle changes, and torsion angle changes. If the conformational change involves a torsion angle, then all dihedrals around the central pair of atoms are rotated. The entire list of internals are applied to each RESIDUE.

3.6.26. list

List all of the variables currently defined. To illustrate, the following (edited) output shows the variables defined when LEaP is started from the leaprc file included in the distribution tape:

```
> list
A
ACE      ALA
ARG      ASN
 :   :
VAL      W
WAT      Y
```

3.6.27. loadAmberParams

```
variable = loadAmberParams  filename
```

```
PARMSET variable
STRING  filename
```

Load an AMBER format parameter set file and place it in variable. All interactions defined in the parameter set will be contained within variable. This command causes the loaded parameter set to be included in LEaP 's list of parameter sets that are searched when parameters are required. General proper and improper torsion parameters are modified during the command execution with the LEaP general type "?" replacing the AMBER general type "X".

```
> parm91 = loadAmberParams parm91X.dat
> saveOff parm91 parm91.lib
Saving parm91.
```

3.6.28. loadAmberPrep

`loadAmberPrep filename [ prefix ]`

```
STRING  filename
STRING  prefix
```

This command loads an AMBER PREP input file. For each residue that is loaded, a new UNIT is constructed that contains a single RESIDUE and a variable is created with the same name as the name of the residue within the PREP file. If the optional argument prefix is provided it will be prefixed to each variable name; this feature is used to prefix UATOM residues, which have the same names as AATOM residues with the string "U" to distinguish them. Let us imagine that the following AMBER PREP input file exists:

```
   0  0  2
     Crown Fragment A
cra.res
CRA  INT 0
CORRECT  NOMIT DU  BEG
0.0
1  DUMM  DU M  0  0  0     0.      0.      0.
2  DUMM  DU M  0  0  0     1.000   0.      0.
3  DUMM  DU M  0  0  0     1.000  90.      0.
4  C1    CT M  0  0  0     1.540 112.    169.
5  H1A   HC E  0  0  0     1.098 109.47  -110.0
6  H1B   HC E  0  0  0     1.098 109.47   110.0
7  O2    OS M  0  0  0     1.430 112.    -72.
8  C3    CT M  0  0  0     1.430 112.    169.
9  H3A   HC E  0  0  0     1.098 109.47  -49.0
10 H3B   HC E  0  0  0     1.098 109.47   49.0

CHARGE
 0.2442  -0.0207  -0.0207  -0.4057  0.2442
 -0.0207  -0.0207

DONE
STOP
```

This fragment can be loaded into LEaP using the following command:

```
> loadAmberPrep cra.in
Loaded UNIT: CRA
```

3.6.29. loadOff

`loadOff filename`

```
STRING  filename
```

This command loads the OFF library within the file named filename. All UNITs and

PARMSETs within the library will be loaded. The objects are loaded into LEaP under the variable names the objects had when they were saved. Variables already in existence that have the same names as the objects being loaded will be overwritten. Any PARMSETs loaded using this command are included in LEaP 's library of PARMSETs that is searched whenever parameters are required (The old AMBER format is used for PARMSETs rather than the OFF format in the default configuration). Example command line:

```
> loadOff parm91.lib
Loading library: parm91.lib
Loading: PARAMETERS
```

3.6.30. loadMol2

```
variable = loadMol2 filename
```

```
STRING  filename
object  variable
```

Load a Sybyl MOL2 format file in a UNIT. This command is very much like *loadOff*, except that it only creates a single UNIT.

3.6.31. loadPdb

```
variable = loadPdb filename
```

```
STRING  filename
object  variable
```

Load a Protein Databank format file with the file name filename. The sequence numbers of the RESIDUEs will be determined from the order of residues within the PDB file ATOM records. This function will search the variables currently defined within LEaP for variable names that map to residue names within the ATOM records of the PDB file. If a matching variable name is found then the contents of the variable are added to the UNIT that will contain the structure being loaded from the PDB file. Adding the contents of the matching UNIT into the UNIT being constructed means that the contents of the matching UNIT are copied into the UNIT being built and that a bond is created between the connect0 ATOM of the matching UNIT and the connect1 ATOM of the UNIT being built. The UNITs are combined in the same way UNITs are combined using the sequence command. As atoms are read from the ATOM records their coordinates are written into the correspondingly named ATOMs within the UNIT being built. If the entire residue is read and it is found that ATOM coordinates are missing, then external coordinates are built from the internal coordinates that were defined in the matching UNIT. This allows LEaP to build coordinates for hydrogens and lone-pairs which are not specified in PDB files.

```
> crambin = loadPdb 1crn
Loading PDB file
Matching PDB residue names to LEaP variables.
Mapped residue THR, term: 0, seq. number: 0 to: NTHR.
Residue THR, term: M, seq. number: 1 was not
found in name map.
```

```
Residue CYS, term: M, seq. number: 2 was not
found in name map.
Residue CYS, term: M, seq. number: 3 was not
found in name map.
Residue PRO, term: M, seq. number: 4 was not
found in name map.
:               :               :
Residue TYR, term: M, seq. number: 43 was not
found in name map.
Residue ALA, term: M, seq. number: 44 was not
found in name map.
Mapped residue ASN, term: 1, seq. number: 45 to: CASN.
Joining NTHR - THR
Joining THR - CYS
Joining CYS - CYS
Joining CYS - PRO
:               :               :
Joining ASP - TYR
Joining TYR - ALA
Joining ALA - CASN
```

The above edited listing shows the use of this command to load a PDB file for the protein crambin. Several disulfide bonds are present in the protein and these bonds are indicated in the PDB file. The loadPdb command, however, cannot read this information from the PDB file. It is necessary for the user to explicitly define disulfide bonds using the *bond* command.

3.6.32. loadPdbUsingSeq

`loadPdbUsingSeq filename unitlist`

```
STRING  filename
LIST    unitlist
```

This command reads a Protein Data Bank format file from the file named filename. This command is identical to *loadPdb* except it does not use the residue names within the PDB file. Instead the sequence is defined by the user in unitlist. For more details see *loadPdb.*

```
> peptSeq = { UALA UASN UILE UVAL UGLY }
> pept = loadPdbUsingSeq pept.pdb peptSeq
```

In the above example, a variable is first defined as a LIST of united atom RESIDUEs. A PDB file is then loaded, in this sequence order, from the file "pept.pdb".

3.6.33. logFile

`logFile filename`

```
STRING  filename
```

This command opens the file with the file name filename as a log file. User input and all

output is written to the log file. Output is written to the log file as if the verbosity level were set to 2. An example of this command is:

```
> logfile /disk/howard/leapTrpSolvate.log
```

3.6.34. measureGeom

`measureGeom atom1 atom2 [ atom3 [ atom4 ] ]`

```
ATOM    atom1
ATOM    atom2
ATOM    atom3
ATOM    atom4
```

Measure the distance, angle, or torsion between two, three, or four ATOMs, respectively.

In the following example, we first describe the RESIDUE ALA of the ALA UNIT in order to find the identity of the ATOMs. Next, the measureGeom command is used to determine a distance, simple angle, and a dihedral angle. As shown in the example, the ATOMs may be identified using atom names or numbers.

```
> desc ALA.ALA
RESIDUE name: ALA
RESIDUE sequence number: 1
Type: protein
Connection atoms:
Connect atom 0: A<N 1>
Connect atom 1: A<C 9>
Contents:
A<N 1>
A<HN 2>
A<CA 3>
A<HA 4>
A<CB 5>
A<HB1 6>
A<HB2 7>
A<HB3 8>
A<C 9>
A<O 10>
> measureGeom ALA.ALA.1 ALA.ALA.3
Distance: 1.45 angstroms
> measureGeom ALA.ALA.1 ALA.ALA.3 ALA.ALA.5
Angle: 111.10 degrees
> measureGeom ALA.ALA.N ALA.ALA.CA ALA.ALA.C ALA.ALA.O
Torsion angle: 0.00 degrees
```

3.6.35. quit

Quit the LEaP program.

3.6.36. remove

`remove a b`

```
CONT    a
CONT    b
```

Remove the object b from the object a. If b is not contained by a then an error message will be displayed. This command is used to remove ATOMs from RESIDUEs, and RESIDUEs from UNITs. If the object represented by b is not referenced by some variable name then it will be destroyed.

```
> dipeptide = combine { ALA GLY }
Sequence: ALA
Sequence: GLY
> desc dipeptide
UNIT name: ALA     !! bug: this should be dipeptide!
Head atom: .R<ALA 1>.A<N 1>
Tail atom: .R<GLY 2>.A<C 6>
Contents:
R<ALA 1>
R<GLY 2>
> remove dipeptide dipeptide.2
> desc dipeptide
UNIT name: ALA     !! bug: this should be dipeptide!
Head atom: .R<ALA 1>.A<N 1>
Tail atom: null
Contents:
R<ALA 1>
```

3.6.37. saveAmberParm

`saveAmberParm unit topologyfilename coordinatefilename`

```
UNIT    unit
STRING  topologyfilename
STRING  coordinatefilename
```

Save the AMBER topology and coordinate files for the UNIT into the files named topologyfilename and coordinatefilename respectively. This command will cause LEaP to search its list of PARMSETs for parameters defining all of the interactions between the ATOMs within the UNIT. This command produces topology files and coordinate files that are identical in format to those produced by AMBER PARM and can be read into AMBER for calculations. The output of this operation can be used for minimizations, dynamics, and thermodynamic integration calculations.

In the following example, the topology and coordinates from the all_amino94.lib UNIT ALA are generated:

```
> saveamberparm ALA ala.top ala.crd
Building topology.
Building atom parameters.
Building bond parameters.
Building angle parameters.
Building proper torsion parameters.
Building improper torsion parameters.
Building H-Bond parameters.
```

3.6.38. saveAmberParmPol

`saveAmberParmPol unit topologyfilename coordinatefilename`

```
UNIT    unit
STRING  topologyfilename
STRING  coordinatefilename
```

Like saveAmberParm, but includes atomic polarizabilities in the topology file for use with IPOL=1 in Sander. The polarizabilities are according to atom type, and are defined in the 'mass' section of the *parm.dat* or *frcmod* file.

3.6.39. saveAmberParmPert

`saveAmberParmPert unit topologyfilename coordinatefilename`

```
UNIT    unit
STRING  topologyfilename
STRING  coordinatefilename
```

This command is the same as *saveAmberParm*, except a perturbation topology file is written instead of a plain minimization/dynamics one.

Save the AMBER topology and coordinate files for the UNIT into the files named topologyfilename and coordinatefilename respectively. This command will cause LEaP to search its list of PARMSETs for parameters defining all of the interactions between the ATOMs within the UNIT.

```
> saveAmberParmPert pert pert.leap.top pert.leap.crd
Building topology.
Building atom parameters.
Building bond parameters.
Building angle parameters.
Building proper torsion parameters.
Building improper torsion parameters.
Building H-Bond parameters.
```

3.6.40. saveAmberParmPolPert

```
saveAmberParmPolPert unit topologyfilename coordinatefile-
name
```

```
UNIT    unit
STRING  topologyfilename
STRING  coordinatefilename
```

Like saveAmberParmPert, but includes atomic polarizabilities in the topology file for use with IPOL=1 in Gibbs. The polarizabilities are according to atom type, and are defined in the 'mass' section of the parm.dat or frcmod file.

3.6.41. saveOff

```
saveOff object filename
```

```
object  object
STRING  filename
```

The saveOff command allows the user to save UNITs and PARMSETs to a file named *filename*. The file is written using the Object File Format (off) and can accommodate an unlimited number of uniquely named objects. The names by which the objects are stored are the variable names specified in the argument of this command. If the file *filename* already exists then the new objects will be added to the file. If there are objects within the file with the same names as objects being saved then the old objects will be overwritten. The argument object can be a single UNIT, a single PARMSET, or a LIST of mixed UNITs and PARMSETs. (See the *add* command for an example of the *saveOff* command.)

3.6.42. savePdb

```
savePdb unit filename
```

```
UNIT    unit
STRING  filename
```

Write UNIT to the file *filename* as a PDB format file. In the following example, the PDB file from the "all_amino94.lib" UNIT ALA is generated:

```
> savepdb ALA ala.pdb
```

3.6.43. scaleCharges

```
scaleCharges container scale_factor
```

```
UNIT/RESIDUE/ATOM   container
NUMBER              scale_factor
```

This command scales the charges in the object by _scale_factor_, which must be > 0. It is useful for building systems for use with polarizable atoms, e.g.

```
> x = copy solute
> scaleCharges x 0.8
> y = copy TIP3PBOX
> scalecharges y 0.875
> solvatebox x y 10
> saveamberparmpol x x.top x.crd
```

3.6.44. sequence

```
variable = sequence  list
```

```
UNIT    variable
LIST    list
```

The sequence command is used to create a new UNIT by combining the contents of a LIST of UNITs. The first argument is a LIST of UNITs. A new UNIT is constructed by taking each UNIT in the sequence in turn and copying its contents into the UNIT being constructed. As each new UNIT is copied, a bond is created between the tail ATOM of the UNIT being constructed and the head ATOM of the UNIT being copied, if both connect ATOMs are defined. If only one is defined, a warning is generated and no bond is created. If neither connection ATOM is defined then no bond is created. As each RESIDUE is copied into the UNIT being constructed it is assigned a sequence number which represents the order the RESIDUEs are added. Sequence numbers are assigned to the RESIDUEs so as to maintain the same order as was in the UNIT before it was copied into the UNIT being constructed. This command builds reasonable starting coordinates for all ATOMs within the UNIT; it does this by assigning internal coordinates to the linkages between the RESIDUEs and building the external coordinates from the internal coordinates from the linkages and the internal coordinates that were defined for the individual UNITs in the sequence.

```
> tripeptide = sequence { ALA GLY PRO }
Sequence: ALA
Sequence: GLY
Joining ALA - GLY
Sequence: PRO
Joining GLY - PRO
> desc tripeptide
UNIT name: ALA     !! bug: this should be tripeptide!
Head atom: .R<ALA 1>.A<N 1>
Tail atom: .R<PRO 3>.A<C 13>
Contents:
R<ALA 1>
R<GLY 2>
R<PRO 3>
```

3.6.45. set

```
set default variable value

      STRING  variable
      STRING  value
```

or

```
set container parameter object

      CONT    container
      STRING  parameter
      object  object
```

This command sets the values of some global parameters (when the first argument is "default") or sets various parameters associated with container. The following parameters can be set within LEaP:

For "default" parameters

PBradii
: Set to "mbondi" to use modified Bondi radii (where the hydrogens are modified from the Bondi values). Here the radius of hydrogen bonded to oxygen or sulfur is set to 0.8; hydrogen bonded to carbon is 1.3; hydrogen bonded to nitrogen is 1.3. These parameters are the default, and are those used by Tsui & Case [42], and are the recommended ones when *igb* = 1 in the *sander* input. The code in Amber (version 6) used values like the "mbondi" values, except that the radius for hydrogen bonded to nitrogen was 1.2; you can use the "amber6" keyword for *PBradii* to use these earlier values [43], but this is only recommended if you want to check results against those from Amber6, or if you need to extend a simulation started with the earlier parameters.

  Set to "mbondi2" to use another set of modified Bondi radii and Tinker screening parameters for generalized Born calculations. These values are recommended when *igb* = 2 and *igb* = 5 in the *sander* input. Here only the radius of hydrogen bonded to nitrogen is increased to 1.3. Original bondi radii, set by "bondi" also perform reasonably well with this GB model; see [8] for details.

  The values specified above are put into the RADII and SCREENING sections of the *prmtop* file, and could be edited by hand from there if further changes were desired.

OldPrmtopFormat
: If set to "on", the saveAmberParm command will write a prmtop file in the format used in Amber6 and before; if set to "off" (the default), it will use the new format.

Dielectric
: If set to "distance" (the default), electrostatic calculations in LEaP will use a distance-dependent dielectric; if set to "constant", and constant dielectric will be used.

PdbWriteCharges
: If set to "on", atomic charges will be placed in the "B-factor" field of pdb files saved with the savePdb command; if set to "off" (the default), no such

charges will be written.

PdbWriteRadii If set to "on", atomic radii will be placed in the " " field pdb files saved with the savePdb command; if set to "off" (the default), no such radii will be written.

For ATOMs:

name A unique STRING descriptor used to identify ATOMs.

type This is a STRING property that defines the AMBER force field atom type.

charge The charge property is a NUMBER that represents the ATOM's electrostatic point charge to be used in a molecular mechanics force field.

position This property is a LIST of NUMBERs containing three values: the (X, Y, Z) Cartesian coordinates of the ATOM.

pertName The STRING is a unique identifier for an ATOM in its final state during a free energy calculation.

pertType The STRING is the AMBER force field atom type of a perturbed ATOM.

pertCharge This NUMBER represents the *difference* between the final electrostatic point charge on an ATOM and its unperturbed (or initial) charge.

For RESIDUEs:

connect0 This defines an ATOM that is used in making links to other RESIDUEs. In UNITs containing single RESIDUEs, the RESIDUEsS connect0 ATOM is usually defined as the UNIT's head ATOM.

connect1 This is an ATOM property which defines an ATOM that is used in making links to other RESIDUEs. In UNITs containing single RESIDUEs, the RESIDUEsS connect1 ATOM is usually defined as the UNIT's tail ATOM.

connect2 This is an ATOM property which defines an ATOM that can be used in making links to other RESIDUEs. In amino acids, the convention is that this is the ATOM to which disulfide bridges are made.

restype This property is a STRING that represents the type of the RESIDUE. Currently, it can have one of the following values: "undefined", "solvent", "protein", "nucleic", or "saccharide".

name This STRING property is the RESIDUE name.

For UNITs:

head Defines the ATOM within the UNIT that is connected when UNITs are joined together: the tail ATOM of one UNIT is connected to the head ATOM of the subsequent UNIT in any sequence.

tail Defines the ATOM within the UNIT that is connected when UNITs are joined together: the tail ATOM of one UNIT is connected to the head ATOM of the subsequent UNIT in any sequence.

box The property defines the bounding box of the UNIT. If it is defined as `null` then no bounding box is defined. If the value is a single NUMBER then the bounding box will be defined to be a cube with each side being NUMBER of angstroms across. If the value is a LIST then it must be a

LIST containing three numbers, the lengths of the three sides of the bounding box.

cap — The property defines the solvent cap of the UNIT. If it is defined as `null` then no solvent cap is defined. If the value is a LIST then it must contain four numbers, the first three define the Cartesian coordinates (X, Y, Z) of the origin of the solvent cap in angstroms, the fourth NUMBER defines the radius of the solvent cap in angstroms.

3.6.46. setBox

```
setBox unit   vdw OR centers  [ buffer OR buffer_xyz_list ]

      UNIT    unit
```

The setBox command adds a periodic box to the UNIT, turning it into a periodic system for the simulation programs. It does not add any solvent to the system. The choice of "vdw" or "centers" determines whether the box encloses the entire atoms or just the atom centers - use "centers" if the system has been previously equilibrated as a periodic box. See the solvateBox command for a description of the buffer variable, which extends either type of box by an arbitrary amount.

3.6.47. solvateBox

```
solvateBox solute solvent buffer [ iso ] [ closeness ]

      UNIT    solute
      UNIT    solvent
      object  buffer
      NUMBER  closeness
```

The *solvateBox* command creates a rectangular parallelepiped solvent box around the solute UNIT. The solute UNIT is modified by the addition of solvent RESIDUEs. (For most liquid state simulations, the *solvateOct* command discussed below is probably a better choice.)

The normal choice for a TIP3 _solvent_ UNIT is TIP3PBOX, which is a snapshot from a room-temperature equilibration for this model. If you want to solvate with other water models, look for a "BOX" with that model (*e.g.* TIP4PBOX, POLBOX, SPCBOX, TIP5PBOX). If you need to use a water model other than these, try the following: *(a)* solvate the system with TIP3PBOX, using the default TIP3 model; *(b)* use *ambpdb* to convert your *prmtop* file to Brookhaven format; *(c)* restart LEaP, choose the water model you want (instructions are in the Database chapter), then use loadPdb to bring back in the system you have created. The issue a setBox command to provide a box. It is also best to manually edit the resulting *prmcrd* file so that its last line (which contains the box information) matches what you had from the original run with TIP3PBOX; this corrects a glitch which prevents setBox from making as good a box as does solvateBox or solvateOct.

Note that equilibration will always be required to bring the artificial box to reasonable density, since Van der Waals voids remain due to the impossibility of natural packing of

solvent around the solute and at the edges of the box. First, equilibrate the system at constant volume to the temperature you want, then turn on constant pressure to adjust the system density to the desired value.

The solvent UNIT is copied and repeated in all three spatial directions to create a box containing the entire solute and a buffer zone defined by the buffer argument. The buffer argument defines the distance, in angstroms, between the wall of the box and the closest ATOM in the solute. If the buffer argument is a single NUMBER, then the buffer distance is the same for the x, y, and z directions, unless the 'iso' option is used to make the box cubic, with the shortest box clearance = buffer. If the buffer argument is a LIST of three NUMBERs, then the NUMBERs are applied to the x, y, and z axes respectively. As the larger box is created and superimposed on the solute, solvent molecules overlapping the solute are removed.

The optional closeness parameter can be used to control how close, in angstroms, solvent ATOMs can come to solute ATOMs. The default value of the closeness argument is 1.0. Smaller values allow solvent ATOMs to come closer to solute ATOMs. The criterion for rejection of overlapping solvent RESIDUEs is if the distance between any solvent ATOM to the closest solute ATOM is less than the sum of the ATOMs VANDERWAAL distances multiplied by the closeness argument.

This command modifies the _solute_ UNIT in several ways. First, the coordinates of the ATOMs are modified to move the center of a box enclosing the Van der Waals radii of the atoms to the origin. Secondly, the UNIT is modified by the addition of _solvent_ RESIDUEs copied from the _solvent_ UNIT. Finally, the box parameter of the new system (still named for the _solute_) is modified to reflect the fact that a periodic, rectilinear solvent box has been created around it.

In this example, it is assumed that the file solvents.lib, containing TIP3PBOX, has been loaded already (as is done by the default leaprc):

```
>> mol = loadpdb my.pdb
>> solvateBox sol TIP3PBOX 10
  Solute vdw bounding box:              7.512 12.339 12.066
  Total bounding box for atom centers:  27.512 32.339 32.066
  Solvent unit box:                     18.774 18.774 18.774
  Total vdw box size:                   30.995 35.538 35.416 angstroms.
  Total mass 14470.768 amu,  Density 0.616 g/cc
  Added 785 residues.
```

Again, note that the density of 0.601 g/cc points to the need for constant pressure equilibration. (See the discussion of equilibration in the Q&A section of the amber web.)

3.6.48. solvateCap

```
solvateCap solute solvent position radius [ closeness ]
```

```
UNIT    solute
UNIT    solvent
object  position
NUMBER  radius
NUMBER  closeness
```

The solvateCap command creates a solvent cap around the solute UNIT. The solute UNIT is modified by the addition of solvent RESIDUEs. The solvent box will be repeated in all three spatial directions to create a large solvent sphere with a radius of radius angstroms.

The position argument defines where the center of the solvent cap is to be placed. If position is a RESIDUE, ATOM, or a LIST of UNITs, RESIDUEs, or ATOMs, then the geometric center of the ATOMs within the object will be used as the center of the solvent cap sphere. If position is a LIST containing three NUMBERs, then the position argument will be treated as a vector that defines the position of the solvent cap sphere center.

The optional closeness parameter can be used to control how close, in angstroms, solvent ATOMs can come to solute ATOMs. The default value of the closeness argument is 1.0. Smaller values allow solvent ATOMs to come closer to solute ATOMs. The criterion for rejection of overlapping solvent RESIDUEs is if the distance between any solvent ATOM to the closest solute ATOM is less than the sum of the ATOMs VANDERWAAL's distances multiplied by the closeness argument.

This command modifies the solute UNIT in several ways. First, the UNIT is modified by the addition of solvent RESIDUEs copied from the solvent UNIT. Secondly, the cap parameter of the UNIT solute is modified to reflect the fact that a solvent cap has been created around the solute.

```
>> mol = loadpdb my.pdb
>> solvateCap mol TIP3PBOX mol.2.CA 8.0 2.0
Added 3 residues.
```

3.6.49. solvateDontClip

```
solvateDontClip solute solvent buffer [ closeness ]
```

```
UNIT    solute
UNIT    solvent
object  buffer
NUMBER  closeness
```

This command is identical to the *solvateBox* command except that the solvent box that is created is not clipped to the boundary of the buffer region. This command forms larger solvent boxes than does *solvateBox* because it does not cause solvent that is outside the buffer region to be discarded. This helps to preserve the periodic structure of properly constructed solvent boxes, preventing hot-spots from forming.

```
>> mol = loadpdb my.pdb
>> solvateDontClip mol TIP3PBOX 10
  Solute vdw bounding box:              7.512 12.339 12.066
  Total bounding box for atom centers:  27.512 32.339 32.066
  Solvent unit box:                     18.774 18.774 18.774
  Total vdw box size:                   41.120 40.899 41.075 angstroms.
  Total mass 30595.088 amu,  Density 0.735 g/cc
  Added 1680 residues.
```

Note the larger number of waters added, compared to solvateBox; in the case of this solute and choice of buffer, the overall box size is increased by about 10 angstroms in each direction.

3.6.50. solvateOct

```
solvateOct solute solvent buffer [aniso] [ closeness ]
```

```
UNIT                    _solute_
UNIT                    _solvent_
object                  _buffer_
NUMBER                  _closeness_
```

The solvateOct command is the same as solvateBox, except the corners of the box are sliced off, resulting in a truncated octahedron, which typically gives a more uniform distribution of solvent around the solute. In solvateOct, when a LIST is given for the buffer argument, four numbers are given instead of three, where the fourth is the diagonal clearance. If 0.0 is given as the fourth number, the diagonal clearance resulting from the application of the x,y,z clearances is reported. If a non-0 value is given, this may require scaling up the other clearances, which is also reported.

Unless the 'aniso' option is used, an isometric truncated octahedron is produced and rotated to an orientation used by the *sander* PME code. (Note: don't use the 'aniso' option unless you are sure you know what you are doing; it is only there for expert backward compatibility, and probably has no real use anymore.)

3.6.51. solvateShell

```
solvateShell solute solvent thickness [ closeness ]
```

```
UNIT    solute
UNIT    solvent
NUMBER  thickness
NUMBER  closeness
```

The *solvateShell* command adds a solvent shell to the solute UNIT. The resulting solute/solvent UNIT will be irregular in shape since it will reflect the contours of the solute. The solute UNIT is modified by the addition of solvent RESIDUEs. The solvent box will be repeated in three directions to create a large solvent box that can contain the entire solute and a shell thickness angstroms thick. The solvent RESIDUEs are then added to the solute UNIT if they lie within the shell defined by thickness and do not overlap with the solute ATOMs. The optional closeness parameter can be used to control how close solvent ATOMs can come to solute ATOMs. The default value of the closeness argument is 1.0. Please see the *solvateBox* command for more details on the closeness parameter.

```
>> mol = loadpdb my.pdb
>> solvateShell mol TIP3PBOX 8.0
  Solute vdw bounding box:              7.512 12.339 12.066
  Total bounding box for atom centers:  23.512 28.339 28.066
  Solvent unit box:                     18.774 18.774 18.774
```

```
Added 147 residues.
```

3.6.52. source

`source filename`

```
STRING  filename
```

This command executes commands within a text file. To display the commands as they are read, see the *verbosity* command.

3.6.53. transform

`transform atoms, matrix`

```
CONT    atoms
LIST    matrix
```

Transform all of the ATOMs within atoms by the (3×3) or (4×4) matrix represented by the nine or sixteen NUMBERs in the LIST of LISTs *matrix*. The general matrix looks like:

```
r11 r12 r13 -tx
r21 r22 r23 -ty
r31 r32 r33 -tz
0   0   0   1
```

The matrix elements represent the intended symmetry operation. For example, a reflection in the (x, y) plane would be produced by the matrix:

```
1   0    0
0   1    0
0   0   -1
```

This reflection could be combined with a six angstrom translation along the x-axis by using the following matrix.

```
1   0    0  6
0   1    0  0
0   0   -1  0
0   0    0  1
```

In the following example, wrB is transformed by an inversion operation:

```
transform wrpB {
 { -1  0  0  }
 {  0 -1  0  }
 {  0  0 -1  }
 }
```

3.6.54. translate

```
translate atoms direction
```

```
CONT    atoms
LIST    direction
```

Translate all of the ATOMs within atoms by the vector defined by the three NUMBERs in the LIST *direction*.

Example:

```
translate wrpB { 0  0 -24.53333 }
```

3.6.55. verbosity

```
verbosity level
```

```
NUMBER  level
```

This command sets the level of output that LEaP provides the user. A value of 0 is the default, providing the minimum of messages. A value of 1 will produce more output, and a value of 2 will produce all of the output of level 1 and display the text of the script lines executed with the *source* command. The following line is an example of this command:

```
> verbosity 2
Verbosity level: 2
```

3.6.56. zMatrix

```
zMatrix object zmatrix
```

```
CONT    object
LIST    matrix
```

The *zMatrix* command is quite complicated. It is used to define the external coordinates of ATOMs within object using internal coordinates. The second parameter of the *zMatrix* command is a LIST of LISTs; each sub-list has several arguments:

```
{ a1 a2 bond12 }
```

This entry defines the coordinate of a1 by placing it bond12 angstroms along the x-axis from ATOM a2. If ATOM a2 does not have coordinates defined then ATOM a2 is placed at the origin.

```
{ a1 a2 a3 bond12 angle123 }
```

This entry defines the coordinate of a1 by placing it bond12 angstroms away from ATOM a2 making an angle of angle123 degrees between a1, a2 and a3. The angle is measured in a right hand sense and in the x-y plane. ATOMs a2 and a3 must have coordinates

defined.

```
{ a1 a2 a3 a4 bond12 angle123 torsion1234 }
```

This entry defines the coordinate of a1 by placing it bond12 angstroms away from ATOM a2, creating an angle of angle123 degrees between a1, a2, and a3, and making a torsion angle of torsion1234 between a1, a2, a3, and a4.

```
{ a1 a2 a3 a4 bond12 angle123 angle124 orientation }
```

This entry defines the coordinate of a1 by placing it bond12 angstroms away from ATOM a2, making angles angle123 between ATOMs a1, a2, and a3, and angle124 between ATOMs a1, a2, and a4. The argument orientation defines whether the ATOM a1 is above or below a plane defined by the ATOMs a2, a3, and a4. If orientation is positive then a1 will be placed in such a way so that the inner product of (a3-a2) cross (a4-a2) with (a1-a2) is positive. Otherwise a1 will be placed on the other side of the plane. This allows the coordinates of a molecule like fluoro-chloro-bromo-methane to be defined without having to resort to dummy atoms.

The first arguments within the *zMatrix* entries (a1, a2, a3, a4) are either ATOMs or STRINGs containing names of ATOMs within object. The subsequent arguments are all NUMBERs. Any ATOM can be placed at the a1 position, even those that have coordinates defined. This feature can be used to provide an endless supply of dummy atoms, if they are required. A predefined dummy atom with the name "*" (a single asterisk, no quotes) can also be used.

There is no order imposed in the sub-lists. The user can place sub-lists in arbitrary order, as long as they maintain the requirement that all atoms a2, a3, and a4 must have external coordinates defined, except for entries that define the coordinate of an ATOM using only a bond length. (See the *add* command for an example of the *zMatrix* command.)

4. Antechamber

This is a set of tools to generate "prep" input files for organic molecules, which can then be read into LEaP. The Antechamber suite was written by Junmei Wang, and is designed to be used in conjunction with the "general AMBER force field (GAFF)" (*gaff.dat*) [4].

Molecular mechanics are the key component in the armamentarium used by computational chemists for rational drug design and many other tasks. Force fields are the cornerstone of molecular mechanics. A successful force field for drug design should work well both for biological macromolecules and the organic molecules. The AMBER force fields have built up a good reputation for its performance in studies of proteins and nucleic acids. However, the fact that AMBER has had only limited parameters for organic molecules has kept it from being widely used in ligand-binding or drug design applications. Antechamber is based on a new, general AMBER force field (GAFF) that covers most pharmaceutical molecules, and which is as compatible as possible with the traditional AMBER force fields.

Like the traditional AMBER force fields, GAFF uses a simple harmonic function form for bonds and angles. Unlike the traditional AMBER force fields, atom types in GAFF are more general and cover most of the organic chemical space. In total there are 33 basic atom types and 22 special atom types. The charge methods used in GAFF can be HF/6-31G* RESP and AM1-BCC [44]. All of the force field parameterizations were carried out with HF/6-31G* RESP charges. However, in most cases, AM1-BCC, which was parameterized to reproduce HF/6-31G* RESP charges, is recommended in the massive calculations because of its simplicity and efficiency.

The van der Waals parameters are the same as those used by the traditional AMBER force fields. The equilibrium bond lengths and bond angles came from statistics derived from the Cambridge Structural Database, and *ab initio* calculations at the MP2/6-31G* level. The force constants for bonds and angles were estimated using empirical models, and the parameters in these models were trained using the force field parameters in the traditional AMBER force fields. General torsional angle parameters were extensively applied in order to reduce the huge number of torsional angle parameters to be derived. The force constants and phase angles in the torsional angle parameters were optimized using our PARMSCAN package [45], with an aim to reproduce the rotational profiles depicted by high-level *ab initio* calculations [geometry optimizations at the MP2/6-31G* level, followed by single point calculations at MP4/6-311G(d,p)].

By design, GAFF is a complete force field (so that missing parameters rarely occur), it covers almost all the organic chemical space that is made up of C, N, O, S, P, H, F, Cl, Br and I. Moreover, GAFF is totally compatible to the AMBER macromolecular force fields. We believe that the combination of GAFF with AMBER macromolecular force fields will provide an useful molecular mechanical tool for rational drug design, especially in binding free energy calculations and molecular docking studies.

As an auxiliary module in AMBER software packages, antechamber is devoted to build up the bridge between the force fields (GAFF and AMBER) and the MM programs, such as sander et al. With antechamber, one may solve the following problems: (1) identifying bond and atom types; (2) judging atomic equivalence; (3) generating residue topology files; and (4) finding missing force field parameters and supplying reasonable suggestions. The combination of GAFF and antechamber enables one to study most of the organic molecules with AMBER more efficiently. In the following, the main programs in the antechamber package are introduced.

4.1. Principal programs

The *antechamber* program itself is the main program of Antechamber: if your molecule falls in fairly broad categories, this should be all you need to convert an input pdb file into a "prep input" file ready for LEaP.

If there are missing parameters after *antechamber* is finished, you may want to run *parmchk* to generate a *frcmod* template that will assist you in generating the needed parameters.

4.1.1. antechamber

This is the most important program in the package. It can perform many file conversions, and can also assign atomic charges and atom types. As required by the input, *antechamber* executes the following programs: *divcon, atomtype, am1bcc, bondtype, espgen, respgen* and *prepgen*. It may also generate a lot of intermediate files (all in capital letters). If there is a problem with *antechamber*, you may want to run the individual programs that are described below. Antechamber options are given here:

```
-i   input file name
-fi  input file format
-o   output file name
-fo  output file format
-c   charge method
-cf  charge file name
-nc  net molecular charge (int)
-a   additional file name
-fa  additional file format
-ao  additional file operation
     crd : only read in coordinate
     crg: only read in charge
     name  : only read in atom name
     type  : only read in atom type
     bond  : only read in bond type
-m   mulitiplicity (2S+1), default is 1
-rn  residue name, if not available in the input file, default is MOL
-rf  residue toplogy file name in prep input file, default is molecule.res
-mk  divcon keyword, in a pair of quotation marks
-gk  gaussian keyword, in a pair of quotation marks
-at  atom type, can be gaff, amber, bcc and sybyl, default is gaff
-du  check atom name duplications, can be yes(y) or no(n), default is yes
-j   atom type and bond type prediction index, default is 4
     0    : no assignment
     1    : atom type
     2    : full  bond types
     3    : part  bond types
     4    : atom and full bond type
     5    : atom and part bond type
-s   status information, can be 0 (brief), 1 (the default) and 2 (verbose)
-pf  remove the intermediate files: can be yes (y) and no (n), default is no
-i -o -fi and -fo must appear in command lines and the others are optional
```

```
              List of the File Formats

file format type  abbre. index | file format type abbre. index
-------------------------------------------------------------
Antechamber        ac       1  | Sybyl Mol2         mol2    2
PDB                pdb      3  | Modifiled PDB      mpdb    4
amber PREP (int)   prepi    5  |
Gaussian Z-Matrix  gzmat    7  | Gaussian Cartesian gcrt    8
Mopac Internal     mopint   9  | Mopac Cartesian    mopcrt 10
Gaussian Output    gout    11  | Mopac Output       mopout 12
Alchemy            alc     13  | CSD                csd    14
MDL                mdl     15  | Hyper              hin    16
amber Restart      rst     17
-------------------------------------------------------------

amber restart file can only be read in as additional file

              List of the Charge Methods

charge method     abbre.  index | charge method      abbre. index
-----------------------------------------------------------------
RESP               resp     1  |  AM1-BCC            bcc     2
CM2                cm2      3  |  ESP (Kollman)      esp     4
Mulliken           mul      5  |  Gasteiger          gas     6
Read in Charge     rc       7  |  Write out charge   wc      8
-----------------------------------------------------------------
```

Examples:

```
antechamber -i g98.out -fi gout -o sustiva_resp.mol2 -fo mol2 -c resp
antechamber -i g98.out -fi gout -o sustiva_bcc.mol2 -fo mol2 -c bcc -j 5
antechamber -i g98.out -fi gout -o sustiva_gas.mol2 -fo mol2 -c gas
antechamber -i g98.out -fi gout -o sustiva_cm2.mol2 -fo mol2 -c cm2
antechamber -i g98.out -fi gout -o sustiva.ac -fo ac
antechamber -i sustiva.ac -fi ac -o sustiva.mpdb -fo mpdb
antechamber -i sustiva.ac -fi ac -o sustiva.mol2 -fo mol2
antechamber -i sustiva.mol2 -fi mol2 -o sustiva.gzmat -fo gzmat
antechamber -i sustiva.ac -fi ac -o sustiva_gas.ac -fo ac -c gas
```

The *-rn* line specifies the residue name to be used; thus, it must be one to three characters long. The *-at* flag is used to specify whether atom types are to be created for the general AMBER force field (gaff) or for atom types consistent with parm94.dat and parm99.dat (amber). Atom types for gaff are all in lower case, and the AMBER atom types are always in upper case. If you are using *antechamber* to create a modified residue for use with the standard AMBER parm94/parm99 force fields, you should set this flag to `amber`; if you are looking at a more arbitrary molecule, set this to `gaff`, even if you plan to use this as a ligand bound to a macromolecule described by the AMBER force fields.

4.1.2. parmchk

Parmchk reads in an ac file as well as a force field file (*gaff.dat* in $AMBERHOME/dat/leap/parm). It writes out a frcmod file for the missing parameters. For each atom type, an atom type corresponding file (ATCOR.DAT) lists its replaceable general atom type. Be careful to those problematic parameters indicated with "ATTN, need revision".

```
Usage: parmchk -i input file name
               -o frcmod file name
               -f input file format (prepi, ac ,mol2)
               -p ff parmfile
               -c atom type correspondening file, default is ATCOR.DAT
```

Example:

```
parmchk -i sustiva.prep -f prepi -o frcmod
```

This command reads in sustiva.prep and finds the missing force field parameters listed in frcmod.

4.2. A simple example for antechamber

The most common use of the *antechamber* program suite is to prepare input files for LEaP, starting from a three-dimensional structure, as found in a pdb file. The *antechamber* suite automates the process of developing a charge model and assigning atom types, and partially automates the process of developing parameters for the various combinations of atom types found in the molecule.

As with any automated procedure, caution should be taken to examine the output. Furthermore, the procedure, although carefully tested, has not been widely used by lots of people, so users should certainly be on the lookout for unusual or incorrect behavior.

Suppose you have a PDB-format file for your ligand, say thiophenol, which looks like this:

```
ATOM      1  CG  TP      1      -1.959   0.102   0.795
ATOM      2  CD1 TP      1      -1.249   0.602  -0.303
ATOM      3  CD2 TP      1      -2.071   0.865   1.963
ATOM      4  CE1 TP      1      -0.646   1.863  -0.234
ATOM      5  C6  TP      1      -1.472   2.129   2.031
ATOM      6  CZ  TP      1      -0.759   2.627   0.934
ATOM      7  HE2 TP      1      -1.558   2.719   2.931
ATOM      8  S15 TP      1      -2.782   0.365   3.060
ATOM      9  H19 TP      1      -3.541   0.979   3.274
ATOM     10  H29 TP      1      -0.787  -0.043  -0.938
ATOM     11  H30 TP      1       0.373   2.045  -0.784
ATOM     12  H31 TP      1      -0.092   3.578   0.781
ATOM     13  H32 TP      1      -2.379  -0.916   0.901
```

(This file may be found at *$AMBERHOME/test/antechamber/tp/tp.pdb*). The basic command to create a "prepin" file for LEaP is just:

```
antechamber -i tp.pdb -fi pdb -o tp.mol2 -fo mol2 -c bcc
```

This command says that the input format is pdb, output format is Sybyl mol2, and the BCC charge model is to be used. The output file is shown in the box titled *tp.mol2*. The format of this file is a common one understood by many programs.

You can now run *parmchk* to see if all of the needed force field parameters are available:

```
parmchk -i tp.mol2 -f mol2 -o frcmod
```

tp.mol2

```
@<TRIPOS>MOLECULE
TP
   13    13     1     0     0
SMALL
bcc


@<TRIPOS>ATOM
      1 CG          -1.9590    0.1020    0.7950 ca        1 TP       -0.1186
      2 CD1         -1.2490    0.6020   -0.3030 ca        1 TP       -0.1138
      3 CD2         -2.0710    0.8650    1.9630 ca        1 TP        0.0162
      4 CE1         -0.6460    1.8630   -0.2340 ca        1 TP       -0.1370
      5 C6          -1.4720    2.1290    2.0310 ca        1 TP       -0.1452
      6 CZ          -0.7590    2.6270    0.9340 ca        1 TP       -0.1122
      7 HE2         -1.5580    2.7190    2.9310 ha        1 TP        0.1295
      8 S15         -2.7820    0.3650    3.0600 sh        1 TP       -0.2540
      9 H19         -3.5410    0.9790    3.2740 hs        1 TP        0.1908
     10 H29         -0.7870   -0.0430   -0.9380 ha        1 TP        0.1345
     11 H30          0.3730    2.0450   -0.7840 ha        1 TP        0.1336
     12 H31         -0.0920    3.5780    0.7810 ha        1 TP        0.1332
     13 H32         -2.3790   -0.9160    0.9010 ha        1 TP        0.1432
@<TRIPOS>BOND
     1    1    2 ar
     2    1    3 ar
     3    1   13 1
     4    2    4 ar
     5    2   10 1
     6    3    5 ar
     7    3    8 1
     8    4    6 ar
     9    4   11 1
    10    5    6 ar
    11    5    7 1
    12    6   12 1
    13    8    9 1
@<TRIPOS>SUBSTRUCTURE
     1 TP          1 TEMP              0 ****  ****    0 ROOT
```

This yields the *frcmod* file:

```
remark goes here
MASS

BOND

ANGLE
ca-ca-ha   50.000     120.000   same as ca-ca-hc

DIHE

IMPROPER
ca-ca-ca-ha         1.1          180.0         2.0     Using default value
ca-ca-ca-sh         1.1          180.0         2.0     Using default value

NONBON
```

In this case, there was one missing angle parameter from the *gaff.dat* file, and it was determined by analogy to a similar, known, parameter. The missing improper dihedral term was assigned a default value. (As *gaff.dat* continues to be developed, there should be fewer and fewer missing parameters to be estimated by *parmchk*. The above example is actually drawn from Amber 7; in Amber 8, all of the needed parameters are in *gaff.dat*, as can be seen in $AMBERHOME/test/antechamber/tp.) In some cases, *parmchk* may be unable to make a good estimate; it will then insert a placeholder (with zeros everywhere) into the *frcmod* file, with the comment "ATTN: needs revision". After manually editing this to take care of the elements that "need revision", you are ready to read this residue into LEaP, either as a residue on its own, or as part of a larger system. The following LEaP input file (*leap.in*) will just create a system with thiophenol in it:

```
source leaprc.gaff
mods = loadAmberParams frcmod
loadMol2 tp.mol2
saveAmberParm TP prmtop prmcrd
quit
```

You can read this into LEaP as follows:

```
tleap -s -f leap.in
```

This will yield a *prmtop* and *prmcrd* file. If you want to use this residue in the context of a larger system, you can insert commands after the loadAmberPrep step to construct the system you want, using standard LEaP commands.

In this respect, it is worth noting that the atom types in *gaff.dat* are all lower-case, whereas the atom types in the standard AMBER force fields are all upper-case. This means that you can load both *gaff.dat* and (say) *parm99.dat* into LEaP at the same time, and there won't be any conflicts. Hence, it is generally expected that you will use one of the AMBER force fields to describe your protein or nucleic acid, and the *gaff.dat* parameters to describe your ligand; as mentioned above, *gaff.dat* has been designed with this in mind, *i.e.* to produce molecular mechanics

descriptions that are generally compatible with the AMBER macromolecular force fields.

The procedure above only works as it stands for neutral molecules. If your molecule is charged, you need to set the *-nc* flag in the initial *antechamber* run. Also note that this procedure depends heavily upon the initial 3D structure: it must have all hydrogens present, and the charges computed are those for the conformation you provide, after minimization in the AM1 Hamiltonian. In fact, this means that you must have an reasonable all-atom initial model of your molecule (so that it can be minimized with the AM1 Hamiltonian), and you must specify what its net charge is. The system should really be a closed-shell molecule, since all of the atom-typing rules assume this implicitly.

Further examples of using *antechamber* to create force field parameters can be found in the *$AMBERHOME/test/antechamber* directory. Here are some practical tips from Junmei Wang:

(1) For the input molecules, make sure there are no open valences and the sturctures are reasonable.

(2) Failures are most likely produced when *antechamber* infers an incorrect connectivity. In such cases, you can revise by hand the connectivity information in "ac" or "mol2" files. Systematic errors could be corrected by revising the parameters in CONNECT.TPL in $AMBERHOME/dat/antechamber.

(3) It is a good idea to check the intermediate files in case of a program failure, and you can run separate programs one by one. Use the "-s 2" flag to *antechamber* to see details of what it is doing.

(4) Please visit *www.amber.ucsf.edu/antechamber.html* to obtain the latest information about *antechamber* development and to download the latest GAFF parameters. Please report program failures to Junmei Wang at <jwang@encysive.com>.

4.3. Programs called by antechamber

The following programs are automatically called by *antechamber* when needed. Generally, you should not need to run them yourself, unless problems arise and/or you want to fine-tune what *antechamber* does.

4.3.1. atomtype

Atomtype reads in an ac file and assigns the atom types. You may find the default definition files in $AMBERHOME/dat/antechamber: ATOMTYPE_AMBER.DEF (AMBER), ATOMTYPE_GFF.DEF (general AMBER force field). ATOMTYPE_GFF.DEF is the default definition file.

```
Usage: atomtype -i input file name
                -o output file name (ac)
                -f input file format(ac (the default) or mol2)
                -p amber or gaff or bcc or gas, it is supressed by "-d" option
                -d atom type defination file, optional
```

Example:

```
atomtype -i sustiva_resp.ac -o sustiva_resp_at.ac -f ac -p amber
```

This command assigns atom types for *sustiva_resp.ac* with amber atom type definitions. The

output file name is *sustiva_resp_at.ac*

4.3.2. am1bcc

Am1bcc first reads in an ac or mol2 file with or without assigned AM1-BCC atom types and bond types. Then the bcc parameter file (the default, BCCPARM.DAT is in $AMBERHOME/dat/antechamber) is read in. An ac file with AM1-BCC charge is written out. Be sure the charges in the input ac file are AM1-Mulliken charges, which can be generated with *antechamber*.

```
Usage: am1bcc -i input file name in ac format
              -o output file name
              -f output file format(pdb or ac, optional, default is ac)
              -p bcc parm file name (optional))
              -j atom and bond type judge option, default is 0)
                 0: No judgement
                 1: Atom type
                 2: Full bond type
                 3: Partial bond type
                 4: Atom and full bond type
                 5: Atom and partial bond type
```

Example:

```
am1bcc -i comp1.ac -o comp1_bcc.ac -f ac -j 4
```

This command reads in *comp1.ac*, assigns both atom types and bond types and finally performs bond charge correction to get AM1-BCC charges. The '-j' option of 4, which is the default, means that both the atom and bond type information in the input file is ignored and a full atom and bond type assignments are performed. The '-j' option of 3 and 5 implies that bond type information (single bond, double bond, triple bond and aromatic bond) is read in and only a bond type adjustment is performed. If the input file is in mol2 format that contains the basic bond type information, option of 5 is highly recommended. *comp1_bcc.ac* is an ac file with the final AM1-BCC charges.

4.3.3. bondtype

bondtype is a program to assign the atom types and bond types according to the AM1-BCC definitions (BCCTYPE.DEF in $AMBERHOME/dat/antechamber). This program can read an ac file or mol2 file; the output file is an ac file with predicted atom types and bond types. You can choose to determine to assign atom types or bond types or both. If there is some problem with the assignment of bond types, you will get some warnings and for each problematic bond, a "!!!" is appended at the end of the line. In initial tests, the current version works for most organic molecules (>95% overall and >90% for charged molecules).

```
Usage: bondtype -i input file name
                -o output file name
                -f input file format (ac or mol2)
                -j judge bond type level option, default is part
                   full  full judgement
                   part  partial judgement, only do reassignment according
                         to known bond type information in the input file
```

Example:

```
#! /bin/csh -fv
set mols = /bin/ls*.ac
foreach mol ($mols)
   set mol_dir = $mol:r
   antechamber -i $mol_dir.ac -fi ac -fo ac -o $mol_dir.ac -c mul
   bondtype -i $mol_dir.ac -f ac -o $mol_dir.dat -j full
   am1bcc -i $mol_dir.dat -o $mol_dir_bcc.ac -f ac -j 0
end
exit(0)
```

The above script finds all the files with the extension of "ac", calculates the Mulliken charges using *antechamber*, and predicts the atom and bond types with bondtype. Finally, AM1-BCC charges are generated by running am1bcc to do the bond charge correction.

4.3.4. prepgen

Prepgen generates the prep input file from an ac file. By default, the program generates a mainchain itself. However, you may also specify the mainchain atom in the mainchain file. From this file, you can also specify which atoms will be deleted, and whether to do charge correction or not. In order to generate the amino-acid-like residue (this kind of residue has one head atom and one tail atom to be connected to other residues), you need a mainchain file. Sample mainchain files are in $AMBERHOME/dat/antechamber.

```
Usage: prepgen -i  input file name(ac)
               -o  output file name
               -f  output file format (car or int, default: int)
               -m  mainchain file name
               -rn residue name (default: MOL)
               -rf residue file name (default: molecule.res)
                   -f -m -rn -rf are optional
```

Examples:

```
prepgen -i sustiva_resp_at.ac -o sustiva_int.prep -f int -rn SUS -rf SUS.res
prepgen -i sustiva_resp_at.ac -o sustiva_car.prep -f car -rn SUS -rf SUS.res
prepgen -i sustiva_resp_at.ac -o sustiva_int_main.prep -f int -rn SUS
        -rf SUS.res -m mainchain_sus.dat
prepgen -i ala_cm2_at.ac -o ala_cm2_int_main.prep -f int -rn ALA -rf ala.res
        -m mainchain_ala.dat
```

The above commands generate different kinds of prep input files with and without specifying a mainchain file.

4.3.5. espgen

Espgen reads in a gaussian (92,94,98,03) output file and extracts the ESP information. An esp file for the resp program is generated.

```
Usage: espgen -i   input file name
              -o   output file name
```

Example:

```
espgen -i sustiva_g98.out -o sustiva.esp
```

The above command reads in sustiva_g98.out and writes out sustiva.esp, which can be used by the resp program. Note that this program replaces shell scripts formerly found on the AMBER web site that perform equivalent tasks.

4.3.6. respgen

Respgen generates the input files for two-stage resp fitting. The current version only supports single molecule fitting. Atom equivalence is recognized automatically.

```
Usage: respgen -i input file name(ac)
               -o output file name
               -f output file format (resp1 or resp2)
                  resp1 - first stage resp fitting
                  resp2 - second stage resp fitting
```

Example:

```
respgen -i sustiva.ac -o sustiva.respin1 -f resp1
respgen -i sustiva.ac -o sustiva.respin2 -f resp2
resp -O -i sustiva.respin1 -o sustiva.respout1 -e sustiva.esp -t qout_stage1
resp -O -i sustiva.respin2 -o sustiva.respout2 -e sustiva.esp -q qout_stage1
        -t qout_stage2
antechamber -i sustiva.ac -fi ac -o sustiva_resp.ac -fo ac -c rc
            -cf qout_stage2
```

The above commands first generate the input files (sustiva.respin1 and sustiva.respin2) for resp fitting, then do two-stage resp fitting and finally use *antechamber* to read in the resp charges and write out an ac file − sustiva_resp.ac.

4.4. Miscellaneous programs

The Antechamber suite also contains some utility programs that perform various tasks in molecular mechanical calculations. They are listed in alphabetical order.

4.4.1. crdgrow

Crdgrow reads an incomplete pdb file (at least three atoms in this file) and a prep input file, and then generates a complete pdb file. It can be used to do residue mutation. For example, if you want to change one protein residue to another one, you can just keep the mainchain atoms in a pdb file and read in the prep input file of the residue to be changed, and crdgrow will generate the coordinates of the missing atoms.

```
Usage: crdgrow -i input file name
               -o output file name
```

```
                    -p prepin file name
                    -f prepin file format: prepi (the default)
```

Example:

```
crdgrow -i ref.pdb -o new.pdb -p sustiva_int.prep
```

This command reads in ref.pdb (only four atoms) and prep input file sustiva_int.prep, then generates the coordinates of the missing atoms and writes out a pdb file (new.pdb).

4.4.2. parmcal

Parmcal is an interactive program to calculate the bond length and bond angle parameters, according to the rules outlined in [4].

```
Please select:
1. calculate the bond length parameter: A-B
2. calculate the bond angle parameter: A-B-C
3. exit
```

4.4.3. database

Database reads in a multiple sdf or mol2 file and a description file to run a set of commands for each record sequentially. The commands are defined in the description file.

```
Usage: database -i database file name
                -d definition file name
```

Example:

```
database -i sample_database.mol2 -d mol2.def
```

This command reads in a multiple mol2 database - sample_database.mol2 and a description file mol2.def to run a set of commands (defined in mol2.def) to generate prep input files and merge them to a single file called total.prepi. Both files are located in the following directory: $AMBERHOME/test/antechamber/database/mol2.

5. Sander

5.1. Introduction.

This is a guide to *sander*, the AMBER module which carries out energy minimization, molecular dynamics, and NMR refinements. The acronym stands for **S**imulated **A**nnealing with **N**MR-**D**erived **E**nergy **R**estraints, but this module is used for a variety of simulations that have nothing to do with NMR refinement. Some general features are outlined in the following paragraphs:

(1) *Sander* provides direct support for several force fields for proteins and nucleic acids, and for several water models and other organic solvents. The basic force field implemented here has the following form, which is about the simplest functional form that preserves the essential nature of molecules in condensed phases:

$$U(\mathbf{R}) = \sum_{bonds} K_r\,(r - r_{eq})^2 \qquad \textit{bond}$$

$$+ \sum_{angles} K_\theta\,(\theta - \theta_{eq})^2 \qquad \textit{angle}$$

$$+ \sum_{dihedrals} \frac{V_n}{2}(1 + \cos[n\phi - \gamma]) \qquad \textit{dihedral}$$

$$+ \sum_{i<j}^{atoms} \frac{A_{ij}}{R_{ij}^{12}} - \frac{B_{ij}}{R_{ij}^{6}} \qquad \textit{van der Waals}$$

$$+ \sum_{i<j}^{atoms} \frac{q_i q_j}{\varepsilon R_{ij}} \qquad \textit{electrostatic}$$

"Non-additive" force fields based on atom-centered dipole polarizabilities can also be used. These add a "polarization" term to what was given above:

$$E_{pol} = -\frac{1}{2}\sum_{i}^{atom} \mu_i \bullet \mathbf{E}_i^{(o)} \qquad \textit{polarization}$$

where μ_i is an induced atomic dipole. In addition, charges that are not centered on atoms, but are off-center (as for lone-pairs or "extra points") can be included in the force field.

(2) The particle-mesh Ewald (PME) procedure (or, optionally, a "true" Ewald sum) is used to handle long-range electrostatic interactions. Long-range van der Waals interactions are estimated by a continuum model. Biomolecular simulations in the NVE ensemble (*i.e.* with Newtonian dynamics) conserve energy well over multi-nanosecond runs without modification of the equations of motion.

(3) Two periodic imaging geometries are included: rectangular parallelepiped and truncated octahedron (box with corners chopped off). (*Sander* itself can handle many other periodically-replicating boxes, but input and output support in *LEaP* and *ptraj* is only available right now for these two.) The size of the repeating unit can be coupled to a given external pressure, and velocities can be coupled to a given external temperature by several schemes. The external conditions and coupling constants can be varied over time, so

various simulated annealing protocols can be specified in a simple and flexible manner.

(4) It is also possible to carry out non-periodic simulations in which aqueous solvation effects are represented *implicitly* by a generalized Born/ surface area model by adding the following two terms to the "vacuum" potential function:

$$+ \sum_{ij}^{atoms} \frac{q_i q_j}{f^{gb}(R_{ij})} \qquad + \sigma \mathbf{A} \qquad \textit{implicit solvation}$$

The first term accounts for the polar part of solvation (free) energy via the f^{gb} function [46] designed to provide an approximation for the reaction field potential, and the second represents the non-polar contribution which is taken to be proportional to the surface area of the molecule *A*.

(5) Users can define internal restraints on bonds, valence angles, and torsions, and the force constants and target values for the restraints can vary during the simulation. The penalty function can consist of as many as three types of region: it can be flat between an "inner" set of upper and lower bounds (called r_2 and r_3); then rise parabolically when the internal coordinate violates these bounds; and finally, since large violations may lead to excessive parabolic penalties, these parabolas can smoothly turn into linear penalties outside even wider upper and lower bounds (called r_1 and r_4). The imposition of restraints can be made dependent upon the distance that residues are apart in the amino-acid sequence, so that much of the functionality of programs like DISMAN or DIANA is available. The relative weights of various terms in the force field can be varied over time, allowing one to implement a variety of simulated annealing protocols in a single run.

(6) Internal restraints can be defined to be "time-averaged", that is, restraint forces are applied based on the averaged value of an internal coordinate over the course of the dynamics trajectory, not only on its current value. Alternatively, restraints can be "ensemble-averaged" using the locally-enhanced-sampling (LES) option.

(7) Restraints can be directly defined in terms of NOESY intensities (calculated with a relaxation matrix technique), residual dipolar couplings, scalar coupling constants and proton chemical shifts. There are provisions for handling overlapping peaks or ambiguous assignments. In conjunction with distance and angle constraints, this provides a powerful and flexible approach to NMR structural refinements.

(8) Restraints can also be defined in terms of the root-mean-square coordinate distance from some reference structure. This allows one to bias trajectories either towards or away from some target.

(9) Free energy calculations, using thermodynamic integration (TI) with a linear or non-linear mixing of the "unperturbed" and "perturbed" Hamiltonian, can be carried out. Alternatively, potentials of mean force can be computed using umbrella sampling.

We have divided this manual into the six sections listed in the accompanying table. If you are just doing "standard" minimization, dynamics, or free energy simulations, read section *one*, and ignore the rest. If you want to carry out simulated annealing, consult section *two*. Those who wish to carry out simulations while imposing internal coordinate restraints should also read sections *three* and *four*. Sections *five* through *seven* allow you to add sophisticated penalty functions during NMR refinement. Sections 5.13 to 5.21 outline some addition sorts of simulations that can be carried out.

Purpose	*Sections involved*
Simple min/md/free energy	1
varying parameters over time (simulated annealing)	1,2
using internal restraints (including NMR distance & angle constraints)	3,4
nmr refinement using NOESY volume restraints	5
nmr refinement using chemical shift restraints	6
nmr refinement using direct dipolar splittings	7

5.2. Credits.

The annealing, "weight change," "restraints" and NMR-specific portions of *sander* were primarily written by David Pearlman and David Case. All of the AMBER crew listed on the title page contributed to the general portions; the polarization implementation is that of Jim Caldwell, Liem Dang, and Tom Darden, and the "targeted MD" code is from Carlos Simmerling. The pseudocontact shift code was provided by Ivano Bertini of the University of Florence. A brief overview and history of parallel implementations is given in the Installation section, as well as in Ref [1].

Particle Mesh Ewald. The Particle Mesh Ewald (PME) method was implemented originally in AMBER 3a by Tom Darden, and has been developed in subsequent versions of AMBER by several people, in particular by Tom Darden, Tom Cheatham, Mike Crowley and David Case. The PME method not only provides a better treatment of long range electrostatics (at a modest computational cost), but can be applied in both rectangular and non-rectangular periodic boundary simulations [47-51].

Generalized Born. When *igb*=1, we use the "pairwise" generalized Born model introduced by Hawkins, Cramer and Truhlar [52,53], which is based on earlier ideas by Still and others [46,54-56]. Radii are the Bondi radii [57], optionally with slight modifications of the different types of hydrogen atoms [42] ; the overlap parameters are taken from the TINKER molecular modeling package (`http://tinker.wustl.edu`). The effects of added monovalent salt are included at a level that approximates the solutions of the linearized Poisson-Boltzmann equation [58]. The implementation is by David Case, who thanks Charlie Brooks for inspiration.

When *igb*=2 or *igb*=5, modifications outlined by Onufriev, Bashford and Case are used [8].

Solvent-accessible surface areas It is also possible to carry out GB/SA simulations, using the surface areas (SA) to approximate the cavity and van der Waals contributions to solvation. The surface area is calculated using the LCPO (Linear Combinations of Pairwise Overlaps) model [59].

5.3. File usage.

Usage: `sander [-help] [-O] -i mdin -o mdout -p prmtop -c inpcrd -r restrt -ref refc -x mdcrd -v mdvel -e mden -inf mdinfo -radii radii -cpin cpin -cpout cpout -cprestrt cprestrt -mmtsb mmtsb_setup.job`

`-O` Overwrite output files if they exist.

file	*in/out*	*purpose*
mdin	input	control data for the min/md run
mdout	output	user readable state info and diagnostics -o stdout will send output to stdout (to the terminal) instead of to a file.
mdinfo	output	latest mdout-format energy info
prmtop	input	molecular topology, force field, periodic box type, atom and residue names
inpcrd	input	initial coordinates and (optionally) velocities and periodic box size
refc	input	(optional) reference coords for position restraints; also used for targeted MD
mdcrd	output	coordinate sets saved over trajectory
mdvel	output	velocity sets saved over trajectory
mden	output	extensive energy data over trajectory
restrt	output	final coordinates, velocity, and box dimensions if any - for restarting run
inpdip	input	polarizable dipole file, when indmeth=3
rstdip	output	polarizable dipole file, when indmeth=3
cpin	input	protonation state definitions
cprestrt		protonation state definitions, final protonation states for restart (same format as cpin)
cpout	output	protonation state data saved over trajectory
mmtsb_setup.job	internal	contents generated by MMTSB server

5.4. Example input files.

Here are a couple of sample files, just to establish a basic syntax and appearance. There are more examples of NMR-related files later in this chapter.

1. Simple restrained minimization

```
Minimization with Cartesian restraints
 &cntrl
   imin=1, maxcyc=200,           (invoke minimization)
   ntpr=5,                       (print frequency)
   ntr=1,                        (turn on Cartesian restraints)
   restraint_wt=1.0,             (force constant for restraint)
   restraintmask=':1-58',        (atoms in residues 1-58 restrained)
 /
```

2. "Plain" molecular dynamics run

```
 molecular dynamics run
 &cntrl
    imin=0, irest=1, ntx=5,                 (restart MD)
    ntt=1, temp0=300.0, tautp=0.2,          (temperature control)
    ntp=1, taup=2.0,                        (pressure control)
    ntb=2, ntc=2, ntf=2,                    (SHAKE, periodic bc.)
    nstlim=500000,                          (run for 0.5 nsec)
    ntwe=100, ntwx=100, ntpr=200,           (output frequency)
 /
```

5.5. Overview of the information in the input file.

Section	*Comments*	*Format*
ONE	Standard minimization and dynamics input	One or more title lines, followed by the (required) `&cntrl` and (optional) `&pb`, `&ewald` or `&debugf` namelist blocks
TWO	Varying conditions	Parameters for changing temperature, restraint weights, etc. during the MD run. Each parameter is specified by a separate `&wt` namelist block, ending with `&wt type='END', /`.
THREE	I/O redirection	TYPE=*filename* lines. Section ends with the first non-blank line which does not correspond to a recognized redirection.
FOUR	Distance and angle restraints	Read only if NMROPT>0 and a *DISANG=filename* line is present in section THREE. Multiple `&rst` namelists are read from DISANG. One `&rst` definition is given per restraint.
FIVE	NOESY volume restraints	Read only if NMROPT= 2 and a *NOEEXP=filename* line is present in section THREE. Defines molecular subgroups. Each definition consists of one `&noeexp` namelist followed by the group cards defining the subgroup.
SIX	Chemical shifts restraints	Read only if NMROPT= 2 and a *SHIFTS=filename* or *PCSHIFT=filename* line is present in section THREE, Exactly one `&shf` or `&pcshf` namelist (or both) must be provided for this section.
SEVEN	Direct dipolar coupling restraints	Read only if NMROPT= 2 and a *DIPOLE=filename* command was given in section THREE, Exactly one `&align` namelist block must be provided for this section.
EIGHT	Group information	Read if IDECOMP=1. Input format is described in Appendix B.

5.6. SECTION ONE: General minimization and dynamics parameters.

Each of the variables listed below is input in a namelist statement with the namelist identifier `&cntrl`. You can enter the parameters in any order, using keyword identifiers. Variables that are not given in the namelist input retain their default values. Support for namelist input is included in almost all current Fortran compilers, and is a standard feature of Fortran 90. A detailed description of the namelist convention is given in Appendix A.

In general, namelist input consists of an arbitrary number of comment cards, followed by a record whose first 7 characters after a " &" (e.g. " `&cntrl` ") name a group of variables that can be set by name. This is followed by statements of the form " `maxcyc=500, diel=2.0, ...` ", and is concluded by an " / " token. The first line of input contains a title, which is then followed by the `&cntrl` namelist. Note that the first character on each line of a namelist block must be a blank.

Some of the options and variables are much more important, and commonly modified, than are others. We have denoted the "common" options by printing them in **boldface** below. In general, you can skip reading about the non-bold options on a first pass, and you should change these from their defaults only if you think you know what you are doing.

5.6.1. General flags describing the calculation.

IMIN Flag to run minimization

- = 0 No minimization (only do molecular dynamics; default)
- = 1 Perform minimization (and no molecular dynamics)
- =5 Read in a trajectory for analysis.

NMROPT

- = 0 no nmr-type analysis will be done; default (Note: this variable replaces `nmrmax` from previous versions, and has a slightly different meaning.)
- > 0 NMR restraints/weight changes will be read
- = 2 NOESY volume restraints or chemical shift restraints will be read as well

---

5.6.2. Nature and format of the input.

NTX Option to read the initial coordinates, velocities and box size from the "inpcrd" file. The options 1-2 must be used when one is starting from minimized or model-built coordinates. If an MD restrt file is used as inpcrd, then options 4-7 may be used.

= 1 X is read formatted with no initial velocity information (default)

= 2 X is read unformatted with no initial velocity information

= 4 X and V are read unformatted.

= 5 X and V are read formatted; box information will be read if ntb>0. The velocity information will only be used if *irest*=1.

= 6 X, V and BOX(1..3) are read unformatted; in other respects, this is the same as option "5".

=7 Same as option "5"; only included for backward compatibility with earlier versions of Amber.

IREST Flag to restart the run.

= 0 No effect (default)

= 1 restart calculation. Requires velocities in coordinate input file, so you also may need to reset NTX if restarting MD

NTRX Format of the Cartesian coordinates for restraint from file "refc". Note: the program expects file "refc" to contain coordinates for all the atoms in the system. A subset for the actual restraints is selected by *restraintmask* in the control namelist.

= 0 Unformatted (binary) form

= 1 Formatted (ascii, default) form

5.6.3. Nature and format of the output.

NTXO Format of the final coordinates, velocities, and box size (if constant volume or pressure run) written to file "restrt".

= 0 Unformatted

= 1 Formatted (default).

NTPR Every NTPR steps energy information will be printed in human-readable form to files "mdout" and "mdinfo". "mdinfo" is closed and reopened each time, so it always contains the most recent energy and temperature. Default 50.

NTAVE Every NTAVE steps of dynamics, running averages of average energies and fluctuations over the last NTAVE steps will be printed out. Default value of 0 disables this printout.

NTWR Every NTWR steps during dynamics, the "restrt" file will be written, ensuring that recovery from a crash will not be so painful. In any case, restrt is written every NSTLIM steps for both dynamics and minimization calculations. If NTWR<0, a unique copy of the file, restrt_nstep, is written every abs(NTWR) steps. This option is useful if for example one wants to run free energy perturbations from multiple starting points or save a series of restrt files for minimization. Default 500.

IWRAP If set to 1, the coordinates written to the restart and trajectory files will be "wrapped" into a primary box. This means that for each molecule, the image closest to the middle of the "primary box" [with x coordinates between 0 and

a, y coordinates between 0 and b, and z coordinates between 0 and c] will be the one written to the output file. This often makes the resulting structures look better visually, but has no effect on the energy or forces. Performing such wrapping, however, can mess up diffusion and other calculations. The default (when *iwrap=0*) is to not perform any such manipulations; in this case it is typical to use *ptraj* as a post-processing program to translate molecules back to the primary box. You may also want to use *iwrap=1* if you are preparing a system for further runs in *gibbs*, since that program requires the coordinates to be wrapped. For very long runs, setting *iwrap*=1 may be required to keep the coordinate output from overflowing the trajectory file format.

NTWX Every NTWX steps the coordinates will be written to file "mdcrd". NTWX=0 inhibits all output. Default 0.

NTWV Every NTWV steps the velocities will be written to file "mdvel". NTWV=0 inhibits all output. Default 0.

NTWE Every NTWE steps the energies and temperatures will be written to file "mden" in compact form. NTWE=0 inhibits all output. Default 0.

IOUTFM Format of velocity, coordinate, and energy sets. Note: these values are "backwards" compared to NTRX and NTXO; this is an ancient mistake that we are reluctant to change, since it would break existing scripts.

= 0 Formatted (default)

= 1 Binary

NTWPRT Coordinate/velocity archive limit flag. This flag can be used to decrease the size of the coordinate / velocity archive files, by only including that portion of the system of greatest interest. (E.g. one can print only the solute and not the solvent, if so desired). The Coord/velocity archives will include:

= 0 all atoms of the system (default).

> 0 only atoms 1->NTWPRT.

IDECOMP This option is only really useful in conjunction with *mm_pbsa*, where it is turned on automatically if required. The options are:

= 0 Do nothing (default).

= 1 Decompose energies on a per-residue basis; 1-4 EEL + 1-4 VDW are added to internal (bond, angle, dihedral) energies.

= 2 Decompose energies on a per-residue basis; 1-4 EEL + 1-4 VDW are added to EEL and VDW.

= 3 Decompose energies on a pairwise per-residue basis; the rest is equal to "1".

= 4 Decompose energies on a pairwise per-residue basis; the rest is equal to "2".

If decomp is switched on, residues may be chosen by the RRES and/or LRES card. The RES card determines about which residues information is finally output. See the *mm_pbsa* chapter for more information. Use of *idecomp* > 0 is incompatible with *ntr* > 0 or *ibelly* > 0.

5.6.4. Potential function.

NTF Force evaluation. Note: If SHAKE is used (see NTC), it is not necessary to calculate forces for the constrained bonds.

= 1 complete interaction is calculated (default)

= 2 bond interactions involving H-atoms omitted (use with NTC=2)

= 3 all the bond interactions are omitted (use with NTC=3)

= 4 angle involving H-atoms and all bonds are omitted

= 5 all bond and angle interactions are omitted

= 6 dihedrals involving H-atoms and all bonds and all angle interactions are omitted

= 7 all bond, angle and dihedral interactions are omitted

= 8 all bond, angle, dihedral and non-bonded interactions are omitted

NTB Periodic boundary. If NTB .EQ. 0 then a boundary is NOT applied regardless of any boundary condition information in the topology file. The value of NTB specifies whether constant volume or constant pressure dynamics will be used. Options for constant pressure are described in a separate section below.

= 0 no periodicity is applied and PME is off

= 1 constant volume (default)

= 2 constant pressure

If NTB .NE. 0, there must be a periodic boundary in the topology file. Constant pressure is not used in minimization (IMIN=1, above).

For a periodic system, constant pressure is the only way to equilibrate density if the starting state is not correct. For example, the solvent packing scheme used in LEaP can result in a net void when solvent molecules are subtracted which can aggregate into "vacuum bubbles" in a constant volume run. Another potential problem are small gaps at the edges of the box. The upshot is that almost every system needs to be equilibrated at constant pressure (*ntb=2, ntp>0*) to get to a proper density. But be sure to equilibrate first (at constant volume) to something close to the final temperature, before turning on constant pressure.

DIELC Dielectric multiplicative constant for the electrostatic interactions. Default is 1.0. Please note this is NOT related to dielectric constants for generalized Born simulations.

CUT This is used to specify the nonbonded cutoff, in Angstroms. For PME, the cutoff is used to limit direct space sum, and the default value of 8.0 is usually a good value. When *igb>0*, the cutoff is used to truncate nonbonded pairs (on an atom-by-atom basis); here a larger value than the default is generally required. A separate parameter (**RGBMAX**) controls the maximum distance between atom pairs that will be considered in carrying out the pairwise summation involved in calculating the effective Born radii, see the generalized Born section below.

SCNB 1-4 vdw interactions are divided by SCNB. Default 2.0.

SCEE 1-4 electrostatic interactions are divided by SCEE; the 1991 and previous force fields used 2.0, while the 1994 force field uses 1.2. Default is 1.2.

NSNB Determines the frequency of nonbonded list updates when *igb=0* and *nbflag=0*; see the description of *nbflag* for more information. Default is 25.

IPOL Inclusion of polarizabilities in the force field (see below).

= 0 non polar calc (default).

= 1 turn on polarization calculation. Polarizabilities must be present in prmtop.

5.6.5. Generalized Born/Surface Area options.

The generalized Born solvation model can be used instead of explicit water for non-polarizable force fields such as ff94 or ff99. There are several "flavors" of GB available, depending upon the value of *igb*. The version that has been most extensively tested corresponds to *igb*=1; the "OBC" models (igb=2 and 5) are newer, but appear to give significant improvements and are recommended for most projects (certainly for peptides or proteins). Users should understand that all (current) GB models have limitations and should proceed with caution. Generalized Born simulations can only be run for non-periodic systems, *i.e.* where *ntb*=0. The nonbonded cutoff for GB calculations should be greater than that for PME calculations, perhaps *cut*=16. The slowly-varying forces generally do not have to be evaluated at every step for GB, either *nrespa*=2 or 4.

IGB

= 0 No generalized Born term is used. (Default)

= 1 The Hawkins, Cramer, Truhlar [52,53] pairwise generalized Born model (GB^{HCT}) is used, with parameters described by Tsui and Case [42]. This model uses the default radii set up by LEaP. It is slightly different from the GB model that was included in Amber6. If you want to compare to Amber 6, or need to continue an ongoing simulation, you should use the command "set default PBradii amber6" in LEaP, and set *igb*=1 in *sander*. For reference, the Amber6 values are those used by an earlier Tsui and Case paper [43].

= 2 Use a modified GB model developed by A. Onufriev, D. Bashford and D.A. Case (GB^{OBC}); the main idea was published earlier [60], but the actual implementation here is an elaboration of this initial idea [8]. With *igb*=2, the inverse of the effective Born radius is given by:

$$R_i^{-1} = \bar{\rho}_i^{-1} - \tanh(\alpha\Psi - \beta\Psi^2 + \gamma\Psi^3)/\rho_i$$

where $\bar{\rho}_i = \rho_i - offset$, and $\Psi = I\,\bar{\rho}_i$, with I given in our earlier paper. The parameters α, β, and γ were determined by empirical fits, and have the values 0.8, 0.0, and 2.909125. This corresponds to model I in Ref [8]. With this option, you should use the LEaP command "set default PBradii mbondi2" or "set default PBradii bondi" to prepare the *prmtop* file.

= 3 or 4 These values are unused; they were used in Amber 7 for parameter sets that are no longer supported.

=5 Same as *igb=2*, except that now α, β, γ are 1.0, 0.8, and 4.85. This corresponds to model II in Ref [8]. With this option, you should use the command "set default PBradii mbondi2" in setting up the *prmtop* file, although "set default PBradii bondi" is also OK. When tested in MD simulations of several proteins [8], both of the above parameterizations of the "OBC" model showed equal performance, although further tests [61] on an extensive set of protein structures revealed that the *igb=5* variant agrees better with the Poisson-Boltzmann treatment in calculating the electrostatic part of the solvation free energy.

=10 Calculate the reaction field for a non-periodic solute in a spherical "cap" of water, using a numerical Poisson-Boltzmann solver. This option is described in Section 5.15, below. Note that this is *not* a generalized Born simulation, in spite of its use of *igb*; it is rather an alternative continuum solvent model.

INTDIEL Sets the interior dielectric constant of the molecule of interest. Default is 1.0. Other values have not been extensively tested.

EXTDIEL Sets the exterior or solvent dielectric constant. Default is 78.5.

SALTCON Sets the concentration (M) of 1-1 mobile counterions in solution, using a modified generalized Born theory based on the Debye-Hückel limiting law for ion screening of interactions [58]. Default is 0.0 M (*i.e.* no Debye-Hückel screening.) Setting *saltcon* to a non-zero value does result in some increase in computation time.

RGBMAX This parameter controls the maximum distance between atom pairs that will be considered in carrying out the pairwise summation involved in calculating the effective Born radii. Atoms whose associated spheres are farther way than *rgbmax* from given atom will not contribute to that atom's effective Born radius. This is implemented in a "smooth" fashion (thanks mainly to W.A. Svrcek-Seiler), so that when part of an atom's atomic sphere lies inside *rgbmax* cutoff, that part contributes to the low-dielectric region that determines the effective Born radius. The default is 25 Å, which is usually plenty for single-domain proteins of a few hundred residues. Even smaller values (of 10-15 Å) are reasonable, changing the functional form of the generalized Born theory a little bit, in exchange for a considerable speed-up in efficiency, and without introducing the usual cut-off artifacts such as drifts in the total energy.
The *rgbmax* parameter affects only the effective Born radii (and the derivatives of these values with respect to atomic coordinates). The *cut* parameter, on the other hand, determines the maximum distance for the electrostatic, van der Waals and "off-diagonal" terms of the generalized Born interaction. The value of *rgbmax* might be either greater or smaller than that of *cut*: these two parameters are independent of each other. However, values of *cut* that are too small are more likely to lead to artifacts than are small values of *rgbmax*; therefore one typically sets *rgbmax* <= *cut*.

RBORNSTAT If *rbornstat = 1*, the statistics of the effective Born radii for each atom of the molecule throughout the molecular dynamics simulation are reported in the output file. Default is 0.

OFFSET The dielectric radii for generalized Born calculations are decreased by a uniform value "offset" to give the "intrinsic radii" used to obtain effective Born radii. Default 0.09 Å.

GBSA Option to carry out GB/SA (generalized Born/surface area) simulations. For the default value of 0, surface area will not be computed and included in the solvation term. If *gbsa = 1*, surface area will be computed using the LCPO model. [59] If *gbsa = 2*, surface area will be computed by recursively approximating a sphere around an atom, starting from an icosahedra. Note that no forces are generated in this case, hence, *gbsa = 2* only works for a single point energy calculation and is mainly intended for energy decomposition in the realm of MM_GBSA.

SURFTEN Surface tension used to calculate the nonpolar contribution to the free energy of solvation (when *gbsa = 1*), as Enp = surften*SA. The default is 0.005 kcal/mol-Å2 [62].

RDT This parameter is only used for GB simulations with LES (Locally Enhanced Sampling). In GB+LES simulations, non-LES atoms require multiple effective Born radii due to alternate descreening effects of different LES copies. When the multiple radii for a non-LES atom differ by less than RDT, only a single radius will be used for that atom. See the LES portion of the manual for more details. Default 0.01 Å.

5.6.6. Frozen or restrained atoms.

IBELLY Flag for belly type dynamics.

= 0 No belly run (default).

= 1 Belly run. A subset of the atoms in the system will be allowed to move, and the coordinates of the rest will be frozen. The *moving* atoms are specified *bellymask*. This option is not available when *igb>0*. Note also that this option does *not* provide any significant speed advantage, and is maintained primarily for backwards compatibility with older version of Amber. Most applications should use the *ntr* variable instead to restrain parts of the system to stay close to some initial configuration.

NTR Flag for restraining specified atoms in Cartesian space using a harmonic potential. The restrained atoms are determined by the *restraintmask* string. The force constant is given by *restraint_wt*. The coordinates are read in "restrt" format from the "refc" file (see NTRX, above).

= 0 No position restraints (default)

= 1 MD with restraint of specified atoms

RESTRAINT_WT

The weight (in $kcal/mol - Å^2$) for the positional restraints. The restraint is of the form $k(\Delta x)^2$, where k is the value given by this variable, and Δx is the difference between one of the Cartesian coordinates of a restrained atom and its reference position. There is a term like this for each Cartesian coordinate of

each restrainted atom.

RESTRAINTMASK
String that specifies the *restrained* atoms when *ntr=1*.

BELLYMASK String that specifies the *moving* atoms when *ibelly=1*. The syntax for both *restraintmask* and *bellymask* is given in Chapter 11.5. Note that these mask strings are limited to a maximum of 80 characters.

5.6.7. Energy minimization.

MAXCYC The maximum number of cycles of minimization. Default 1.

NCYC If NTMIN is 1 then the method of minimization will be switched from steepest descent to conjugate gradient after NCYC cycles. Default 10.

NTMIN Flag for the method of minimization.

- = 0 Full conjugate gradient minimization. The first 4 cycles are steepest descent at the start of the run and after every nonbonded pairlist update. Note that the Amber7 documentation incorrectly indicated 10 cycles instead of 4. The first Amber version in which the documentation became incorrect is unknown.
- = 1 For NCYC cycles the steepest descent method is used then conjugate gradient is switched on (default).
- = 2 Only the steepest descent method is used.
- = 3 The XMIN method is used, see *amber8/doc/lmod.pdf*.
- = 4 The LMOD method is used, see *amber8/doc/lmod.pdf*.

DX0 The initial step length. If the initial step length is big then the minimizer will try to leap the energy surface and sometimes the first few cycles will give a huge energy, however the minimizer is smart enough to adjust itself. Default 0.01.

DRMS The convergence criterion for the energy gradient: minimization will halt when the root-mean-square of the Cartesian elements of the gradient is less than DRMS. Default 1.0E-4 kcal/mole Å.

5.6.8. Molecular dynamics.

NSTLIM Number of MD-steps to be performed. Default 1.

NSCM Flag for the removal of translational and rotational center-of-mass (COM) motion at regular intervals. For non-periodic simulations, after every NSCM steps, translational and rotational motion will be removed. For periodic systems, just the translational center-of-mass motion will be removed. This flag is ignored for belly simulations. Default 1000.

T The time at the start (psec) this is for your own reference and is not critical. Start time is taken from the coordinate input file if IREST=1. Default 0.0.

DT The time step (psec). Recommended MAXIMUM is .002 if SHAKE is used, or .001 if it isn't. Note that for temperatures above 300K, the step size should be reduced since greater temperatures mean increased velocities and longer distance traveled between each force evaluation, which can lead to anomalously high energies and system blowup. Default 0.001.

NRESPA This variable allows the user to evaluate slowly-varying terms in the force field less frequently. For PME, "slowly-varying" (now) means the reciprocal sum. For generalized Born runs, the "slowly-varying" forces are those involving derivatives with respect to the effective radii, and pair interactions whose distances are greater than the "inner" cutoff, currently hard-wired at 8 Å.
If NRESPA>1 these slowly-varying forces are evaluated every *nrespa* steps. The forces are adjusted appropriately, leading to an impulse at that step. If *nrespa*dt* is less than or equal to 4 fs the energy conservation is not seriously compromised. However if *nrespa*dt* > 4 fs the simulation becomes less stable. Note that energies and related quantities are only accessible every *nrespa* steps, since the values at other times are meaningless.

5.6.9. Temperature regulation.

NTT Switch for temperature scaling. Note that setting *ntt*=0 corresponds to the microcanonical (NVE) ensemble (which should approach the canonical one for large numbers of degrees of freedom). Some aspects of the "weak-coupling ensemble" (*ntt*=1) have been examined, and roughly interpolate between the microcanonical and canonical ensembles [63]. The *ntt*=2 and 3 options correspond to the canonical (constant T) ensemble. The *ntt*=4 option is included for historical reasons, but does not correspond to any of the traditional ensembles.

= 0 Constant total energy classical dynamics (assuming that *ntb*<2, as should probably always be the case when *ntt=0*).

= 1 Constant temperature, using the weak-coupling algorithm [64]. A single scaling factor is used for all atoms. Note that this algorithm just ensures that the total kinetic energy is appropriate for the desired temperature; it does nothing to ensure that the temperature is even over all parts of the molecule. Atomic collisions should serve to ensure an even temperature distribution, but this is not guaranteed, and can be a particular problem for generalized Born simulations, where there are no collisions with solvent. Other temperature coupling options (especially *ntt*=3) should probably be used for generalized Born simulations.

= 2 Andersen temperature coupling scheme [65], in which imaginary "collisions" randomize the velocities to a distribution corresponding to *temp0* every *vrand* steps. Note that in between these "massive collisions", the dynamics is Newtonian. Hence, time correlation functions (etc.) can be computed in these sections, and the results averaged over an initial canonical distribution. Note also that too high a collision rate (too small a value of *vrand*) will slow down the speed at which the molecules explore configuration space, whereas too low

a rate means that the canonical distribution of energies will be sampled slowly. A discussion of this rate is given by Andersen [66].

= 3 Use Langevin dynamics with the collision frequency γ given by *gamma_ln*, discussed below. Note that when γ has its default value of zero, this is the same as setting *ntt* = 0.

TEMP0 Reference temperature at which the system is to be kept, if *ntt* > 0. Note that for temperatures above 300K, the step size should be reduced since increased distance traveled between evaluations can lead to SHAKE and other problems. Default 300.

TEMP0LES This is the target temperature for all LES particles (see Chapter 6). If *temp0les*<0, a single temperature bath is used for all atoms, otherwise separate thermostats are used for LES and non-LES particles. Default is -1, corresponding to a single (weak-coupling) temperature bath.

TEMPI Initial temperature. For the initial dynamics run, (NTX .lt. 3) the velocities are assigned from a Maxwellian distribution at TEMPI K. If TEMPI = 0.0, the velocities will be calculated from the forces instead. TEMPI has no effect if NTX .gt. 3. Default 0.0.

IG The seed for the random number generator. The MD starting velocity is dependent on the random number generator seed if NTX .lt. 3 .and. TEMPI .ne. 0.0. Default 71277.

TAUTP Time constant, in ps, for heat bath coupling for the system, if *ntt* = 1. Default is 1.0. Generally, values for TAUTP should be in the range of 0.5-5.0 ps, with a smaller value providing tighter coupling to the heat bath and, thus, faster heating and a less natural trajectory. Smaller values of TAUTP result in smaller fluctuations in kinetic energy, but larger fluctuations in the total energy. Values much larger than the length of the simulation result in a return to constant energy conditions.

GAMMA_LN The collision frequency γ, in ps^{-1}, when *ntt* = 3. A simple Leapfrog integrator is used to propagate the dynamics, with the kinetic energy adjusted to be correct for the harmonic oscillator case [67,68]. Note that it is not necessary that γ approximate the physical collision frequency. In fact, it is often advantageous, in terms of sampling or stability of integration, to use much smaller values. Default is 0 [68,69].

VRAND If *vrand>0* and *ntt=2*, the velocities will be randomized to temperature TEMP0 every *vrand* steps.

VLIMIT If not equal to 0.0, then any component of the velocity that is greater than abs(VLIMIT) will be reduced to VLIMIT (preserving the sign). This can be used to avoid occasional instabilities in molecular dynamics runs. VLIMIT should generally be set to a value like 20 (the default), which is well above the most probable velocity in a Maxwell-Boltzmann distribution at room temperature. A warning message will be printed whenever the velocities are modified. Runs that have more than a few such warnings should be carefully examined.

5.6.10. Pressure regulation.

In "constant pressure" dynamics, the volume of the unit cell is adjusted (by small amounts on each step) to make the computed pressure approach the target pressure, *pres0*. Equilibration with *ntp* > 0 is generally necessary to adjust the density of the system to appropriate values. Note that fluctuations in the instantaneous pressure on each step will appear to be large (several hundred bar), but the average value over many steps should be close to the target pressure. Pressure regulation only applies when Constant Pressure periodic boundary conditions are used (*ntb* = 2). Pressure coupling algorithms used in AMBER are of the "weak-coupling" variety, analogous to temperature coupling [64]. Please note: in general you will need to equilibrate the temperature to something like the final temperature using constant volume (*ntp*=0) *before* switching on constant pressure simulations to adjust the system to the correct density. If you fail to do this, the program will try to adjust the density too quickly, and bad things (such as SHAKE failures) are likely to happen.

NTP Flag for constant pressure dynamics. This option should be set to 1 or 2 when Constant Pressure periodic boundary conditions are used (NTB = 2).

- = 0 Used with NTB not = 2 (default); no pressure scaling
- = 1 md with isotropic position scaling
- = 2 md with anisotropic (x-,y-,z-) pressure scaling: this should only be used with orthogonal boxes (i.e. with all angles set to 90º). Anisotropic scaling is primarily intended for non-isotropic systems, such as membrane simulations, where the surface tensions are different in different directions; it is generally not appropriate for solutes dissolved in water.

PRES0 Reference pressure (in units of bars, where 1 bar ˜ 1 atm) at which the system is maintained (when NTP > 0). Default 1.0.

COMP compressibility of the system when NTP > 0. The units are in 1.0E-06/bar; a value of 44.6 (default) is appropriate for water.

TAUP Pressure relaxation time (in ps), when NTP > 0. The recommended value is between 1.0 and 5.0 psec. Default value is 1.0, but larger values may sometimes be necessary (if your trajectories seem unstable).

5.6.11. SHAKE bond length constraints.

NTC Flag for SHAKE to perform bond length constraints [70]. (See also NTF in the **Potential function** section. In particular, typically NTF = NTC.) The SHAKE option should be used for most MD calculations. The size of the MD timestep is determined by the fastest motions in the system. SHAKE removes the bond stretching freedom, which is the fastest motion, and consequently allows a larger timestep to be used. For water models, a special "three-point" algorithm is used [71]. Consequently, to employ TIP3P set NTF = NTC = 2. Since SHAKE is an algorithm based on dynamics, the minimizer is not aware of what SHAKE is doing; for this reason, minimizations generally should be carried out without SHAKE. One exception is short minimizations whose

purpose is to remove bad contacts before dynamics can begin.

= 1 SHAKE is not performed (default)

= 2 bonds involving hydrogen are constrained

= 3 all bonds are constrained (not available for parallel runs in *sander*)

TOL Relative geometrical tolerance for coordinate resetting in shake. Recommended maximum: <0.00005 Angstrom Default 0.00001.

JFASTW Fast water definition flag. By default, the system is searched for water residues, and special routines are used to SHAKE these systems [71].

= 0 Normal operation. Waters are identified by the default names (given below), unless they are redefined, as described below.

= 4 Do not use the fast SHAKE routines for waters.

The following variables allow redefinition of the default residue and atom names used by the program to determine which residues are waters.

WATNAM The residue name the program expects for water. Default `'WAT '`.

OWTNM The atom name the program expects for the oxygen of water. Default `'O '`.

HWTNM1 The atom name the program expects for the 1st H of water. Default `'H1 '`.

HWTNM2 The atom name the program expects for the 2nd H of water. Default `'H2 '`.

5.6.12. Water cap.

IVCAP Flag to control cap option. The "cap" refers to a spherical portion of water centered on a point in the solute and restrained by a soft half-harmonic potential. For the best physical realism, this option should be combined with *igb=10*, in order to include the reaction field of waters that are beyond the cap radius.

= 0 Cap will be in effect if it is in the *prmtop* file (default).

= 2 Cap will be inactivated, even if parameters are present in the *prmtop* file.

FCAP The force constant for the cap restraint potential.

5.6.13. NMR refinement options.

ISCALE Number of additional variables to optimize beyond the 3N structural parameters. (Default = 0). At present, this is only used with residual dipolar coupling restraints, as discussed in section SEVEN.

NOESKP The NOESY volumes will only be evaluated if mod(nstep, noeskp) = 0; otherwise the last computed values for intensities and derivatives will be used. (default = 1, i.e. evaluate volumes at every step)

IPNLTY

= 1 the program will minimize the sum of the absolute values of the errors; this is akin to minimizing the crystallographic R-factor (default).

= 2 the program will optimize the sum of the squares of the errors.

= 3 For NOESY intensities, the penalty will be of the form

$awt\,[I_c^{(1/6)} - I_o^{(1/6)}]^2$.

Chemical shift penalties will be as for *ipnlty=1*.

MXSUB Maximum number of submolecules that will be used. This is used to determine how much space to allocate for the NOESY calculations. Default 1.

SCALM "Mass" for the additional scaling parameters. Right now they are restricted to all have the same value. The larger this value, the slower these extra variables will respond to their environment. Default 100 amu.

PENCUT In the summaries of the constraint deviations, entries will only be made if the penalty for that term is greater than PENCUT. Default 0.1.

TAUSW For noesy volume calculations (*NMROPT = 2*), intensities with mixing times less that TAUSW (in seconds) will be computed using perturbation theory, whereas those greater than TAUSW will use a more exact theory. See the theory section (below) for details. To always use the "exact" intensities and derivatives, set TAUSW = 0.0; to always use perturbation theory, set TAUSW to a value larger than the largest mixing time in the input. Default is TAUSW of 0.1 second, which should work pretty well for most systems.

5.6.14. Particle Mesh Ewald.

The Particle Mesh Ewald (PME) method is always "on", unless *ntb* = 0, or *use_pme* is set to zero. PME is a fast implementation of the Ewald summation method for calculating the full electrostatic energy of a unit cell (periodic box) in a macroscopic lattice of repeating images. As implemented, the PME in AMBER bypasses the "old" pairlist creation and nonbonded energy and force evaluation, calling special PME functions to calculate the Lennard-Jones and electrostatic interactions. The PME method is fast since the reciprocal space Ewald sums are B-spline interpolated on a grid and since the convolutions necessary to evaluate the sums are calculated via fast Fourier transforms. Note that the accuracy of the PME is related to the density of the charge grid (NFFT1, NFFT2, and NFFT3), the spline interpolation order (ORDER), and the direct sum tolerance (DSUM_TOL); see the descriptions below for more information.

The `&ewald` namelist is read immediately after the `&cntrl` namelist. We have tried hard to make the defaults for these parameters appropriate for solvated simulations. *Please take care in changing any values from their defaults.* The `&ewald` namelist has the following variables:

NFFT1, NFFT2, NFFT3

These give the size of the charge grid (upon which the reciprocal sums are interpolated) in each dimension. Higher values lead to higher accuracy (when the DSUM_TOL is also lowered) but considerably slow the calculation. Generally it has been found that reasonable results are obtained when NFFT1, NFFT2 and NFFT3 are approximately equal to A, B and C, respectively,

leading to a grid spacing (A/NFFT1, etc) of 1.0 Å. Significant performance enhancement in the calculation of the fast Fourier transform is obtained by having each of the integer NFFT1, NFFT2 and NFFT3 values be a *product of powers* of 2, 3, and 5. If the values are not given, the program will chose values to meet these criteria.

ORDER The order of the B-spline interpolation. The higher the order, the better the accuracy (unless the charge grid is too coarse). The minimum order is 3. An order of 4 (the default) implies a cubic spline approximation which is a good standard value. Note that the cost of the PME goes as roughly the order to the third power.

VERBOSE Standard use is to have VERBOSE = 0. Setting VERBOSE to higher values (up to a maximum of 3) leads to voluminous output of information about the PME run.

EW_TYPE Standard use is to have EW_TYPE = 0 which turns on the particle mesh ewald (PME) method. When EW_TYPE = 1, instead of the approximate, interpolated PME, a *regular* Ewald calculation is run. The number of reciprocal vectors used depends upon RSUM_TOL, or can be set by the user. The exact Ewald summation is present mainly to serve as an accuracy check allowing users to determine if the PME grid spacing, order and direct sum tolerance lead to acceptable results. Although the cost of the exact Ewald method formally increases with system size at a much higher rate than the PME, it may be faster for small numbers of atoms (< 500). For larger, macromolecular systems, with > 500 atoms, the PME method is significantly faster.

DSUM_TOL This relates to the width of the direct sum part of the Ewald sum, requiring that the value of the direct sum at the Lennard-Jones cutoff value (specified in CUT as during standard dynamics) be less than DSUM_TOL. In practice it has been found that the relative error in the Ewald forces (RMS) due to cutting off the direct sum at CUT is between 10.0 and 50.0 times DSUM_TOL. Standard values for DSUM_TOL are in the range of 10^{-6} to 10^{-5}, leading to estimated RMS deviation force errors of 0.00001 to 0.0005. Default is 10^{-5}.

RSUM_TOL This serves as a way to generate the number of reciprocal vectors used in an Ewald sum. Typically the relative RMS reciprocal sum error is about 5-10 times RSUM_TOL. Default is 5 x 10^{-5}.

MLIMIT(1,2,3) This allows the user to explicitly set the number of reciprocal vectors used in a regular Ewald run. Note that the sum goes from -MLIMIT(2) to MLIMIT(2) and -MLIMIT(3) to MLIMIT(3) with symmetry being used in first dimension. Note also the sum is truncated outside an automatically chosen sphere.

EW_COEFF Ewald coefficient, in $Å^{-1}$. Default is determined by *dsum_tol* and *cutoff*. If it is explicitly inputed then that value is used, and *dsum_tol* is computed from *ew_coeff* and *cutoff*.

NBFLAG If *nbflag = 0*, construct the direct sum nonbonded list in the "old" way, *i.e.* update the list every *nsnb* steps. If *nbflag = 1* (the default when *imin = 0* or *ntb > 0*), *nsnb* is ignored, and the list is updated whenever any atom has moved more than 1/2 *skinnb* since the last list update.

SKINNB Width of the nonbonded "skin". The direct sum nonbonded list is extended to *cut + skinnb*, and the van der Waals and direct electrostatic interactions are truncated at *cut*. Default is 2.0 Å. Use of this parameter is required for energy conservation, and recommended for all PME runs.

NBTELL If *nbtell = 1*, a message is printed when any atom has moved far enough to trigger a list update. Use only for debugging or analysis. Default of 0 inhibits the message.

NETFRC The basic "smooth" PME implementation used here does not necessarily conserve momentum. If *netfrc = 1*, (the default) the total force on the system is artificially removed at every step. This parameter is set to 0 if minimization is requested.

USE_PME The default value of 1 means that the reciprocal part of the sum will be computed; if this is set to zero, the reciprocal part will be skipped.

VDWMETH Determines the method used for van der Waals interactions beyond those included in the direct sum. A value of 0 includes no correction; the default value of 1 uses a continuum model correction for energy and pressure.

EEDMETH Determines how the switch function for the direct sum Coulomb interaction is evaluated. The default value of 1 uses a cubic spline. A value of 2 implies a linear table lookup. A value of three implies use of an "exact" subroutine call. When *eedmeth=4*, no switch is used (*i.e.* the bare Coulomb potential is evaluated in the direct sum, cut off sharply at CUT). When *eedmeth=5*, there is no switch, and a distance-dependent dielectric is used (*i.e.* the distance dependence is $1/r^2$ rather than 1/r). The last two options are intended for non-periodic calculations, where no reciprocal term is computed.

EEDTBDNS Density of spline or linear lookup table, if *eedmeth* is 1 or 2. Default is 2500 points per unit.

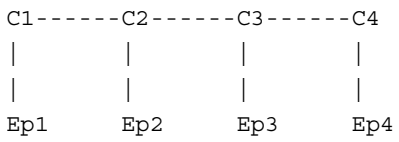
5.6.15. Extra point options.

Several parameters deal with "extra-points" (sometimes called lone-pairs), which are force centers that are not at atomic positions. These are currently defined as atoms with "EP" in their names. These input variables are really only for the convenience of force-field developers; *do not change the defaults unless you know what you are doing, and have read the code.* These variables are set in the `&ewald` namelist.

FRAMEON If *frameon* is set to 1, (default) the bonds, angles and dihedral interactions involving the lone pairs/extra points are removed except for constraints added during parm. The lone pairs are kept in ideal geometry relative to local atoms, and resulting torques are transferred to these atoms. To treat extra points as regular atoms, set frameon=0.

CHNGMASK If *chngmask*=1 (default), new 1-1, 1-2, 1-3 and 1-4 interactions are calculated. An extra point belonging to an atom has a 1-1 interaction with it, and participates in any 1-2, 1-3 or 1-4 interaction that atom has.

For example, suppose (excusing the geometry) C1,C2,C3,C4 form a dihedral and each has 1 extra point attached as below



The 1-4 interactions include C1−C4, Ep1−C4, C1−Ep4, and Ep1−Ep4. (To see a printout of all 1-1, 1-2, 1-3 and 1-4 interactions set verbose=1.) These interactions are masked out of nonbonds. Thus the amber mask list is rebuilt from these 1-1, 1-2, 1-3 and 1-4 pairs.

A separate list of 1-4 nonbonds is then compiled. This list does not agree in general with the above 1-4, since a 1-4 could also be a 1-3 if its in a ring. The rules in EPHI() are used to see who is included:

```
              DO 700 JN = 1,MAXLEN
                I3 = IP(JN+IST)
                K3T = KP(JN+IST)
                L3T = LP(JN+IST)
                IC0 = ICP(JN+IST)
                IDUMI = ISIGN(1,K3T)
                IDUML = ISIGN(1,L3T)
                KDIV = (2+IDUMI+IDUML)/4
                L3 = IABS(L3T)
                FMULN = FLOAT(KDIV)*FMN(IC0)
C
                II = (I3+3)/3
                JJ = (L3+3)/3
                IA1 = IAC(II)
                IA2 = IAC(JJ)
                IBIG = MAX0(IA1,IA2)
                ISML = MIN0(IA1,IA2)
                IC = IBIG*(IBIG-1)/2+ISML
C
C                 ----- CALCULATE THE 14-EEL ENERGY -----
C
                R2 = FMULN/CT(JN)
                R1 = SQRT(R2)
            ...........
```

so a pair in the 1-4 list is included if kdiv is > 0 and so is FMN(ic0); this is decided at startup. This decision logic is applied to the parent atoms, and if they are included, so are extra points attached. That is, in the above situation, if C1 and C4 pass the test, then C1−C4, Ep1−C4, C1−Ep4, and Ep1−Ep4 are included. Dihedrals involving the extra points are not tested since the decision is based solely on parent atoms.

The list of 1-4 nonbonds is also spit out if verbose=1.

5.6.16. Polarizable potentials.

The following parameters are relevant for *polarizable potentials*, that is, when *ipol* is set to 1 in the `&cntrl` namelist. These variables are set in the `&ewald` namelist.

INDMETH If indmeth is 0, 1, or 2 then the nonbond force is called iteratively until successive estimates of the induced dipoles agree to within DIPTOL (default 0.0001 debye) in the root mean square sense. The difference between indmeth = 0, 1, or 2 have to do with the level of extrapolation (1st, 2nd or 3rd-order) used from previous time steps for the initial guess for dipoles to begin the iterative loop. So far 2nd order (indmeth=1) seems to work best.

If indmeth = 3, use a Car-Parinello scheme wherein dipoles are assigned a fictitious mass and integrated each time step. This is much more efficient and is the current default. Note that this method is unstable for dt > 1 fs.

DIPTOL Convergence criterion for dipoles in the iterative methods. Default is 0.0001 Debye.

MAXITER For iterative methods (indmeth<3), this is the maximum number of iterations allowed per time step. Default is 20.

DIPMASS The fictitious mass assigned to dipoles. Default value is 0.33, which works well for 1fs time steps. If dipmass is set much below this, the dynamics are rapidly unstable. If set much above this the dynamics of the system are affected.

DIPTAU This is used for temperature control of the dipoles (for indmeth=3). If *diptau* is greater than 10 (ps units) temperature control of dipoles is turned off. Experiments so far indicate that running the system in NVE with no temperature control on induced dipoles leads to a slow heating, barely noticeable on the 100ps time scale. For runs of length 10ps, the energy conservation with this method rivals that of SPME for standard fixed charge systems. For long runs, we recommend setting a weak temperature control (e.g. 9.99 ps) on dipoles as well as on the atoms. Note that to achieve good energy conservation with iterative method, the diptol must be below 10^{-7} debye, which is much more expensive. Default is 11 ps (*i.e.* default is turned off).

IRSTDIP If indmeth=3, a restart file for dipole positions and velocities is written along with the restart for atomic coordinates and velocities. If irstdip=1, the dipolar positions and velocities from the inpdip file are read in. If irstdip=0, an iterative method is used for step 1, after which Car-Parrinello is used.

SCALDIP To scale 1-4 charge-dipole and dipole-dipole interactions the same as 1-4 charge-charge (i.e. divided by scee) set scaldip=1 (default). If scaldip=0 the 1-4 charge-dipole and dipole-dipole interactions are treated the same as other dipolar interactions (i.e. divided by 1).

5.6.17. Dipole Printing

By including a `&dipoles` namelist containing a series of groups, at the end of the input file, the printing of permanent, induced and total dipoles is enabled.

The X, Y and Z components of the dipole (in debye) for each group will be written to mdout every NTPR steps. In order to avoid ambiguity with charged groups all of the dipoles for a given group are calculated with respect to the centre of mass of that group.

It should be noted that the permanent, inducible and total dipoles will be printed regardless of whether a *polarizable potential* is in use. However, only the permanent dipole will have any physical meaning when *non-polarizable potentials* are in use.

It should also be noted that the groups used in the dipole printing routines are not exclusive to these routines and so the dipole printing procedure can only be used when group input is *not* in use for something else (*i.e.* restraints).

5.7. SECTION TWO: Weight change information.

This section of information is read (*if NMROPT > 0*) as a series of namelist specifications, with name "`&wt`". This namelist is read repeatedly until a namelist `&wt` statement is found with TYPE=END.

Overview of weight change variables	
variable	*description*
TYPE	Defines quantity being varied; valid options are listed below.
ISTEP1,ISTEP2	This change is applied over steps/iterations ISTEP1 through ISTEP2. If ISTEP2 = 0, this change will remain in effect from step ISTEP1 to the end of the run at a value of VALUE1 (VALUE2 is ignored in this case). *(default= both 0)*
VALUE1,VALUE2	Values of the change corresponding to ISTEP1 and ISTEP2, respectively. If ISTEP2=0, the change is fixed at VALUE1 for the remainder of the run, once step ISTEP1 is reached.
IINC IMULT	If IINC > 0, then the change is applied as a step function, with IINC steps/iterations between each change in the target VALUE (ignored if ISTEP2=0). If IINC =0, the change is done continuously. *(default=0)* If IMULT=0, then the change will be linearly interpolated from VALUE1 to VALUE2 as the step number increases from ISTEP1 to ISTEP2. *(default)* If IMULT=1, then the change will be effected by a series of multiplicative scalings, using a single factor, R, for all scalings. i.e. VALUE2 = (R**INCREMENTS) * VALUE1. INCREMENTS is the number of times the target value changes, which is determined by ISTEP1, ISTEP2, and IINC.

The remainder of this section describes the options for the TYPE parameter. For a few types of cards, the meanings of the other variables differ from that described above; such differences are noted below. Valid Options for TYPE (you must use uppercase) are:

BOND	Varies the relative weighting of bond energy terms.
ANGLE	Varies the relative weighting of valence angle energy terms.
TORSION	Varies the relative weighting of torsion (and J-coupling) energy terms. Note that any restraints defined in the input to the PARM program are included in the above. Improper torsions are handled separately (IMPROP).
IMPROP	Varies the relative weighting of the "improper" torsional terms. These are not included in TORSION.
VDW	Varies the relative weighting of van der Waals energy terms. This is equivalent to changing the well depth (epsilon) by the given factor.
HB	Varies the relative weighting of hydrogen-bonding energy terms.
ELEC	Varies the relative weighting of electrostatic energy terms.
NB	Varies the relative weights of the non-bonded (VDW, HB, and ELEC) terms.
ATTRACT	Varies the relative weights of the attractive parts of the van der waals and h-bond terms.
REPULSE	Varies the relative weights of the repulsive parts of the van der waals and h-bond terms.
RSTAR	Varies the effective van der Waals radii for the van der Waals (VDW) interactions by the given factor. Note that this is done by changing the relative attractive and repulsive coefficients, so ATTRACT/REPULSE should not be used over the same step range as RSTAR.
INTERN	Varies the relative weights of the BOND, ANGLE and TORSION terms. "Improper" torsions (IMPROP) must be varied separately.
ALL	Varies the relative weights of all the energy terms above (BOND, ANGLE, TORSION, VDW, HB, and ELEC; does not affect RSTAR or IMPROP).
REST	Varies the relative weights of *all* the NMR restraint energy terms.
RESTS	Varies the weights of the "short-range" NMR restraints. Short- range restraints are defined by the SHORT instruction (see below).
RESTL	Varies the weights of any NMR restraints which are not defined as "short range" by the SHORT instruction (see below). When no SHORT instruction is given, RESTL is equivalent to REST.
NOESY	Varies the overall weight for NOESY volume restraints. Note that this value multiplies the individual weights read into the "awt" array. (Only if NMROPT=2; see Section 4 below).
SHIFTS	Varies the overall weight for chemical shift restraints. Note that this value multiplies the individual weights read into the "wt" array. (Only if NMROPT=2; see section 4 below).
SHORT	Defines the short-range restraints. For this instruction, ISTEP1, ISTEP2, VALUE1, and VALUE2 have different meanings. A short-range restraint can be defined in two ways.

(1) If the residues containing each pair of bonded atoms comprising the restraint are close enough in the primary sequence:

$$\text{ISTEP1} \le \text{ABS(delta\_residue)} \le \text{ISTEP2},$$

where delta_residue is the difference in the numbers of the residues containing the pair of bonded atoms.

(2) If the distances between each pair of bonded atoms in the restraint fall within a prescribed range:

$$\text{VALUE1} \le \text{distance} \le \text{VALUE2}.$$

Only one SHORT command can be issued, and the values of ISTEP1, ISTEP2, VALUE1, and VALUE2 remain fixed throughout the run. However, if IINC>0, then the short-range interaction list will be re-evaluated every IINC steps.

TGTRMSD Varies the RMSD target value for targeted MD.

TEMP0 Varies the target temperature TEMP0.

TEMP0LES Varies the LES target temperature TEMP0LES.

TAUTP Varies the coupling parameter, TAUTP, used in temperature scaling when temperature coupling options NTT=1 is used.

CUT Varies the non-bonded cutoff distance.

NSTEP0 If present, this instruction will reset the initial value of the step counter (against which ISTEP1/ISTEP2 and NSTEP1/NSTEP2 are compared) to the value ISTEP1. An NSTEP0 instruction only has an effect at the beginning of a run. For this card (only) ISTEP2, VALUE1, VALUE2 and IINC are ignored. If this card is omitted, NSTEP0 = 0. This card can be useful for simulation restarts, where NSTEP0 is set to the final step on the previous run.

STPMLT If present, the NMR step counter will be changed in increments of STPMLT for each actual dynamics step. For this card, only VALUE1 is read. ISTEP1, ISTEP2, VALUE2, IINC, and IMULT are ignored. Default = 1.0.

DISAVE

ANGAVE

TORAVE If present, then by default time-averaged values (rather than instantaneous values) for the appropriate set of restraints will be used. DISAVE controls distance data, ANGAVE controls angle data, TORAVE controls torsion data.

See below for the functional form used in generating time-averaged data.

For these cards: VALUE1 = τ (characteristic time for exponential decay)
VALUE2 = POWER (power used in averaging; the nearest integer of value2 is used)

Note that the range (ISTEP1→ISTEP2) applies only to TAU; The value of POWER is not changed by subsequent cards with the same ITYPE field, and time-averaging will always be turned on for the entire run if one of these cards appears.

Note also that, due to the way that the time averaged internals are calculated, changing τ at any time after the start of the run will only affect the relative weighting of steps occurring after the change in τ.

Separate values for τ and POWER are used for bond, angle, and torsion averaging.

The default value of τ (if it is 0.0 here) is 1.0D+6, which results in no exponential decay weighting. Any value of $\tau \geq$ 1.D+6 will result in no exponential decay.
If DISAVE,ANGAVE, or TORAVE is chosen, one can still force use of an instantaneous value for specific restraints of the particular type (bond, angle, or torsion) by setting the IFNTYP field to "1" when the restraint is defined (IFNTYP is defined in section FOUR below).
If time-averaging for a particular class of restraints is being performed, all restraints of that class that are being averaged (that is, all restraints of that class except those for which IFNTYP=1) *must* have the same values of NSTEP1 and NSTEP2 (NSTEP1 and NSTEP2 are defined below).

(For these cards, IINC and IMULT are ignored)

See the discussion of time-averaged restraints following the input descriptions.

DISAVI

ANGAVI

TORAVI

ISTEP1: Ignored.

ISTEP2: Sets IDMPAV. If IDMPAV > 0, *and* a dump file has been specified (DUMPAVE is set in the file redirection section below), then the time-averaged values of the restraints will be written every IDMPAV steps. Only one value of IDMPAV can be set (corresponding to the first DISAVI/ANGAVI/TORAVI card with ISTEP2 > 0), and *all* restraints (even those with IFNTYP=1) will be "dumped" to this file every IDMPAV steps. The values reported reflect the current value of τ.

VALUE1: The integral which gives the time-averaged values is undefined for the first step. By default, for each time-averaged internal, the integral is assigned the current value of the internal on the first step. If VALUE1≠0, this initial value of internal r is reset as follows:

```
-1000. <  VALUE1 <    1000.:  Initial value = r_initial + VALUE
          VALUE1 <= -1000.:  Initial value = r_target + 1000.
 1000. <= VALUE1           :  Initial value = r_target - 1000.
```

r_target is the target value of the internal, given by R2+R3 (or just R3, if R2 is 0). VALUE1 is in angstroms for bonds, in degrees for angles.

VALUE2: This field can be used to set the value of τ used in calculating the time-averaged values of the internal restraints reported at the end of a simulation (if LISTOUT is specified in the redirection section below). By default, no exponential decay weighting is used in calculating the final reported values, regardless of what value of τ was used during the simulation. If VALUE2>0, then τ = VALUE2 will be used in calculating these final reported averages. Note that the value of VALUE2 = τ specified here only affects the reported averaged values in at the end of a simulation. It does not affect the time-averaged values used during the simulation (those are changed by the

VALUE1 field of DISAVE, ANGAVE and TORAVE instructions).

IINC: If IINC = 0, then forces for the class of time-averaged restraints will be calculated exactly as (dE/dr_ave) (dr_ave/dx). If IINC = 1, then then forces for the class of time-averaged restraints will be calculated as (dE/dr_ave) (dr(t)/dx). Note that this latter method results in a non-conservative force, and does not integrate to a standard form. But this latter formulation helps avoid the large forces due to the (1+*i*) term in the exact derivative calculation--and may avert instabilities in the molecular dynamics trajectory for some systems. See the discussion of time-averaged restraints following the input description.

Note that the DISAVI, ANGAVI, and TORAVI instructions will have no affect unless the corresponding time average request card (DISAVE, ANGAVE or TORAVE, respectively) is also present.

DUMPFREQ Istep1 is the only parameter read, and it sets the frequency at which the coordinates in the distance or angle restraints are dumped to the file specified by the DUMPAVE command in section THREE.

(For these cards, ISTEP1 and IMULT are ignored).

END END of this section.

NOTES:

(1) All weights are relative to a default of 1.0 in the standard force field.

(2) Weights are not cumulative.

(3) For any range where the weight of a term is not modified by the above, the weight reverts to 1.0. For any range where TEMP0, SOFTR or CUTOFF is not specified, the value of the relevant constant is set to that specified in the input file.

(4) If a weight is set to 0.0, it is set internally to 1.0D-7. This can be overridden by setting the weight to a negative number. In this case, a weight of exactly 0.0 will be used. *However,* if any weight is set to exactly 0.0, it cannot be changed again during this run of the program.

(5) If two (or more) cards change a particular weight over the same range, the weight given on the last applicable card will be the one used.

(6) Once any weight change for which NSTEP2=0 becomes active (i.e. one which will be effective for the remainder of the run), the weight of this term cannot be further modified by other instructions.

(7) Changes to RSTAR result in exponential weighting changes to the attractive and repulsive terms (proportional to the scale factor**6 and **12, respectively). For this reason, scaling RSTAR to a very small value (e.g. ≤0.1) may result in a zeroing-out of the vdw term.

5.8. SECTION THREE: File redirection commands.

Input/output redirection information can be read as described here. Redirection cards must follow the end of the SECTION TWO input. Redirection card input is terminated by the first non-blank line which does not start with a recognized redirection TYPE (e.g. LISTIN, LISTOUT, etc.).

The format of the redirection cards is

TYPE = filename

where TYPE is any valid redirection keyword (see below), and filename is any character string. The equals sign ("=") is required, and TYPE must be given in *uppercase* letters.

Valid redirection keywords are:

LISTIN	An output listing of the restraints which have been read, and their deviations from the target distances *before* the simulation has been run. By default, this listing is not printed. If POUT is used for the filename, these deviations will be printed in the normal output file.
LISTOUT	An output listing of the restraints which have been read, and their deviations from the target distances *_after* the simulation has finished. By default, this listing is not printed. If POUT is used for the filename, these deviations will be printed in the normal output file.
DISANG	The file from which the distance and angle restraint information described below (Section 4) will be read.
NOESY	File from which NOESY volume information (Section 5), if any, will be read.
SHIFTS	File from which chemical shift information (Section 6), if any, will be read.
PCSHIFT	File from which paramagnetic shift information (Section 6), if any, will be read.
DIPOLE	File from which residual dipolar couplings (Section 7), if any, will be read.
DUMPAVE	File to which the time-averaged values of all restraints will be written. If DISAVI / ANGAVI / TORAVI has been used to set IDMPAV≠0, then averaged values will be output. If the DUMPFREQ command has been used, the instantaneous values will be output.

5.9. SECTION FOUR: Distance, angle and torsional restraints.

Distance and angle restraints are read from the DISANG file if *nmropt* > 0. Namelist rst ("`&rst`") contains the following variables; it is read repeatedly until a namelist `&rst` statement is found with IAT(1)=0, or until reaching the end of the DISANG file.

If you wish to include weight changes but have no internal constraints, set *nmropt*=1, but do not include a DISANG line in section THREE. (Note that, unlike earlier versions of Amber, the `&rst` namelists must be in the DISANG file, and not in the *mdin* file.)

In many cases, the user will not prepare this section of the input by hand, but will use the auxiliary programs *makeDIST_RST, makeANG_RST* and *makeCHIR_RST* to prepare input from simpler files.

Variables in the `&rst` namelist:

IAT(1)→IAT(4) *If IRESID = 0 (normal operation):*

The atoms defining the restraint. If IAT(3) ≤0, this is a distance restraint. If IAT(4) ≤0, this is an angle restraint. Otherwise, this is a torsional (or J-coupling, if desired) restraint.

If this is a distance restraint, and IAT1 <0, then a group of atoms is defined below, and the coordinate-averaged position of this group will be used in place of the coordinates of atom 1 [IAT(1)]. Similarly, if IAT(2) < 0, a group of atoms will be defined below whose coordinate-averaged position will be used in place of the coordinates for atom 2 [IAT(2)].

If IRESID=1:

IAT(1) → IAT(4) point to the numbers of the *residues* containing the atoms comprising the internal. Residue numbers are the absolute numbers in the entire system. In this case, the variables ATNAM(1) → ATNAM(4) must be specified, and give the character names of the desired atoms within the respective residues.

If IAT(1) < 0 or IAT(2) < 0, then group input will still be read in place of the corresponding atom, as described below.

Defaults for IAT(1)→IAT(4) are 0.

ATNAM If IRESID = 1, then the character names of the atoms defining the internal are contained in ATNAM(1)→ATNAM(4). Residue IAT(1) is searched for atom name ATNAM(1); residue IAT(2) is searched for atom name ATNAM(2); etc. On machines using the portable namelist code, the form is atnam(1)='AT1',atnam(2)='AT2' etc, otherwise the form atnam='AT1','AT2' etc can be used.

Defaults for ATNAM(1)→ATNAM(4) are ' '.

IRESID Indicates whether IAT(I) points to an atom # or a residue #. See descriptions of IAT() and ATNAM() above.

Default = 0.

NSTEP1

NSTEP2 This restraint is applied for steps/iterations NSTEP1 through NSTEP2. If NSTEP2 = 0, the restraint will be applied from NSTEP1 through the end of the run. Note that the first step/iteration is considered step zero (0).

Defaults for NSTEP1, NSTEP2 are both 0.

IRSTYP Normally, the restraint target values defined below (R1→R4) are used directly. If IRSTYP = 1, the values given for R1→R4 define relative displacements from the current value (value determined from the starting coordinates) of the restrained internal. For example, if IRSTYP=1, the current value of a restrained distance is 1.25, and R1 (below) is -0.20, then a value of R1=1.05 will be used.

Default is IRSTYP=0.

IALTD Determines what happens when a distance restraint gets very large. If IALTD=1, then the potential "flattens out", and there is no force for large violations; this allows for errors in constraint lists, but might tend to ignore constraints that *should* be included to pull a bad initial structure towards a more correct one. When IALTD=0 the penalty energy continues to rise for large violations. See below for the detailed functional forms that are used for distance restraints. Set IALTD=0 to recover the behavior of earlier versions of *sander*. Default value is 0, or the last value that was explicitly set in a previous restraint. This value is set to 1 if *makeDIST_RST* is called with the *-altdis* flag.

IFVARI If IFVARI > 0, then the force constants/positions of the restraint will vary with step number. Otherwise, they are constant throughout the run. If IFVARI >0, then the values R1A→R4A, RK2A, and RK3A must be specified (see below).

Default is IFVARI=0.

NINC If IFVARI > and NINC > 0, then the change in the target values of of R1→R4 and K2,K3 is applied as a step function, with NINC steps/ iterations between each change in the target values. If NINC = 0, the change is effected continuously (at every step).

Default for NINC is the value assigned to NINC in the most recent namelist where NINC was specified. If NINC has not been specified in any namelist, it defaults to 0.

IMULT If IMULT=0, and the values of force constants RK2 and RK3 are changing with step number, then the changes in the force constants will be linearly interpolated from rk2→rk2a and rk3→rk3a as the step number changes.

If IMULT=1 and the force constants are changing with step number, then the changes in the force constants will be effected by a series of multiplicative scalings, using a single factor, R, for all scalings. *i.e.*

rk2a = R**INCREMENTS * rk2
rk3a = R**INCREMENTS * rk3.

INCREMENTS is the number of times the target value changes, which is determined by NSTEP1, NSTEP2, and NINC.

Default for IMULT is the value assigned to IMULT in the most recent namelist where IMULT was specified. If IMULT has not been specified in any

namelist, it defaults to 0.

R1→R4

RK2,RK3

R1A→R4A

RK2A,RK3A If IALTD=0, the restraint is a well with a square bottom with parabolic sides out to a defined distance, and then linear sides beyond that. Force constants are in units of kcal/mol. If R is the value of the restraint in question:

R < r1 Linear, with the slope of the "left-hand" parabola at the point R=r1.

r1 <= R < r2 Parabolic, with force constant k2. E=0 at R=r2.

r2 <= R < r3 E = 0.

r3 <= R < r4 Parabolic, with force constant K3. E=0 at R=r3.

r4 <= R Linear, with the slope of the "right-hand" parabola at the point R=r4.

For torsional restraints, the value of the torsion is translated by +-n*360, if necessary, so that it falls closest to the mean of r2 and r3.

Specified distances are in Angstroms. Specified angles are in degrees. Force constants for distances are in kcal/mol-Å^2 Force constants for angles are in kcal/mol-rad^2. (Note that angle positions are specified in degrees, but force constants are in radians, consistent with typical reporting procedures in the literature).

If IALTD=1, distance restraints are interpreted in a slightly different fashion. Again, If R is the value of the restraint in question:

R < r2 Parabolic, with force constant k2. E=0 at R=r2.

r2 <= R < r3 E = 0.

r3 <= R < r4 Parabolic, with force constant k3. E=0 at R=r3.

r4 <= R Hyperbolic, with energy k3*[b/(R-r3) + a], where $a = 3(r4 - r3)^2$ and $b = -2(r4 - r3)^3$. This function matches smoothly to the parabola at R = r3, and tends to an asymptote of ak_3 are large R. The functional form is adapted from that suggested by Michael Nilges, *Prot. Eng.* **2,** 27-38 (1988). Note that if IALTD=1, the value of r1 is ignored.

IFVARI = 0 The values of r1→r4, rk2, and rk3 will remain constant throughout the run.

IFVARI > 0 The values r1a, r2a, r3a, r4a, r2ka and r3ka are also used. These variables are defined as for r1→r4 and rk2, rk3, but correspond to the values appropriate for NSTEP = NSTEP2: e.g., if IVARI >0, then the value of r1 will vary between NSTEP1 and NSTEP2, so that, e.g. r1(NSTEP1) = r1 and r1(NSTEP2) = r1a. Note that you *must* specify an explicit value for *nstep1* and *nstep2* if you use this option.

Defaults for r1→r4,rk2,rk3,r1a→r4a,rk2a and rk3a are the values assigned to them in the most recent namelist where they were specified. They should always be specified in the first `&rst` *namelist.*

(IGR1(i),i=1→200)

If IAT(1) < 0 and IAT(3)=IAT(4)=0, then IGR1() gives the atoms defining the group whose coordinate averaged position is used to define "atom 1" in a distance restraint. If IRESID = 0, absolute atom numbers are specified by the elements of IGR1(). If IRESID = 1, then IGR1(I) specifies the number of the residue containing atom I, and the name of atom I must be specified using GRNAM1(I). A maximum of 200 atoms are allowed in any group. Only specify those atoms which are needed.

RJCOEF(1)→RJCOEF(3)

By default, 4-atom sequences specify torsional restraints. It is also possible to impose restraints on the vicinal ^{3}J-coupling value related to the underlying torsion. J is related to the torsion τ by the approximate Karplus relationship: $J = A\cos^2(\tau) + B\cos(\tau) + C$. If you specify a non-zero value for either RJCOEF(1) or RJCOEF(2), then a J-coupling restraint, rather than a torsional restraint, will be imposed. At every MD step, J will be calculated from the Karplus relationship with A = RJCOEF(1), B = RJCOEF(2) and C = RJCOEF(3). In this case, the target values (R1->R4, R1A->R4A) and force constants (RK2, RK3, RK2A, RK3A) refer to J-values for this restraint. RJCOEF(1)->RJCOEF(3) must be set individually for each torsion for which you wish to apply a J-coupling restraint, and RJCOEF(1)->RJCOEF(3) may be different for each J-coupling restraint.

With respect to other options and reporting, J-coupling restraints are treated identically to torsional restraints. This means that if time-averaging is requested for torsional restraints, it will apply to J-coupling restraints as well. The J-coupling restraint contribution to the energy is included in the "torsional" total. And changes in the relative weights of the torsional force constants also change the relative weights of the J-coupling restraint terms.

Setting RJCOEF has no effect for distance and angle restraints.

Defaults for RJCOEF(1)->RJCOEF(3) are 0.0.

(IGR2(i),i=1→200)

If IAT(2) < 0 and IAT(3)=IAT(4)=0, then IGR1 gives the atoms defining the group whose coordinate averaged position is used to define "atom 2" in a distance restraint. If IRESID = 0, absolute atom numbers are specified by the elements of IGR2(). If IRESID = 1, then IGR2(I) specifies the number of the residue containing atom I, and the name of atom I must be specified using GRNAM1(I). A maximum of 200 atoms are allowed in any group. Only specify those atoms which are needed.

Default value for any unspecified element of IGR1 or IGR2 is 0.

(GRNAM1(i),i=1→200)

(GRNAM2(i),i=1→200)

If group input is being specified (IAT(1) or IAT(2) < 0 and IAT(3)=IAT(4)=0), *and* IRESID = 1, then the character names of the atoms defining the group are contained in GRNAM1(i) or GRNAM2(i)), as described above. In the case IAT(1) < 0, each residue IGR1(i) is searched for an atom name GRNAM1(i) and added to the first group list. In the case IAT(2) < 0, each residue IGR2(i) is searched for an atom name GRNAM2(i) and added to the second group list.

Defaults for GRNAM1(i) and GRNAM2(i) are ' '.

IR6 If a group coordinate-averaged position is being used (see IGR1 and IGR2 above), the average position can be calculated in either of two manners: If IR6 = 0, center-of-mass averaging will be used. If IR6=1, the $< r^{-6} >^{-1/6}$ average of all interaction distances to atoms of the group will be used.

Default for IR6 is the value assigned to IR6 in the most recent namelist where IR6 was specified. If IR6 has not been specified in any namelist, it defaults to 0.

IFNTYP If time-averaged restraints have been requested (see DISAVE/ANGAVE/TORAVE above), they are, by default, applied to all restraints of the class specified. Time-averaging can be overridden for specific internals of that class by setting IFNTYP for that internal to 1. IFNTYP has no effect if time-averaged restraint are not being used.

Default value is IFNTYP=0.

IXPK

NXPK These are user-defined integers than can be set for each constraint. They are typically the "peak number" and "spectrum number" associated with the cross-peak that led to this particular distance restraint. Nothing is ever done with them except to print them out in the "violation summaries", so that NMR people can more easily go from a constraint violation to the corresponding peak in their spectral database. Default values are zero.

Namelist `&rst` is read for each restraint. Restraint input ends when a namelist statement with iat(1) = 0 (or iat(1) not specified) is found. Note that comments can precede or follow any namelist statement, allowing comments and restraint definitions to be freely mixed.

5.10. SECTION FIVE: NOESY volume restraints.

After the previous section, NOESY volume restraints may be read. This data described in this section is only read if NMROPT = 2. The molecule may be broken in overlapping submolecules, in order to reduce time and space requirements. Input *for each submolecule* consists of namelist "`&noeexp`", followed *immediately* by standard AMBER "group" cards defining the atoms in the submolecule. In addition to the submolecule input ("`&noeexp`"), you may also need to specify some additional variables in the `cntrl` namelist in section ONE; see the "NMR variables" description in that section.

In many cases, the user will not prepare this section of the input by hand, but will use the auxiliary program *makeDIST_RST* to prepare input from simpler files.

Variables in the `&noeexp` namelist:

For each submolecule, the namelist "`&noeexp`" is read (either from *stdin* or from the NOESY redirection file) which contains the following variables. There are no effective defaults for *npeak, emix, ihp, jhp,* and *aexp*: you must specify these.

NPEAK*(imix)* Number of peaks for each of the "imix" mixing times; if the last mixing time is *mxmix*, set NPEAK(*mxmix+1*) = -1. End the input when NPEAK(1) < 0.

EMIX*(imix)* Mixing times (in seconds) for each mixing time.

IHP*(imix,ipeak)*

JHP*(imix,ipeak)* Atom numbers for the atoms involved in cross-peak "ipeak" at mixing time "imix"

AEXP(imix,ipeak) Experimental target integrated intensity for this cross peak. If AEXP is negative, this cross peak is part of a set of overlapped peaks. The computed intensity is added to the peak that follows; the next time a peak with AEXP > 0 is encountered, the running sum for the calculated peaks will be compared to the value of AEXP for that last peak in the list. In other words, a set of overlapped peaks is represented by one or more peaks with AEXP < 0 followed by a peak with AEXP > 0. The computed total intensity for these peaks will be compared to the value of AEXP for the final peak.

ARANGE*(imix,ipeak)*
"Uncertainty" range for this peak: if the calculated value is within ±ARANGE of AEXP, then no penalty will be assessed. Default uncertainties are all zero.

AWT*(imix,ipeak)* Relative weight for this cross peak. Note that this will be multiplied by the overall weight given by the NOESY weight change cards in the weight changes section (Section 1). Default values are 1.0, unless INVWT1,INVWT2 are set (see below), in which case the input values of AWT are ignored.

INVWT1,INVWT2
Lower and upper bounds on the weights for the peaks respectively, such that the relative weight for each peak is 1/intensity if 1/intensity lies between the lower and upper bounds. This is the intensity after being scaled by *oscale*. The inverse weighing scheme adopted by this option prevents placing too much influence on the strong peaks at the expense of weaker peaks and was

previously invoked using the compilation flag "INVWGT". Default values are INVWT1=INVWT2=1.0, placing equal weights on all peaks.

OMEGA — Spectrometer frequency, in Mhz. Default is 500. It is possible for different sub-molecules to have different frequencies, but omega will only change when it is explicitly re-set. Hence, if all of your data is at 600 Mhz, you need only set *omega* to 600. in the first submolecule.

TAUROT — Rotational tumbling time of the molecule, in nsec. Default is 1.0 nsec. Like *omega*, this value is "sticky", so that a value set in one submolecule will remain until it is explicitly reset.

TAUMET — Correlation time for methyl jump motion, in ns. This is only used in computing the intra-methyl contribution to the rate matrix. The ideas of Woessner are used, specifically as recommended by Kalk & Berendsen [72]. Default is 0.0001 ns, which is effectively the fast motion limit. The default is consistent with the way the rest of the rate matrix elements are determined (also in the fast motion limit,) but probably is not the best value to use, since methyl groups appear to have T1 values that are systematically shorter than other protons, and this is likely to arise from the fact that the methyl correlation time can be near to the inverse of the spectrometer frequency. A value of 0.02 - 0.05 ns is probably better than 0.0001, but this is still an active research area, and you are on your own here, and should consult the literature for further discussion [73]. As with *omega*, *taumet* can be different for different submolecules, but will only change when it is explicitly re-set.

ID2O — Flag for determining if exchangeable protons are to be included in the spin-diffusion calculation. If ID2O=0 (default) then all protons are included. If ID2O=1, then all protons bonded to nitrogen or oxygen are assumed to not be present for the purposes of computing the relaxation matrix. No other options exist at present, but they could easily be added to the subroutine *indexn*. Alternatively, you can manually rename hydrogens in the *prmtop* file so that they do not begin with "H": such protons will not be included in the relaxation matrix. (*Note:* for technical reasons, the HOH proton of tyrosine must always be present, so setting ID2O=1 will not remove it; we hope that this limitation will be of minor importance to most users.) The *id2o* variable retains its value across namelist reads, *i.e.* its value will only change if it is explicitly reset.

OSCALE — overall scaling factor between experimental and computed volume units. The experimental intensities are multiplied by *oscale* before being compared to calculated intensities. This means that the weights WNOESY and AWT always refer to "theoretical" intensity scales rather than to the (arbitrary) experimental units. The *oscale* variable retains its value across namelist reads, *i.e.* its value will only change if it is explicitly reset. The initial (default) value is 1.0.

The atom numbers *ihp* and *jhp* are the absolute atom numbers. For methyl groups, use the number of the last proton of the group; for the delta and epsilon protons of aromatic rings, use the delta-2 or epsilon-2 atom numbers. Since this input requires you to know the absolute atom numbers assigned by AMBER to each of the protons, you may wish to use the separate *makeDIST_RST* program which provides a facility for more turning human-readable input into the required file for *sander*.

Following the `&noeexp` namelist, give the AMBER "group" cards that identify this submolecule. This combination of "`&noeexp`" and "group" cards can be repeated as often as needed for many submolecules, subject to the limits described in the *nmr.h* file. As mentioned above, this input section ends when NPEAK(1) < 0, or when and end-of-file is reached.

5.11. SECTION SIX: Chemical shift restraints.

After reading NOESY restraints above (if any), read the chemical shift restraints in namelist *&shf*, or the pseudocontact restraints in namelist *&pcshift*. Reading this input is triggered by the presence of a SHIFTS or PCSHIFT line in section THREE. In many cases, the user will not prepare this section of the input by hand, but will use the auxiliary programs *makeSHF* or *fantasian* to prepare input from simpler files.

Variables in the `&shf` namelist. (Defaults are only available for *shrang*, *wt*, *nter*, and *shcut*; you must specify the rest.)

NRING — Number of rings in the system.

NATR*(i)* — Number of atoms in the *i-th* ring.

IATR*(j,i)* — Absolute atom number for the *j-th* atom of the *i-th* ring.

NAMR*(i)* — Eight-character string that labels the *i-th* ring. The first three characters give the residue name (in caps); the next three characters contain the residue number (right justified); column 7 is blank; column 8 may optionally contain an extra letter to distinguish the two rings of trp, or the 5 or 8 rings of the heme group.

STR*(i)* — Ring current intensity factor for the *i-th* ring. Older values are summarized by Cross and Wright [74]; more recent empirical parametrizations seem to give improved results [75,76].

NPROT — Number of protons for which penalty functions are to be set up.

IPROT*(i)* — Absolute atom number of the *i-th* proton whose shifts are to be evaluated. For equivalent protons, such as methyl groups or rapidly flipping phenylalanine rings, enter all two or three atom numbers in sequence; averaging will be controlled by the *wt* parameter, described below.

OBS*(i)* — Observed secondary shift for the *i-th* proton. This is typically calculated as the observed value minus a random coil reference value.

SHRANG*(i)* — "Uncertainty" range for the observed shift: if the calculated shift is within ±SHRANG of the observed shift, then no penalty will be imposed. The default value is zero for all shifts.

WT*(i)* — Weight to be assigned to this penalty function. Note that this value will be multiplied by the overall weight (if any) given by the SHIFTS command in the assignment of weights (above). Default values are 1.0. For sets of equivalent protons, give a negative weight for all but the last proton in the group; the last proton gets a normal, positive value. The average computed shift of the group will be compared to *obs* entered for the last proton.

SHCUT Values of calculated shifts will be printed only if the absolute error between calculated and observed shifts is greater than this value. *Default = 0.3 ppm.*

NTER Residue number of the N-terminus, for protein shift calculations; *default = 1.*

CTER Residue number of the C-terminus, for protein shift calculations. Believe it or not, the current code cannot figure this out for itself.

The PCSHIFT module allows the inclusion of pseudocontact shifts as constraints in energy minimization and molecular dynamics calculations on paramagnetic molecules. The pseudocontact shift depends on the magnetic susceptibility anisotropy of the metal ion and on the location of the resonating nucleus with respect to the axes of the magnetic susceptibility tensor. For the nucleus i, it is given by:

$$\delta^i_{pc} = \sum_j \frac{1}{12\pi r^3_{ij}} \left[\Delta\chi^j_{ax}(3n^2_{ij} - 1) + (3/2)\Delta\chi^j_{rh}(l^2_{ij} - m^2_{ij}) \right]$$

where l_{ij}, m_{ij}, and n_{ij} are the direction cosines of the position vector of atom i with respect to the j-th magnetic susceptibility tensor coordinate system, r_{ij} is the distance between the j-th paramagnetic center and the proton i, *jax* and *jrh* are the axial and the equatorial anisotropies of the magnetic susceptibility tensor of the j-th paramagnetic center. For a discussion, see: Lucia Banci, Ivano Bertini, Giovanni Gori-Savellini, Andrea Romagnoli, Paola Turano, Mauro Andrea Cremonini, Claudio Luchinat and Harry B. Gray "Pseudocontact shifts as Constraints for Energy minimization and molecular dynamics calculations on solution structures of paramagnetic metalloproteins", Proteins: Structure, Function and Genetics, in press.

The PCSHIFT module to be used needs a namelist file which includes information on the magnetic susceptibility tensor and on the paramagnetic center, and a line of information for each nucleus. This module allows to include more than one paramagnetic center in the calculations. To include pseudocontact shifts as constraints in energy minimization and molecular dynamics calculations the NMROPT flag should be set to 2, and a *PCSHIFT=filename* statement entered in section THREE.

To perform molecular dynamics calculations it is necessary to eliminate the rotational and translational degree of freedom about the center of mass (this because during molecular dynamics calculations the relative orientation between the external reference coordinate system and the magnetic anisotropy tensor coordinate system has to be fixed).This option can be obtained with the NSCM flag of *sander*.

Variables in the `pcshift` **namelist.**

NPROT number of pseudocontact shift constraints.

NME number of paramagnetic centers.

NMPMC name of the paramagnetic atom

OPTPHI(n)

OPTTET(n)

OPTOMG(n)

OPTA1(n)

OPTA2(n) the five parameters of the magnetic anisotropy tensor for each paramagnetic center.

OPTKON force constant for the pseudocontact shift constraints

Following this, there is a line for each nucleus for which the pseudocontact shift information is given has to be added. Each line contains :

IPROT(i) atom number of the i-th proton whose shift is to be used as constraint.

OBS(i) observed pseudocontact shift value, in ppm

WT(i) relative weight

TOLPRO(i) relative tolerance ix mltpro

MLTPRO(i) multiplicity of the NMR signal (for example the protons of a methyl group have mltprot(i)=3)

Example. Here is a &pcshf namelist example: a molecule with three paramagnetic centers and 205 pseudocontact shift constraints.

```
&pcshf
nprot=205,
nme=3,
nmpcm='FE ',
optphi(1)=-0.315416,
opttet(1)=0.407499,
optomg(1)=0.0251676,
opta1(1)=-71.233,
opta2(1)=1214.511,
optphi(2)=0.567127,
opttet(2)=-0.750526,
optomg(2)=0.355576,
opta1(2)=-60.390,
opta2(2)=377.459,
optphi(3)=0.451203,
opttet(3)=-0.0113097,
optomg(3)=0.334824,
opta1(3)=-8.657,
opta2(3)=704.786,
optkon=30,
iprot(1)=26, obs(1)=1.140, wt(1)=1.000, tolpro(1)=1.00, mltpro(1)=1,
iprot(2)=28, obs(2)=2.740, wt(2)=1.000, tolpro(2)=.500, mltpro(2)=1,
iprot(3)=30, obs(3)=1.170, wt(3)=1.000, tolpro(3)=.500, mltpro(3)=1,
iprot(4)=32, obs(4)=1.060, wt(4)=1.000, tolpro(4)=.500, mltpro(4)=3,
iprot(5)=33, obs(5)=1.060, wt(5)=1.000, tolpro(5)=.500, mltpro(5)=3,
iprot(6)=34, obs(6)=1.060, wt(6)=1.000, tolpro(6)=.500, mltpro(6)=3,
...
```

```
 ...
 iprot(205)=1215, obs(205)=.730, wt(205)=1.000, tolpro(205)=.500,
     mltpro(205)=1,
 /
```

An `mdin` file that might go along with this, to perform a maximum of 5000 minimization cycles, starting with 500 cycles of steepest descent. PCSHIFT=./pcs.in redirects the input from the namelist "pcs.in" which contains the pseudocontact shift information.

```
Example of minimization including pseudocontact shift constraints
 &cntrl
 ibelly=0,imin=1,ntpr=100,
 ntwx=100,ntwe=100,ioutfm=0,ntr=0,maxcyc=500,
 ncyc=50,ntmin=1,dx0=0.0001,
 drms=.1,cut=10.,scee=2.0,
 nmropt=2,pencut=0.1, ipnlty=2,
 /
 &wt type='REST', istep1=0,istep2=1,value1=0.,
     value2=1.0,  /
 &wt type='END'  /
DISANG=./noe.in
PCSHIFT=./pcs.in
LISTOUT=POUT
```

5.12. SECTION SEVEN: Direct dipolar coupling restraints

Energy restraints based on direct dipolar coupling constants are entered in this section. All variables are in the namelist `&align`; reading of this section is triggered by the presence of a DIPOLE line in Section THREE of the input.

When dipolar coupling restraints are turned on, the five unique elements of the alignment tensor are treated as additional variables, and are optimized along with the structural parameters. Their effective masses are determined by the *scalm* parameter entered in Section ONE. Unlike some other programs, the variables used are the Cartesian components of the alignment tensor in the axis system defined by the molecule itself: *e.g.* $S_{mn} \equiv 10^5 < (3\cos\theta_m \cos\theta_n - \delta_{mn})/2 >$, where θ_x is the angle between the x axis and the spectrometer field [77]. The factor of 10^5 is just to make the values commensurate with atomic coordinates, since both the coordinates and the alignment tensor values will be updated during the refinement. The calculated dipolar splitting is then

$$D_{calc} = -\left(\frac{10^{-5}\gamma_i\gamma_j h}{2\pi^2 r_{ij}^3}\right) \sum_{m,n=x,y,z} \cos\phi_m \cdot S_{mn} \cdot \cos\phi_n$$

where ϕ_x is the angle between the internuclear vector and the x axis. Geometrically, the splitting is proportional to the transformation of the alignment tensor onto the internuclear axis. This is

just Eqs. (5) and (13) of the above reference, with any internal motion corrections (which might be a part of S_{system}) set to unity. If there is an internal motion correction which is the same for all observations, this can be assimilated into the alignment tensor. The current code does not allow for variable corrections for internal motion, but this is coming. See ref. [78] for a fuller discussion of these issues.

At the end of the calculation, the alignment tensor is diagonalized to obtain information about its principal components. This allows the alignment tensor to be written in terms of the "axial" and "rhombic" components that are often used to describe alignment.

Variables in the `&align` namelist.

NDIP Number of observed dipolar coupling restraints to be used as restraints.

ID,JD Atom numbers of the two atoms involved in the dipolar coupling.

DOBSL, DOBSU Limiting values for the observed dipolar splitting, in Hz. If the calculated coupling is less than *dobsl*, the energy penalty is proportional to $(D_{calc} - D_{obs,l})^2$; if it is larger than *dobsu*, the penalty is proportional to $(D_{calc} - D_{obs,u})^2$. Calculated values between *dobsl* and *dobsu* are not penalized. Note that *dobsl* must be less than *dobsu*; for example, if the observed coupling is -6 Hz, and a 1 Hz "buffer" is desired, you could set *dobsl* to -7 and *dobsu* to -5.

DWT The relative weight of each observed value. Default is 1.0. The penalty function is thus:

$$E^i_{align} = D^i_{wt}(D^i_{calc} - D^i_{obs(u,l)})^2$$

where D_{wt} may vary from one observed value to the next. Note that the default value is arbitrary, and a smaller value may be required to avoid overfitting the dipolar coupling data [78].

DATASET Each dipolar peak can be associated with a "dataset", and a separate alignment tensor will be computed for each dataset. This is generally used if there are several sets of experiments, each with a different sample or temperature, etc., that would imply a different value for the alignment tensor. By default, there is one dataset to which each observed value is assigned.

NUM_DATASETS The number of datasets in the constraint list. Default is 1.

S11,S12,S13,S22,S23

Initial values for the Cartesian components of the alignment tensor. The tensor is traceless, so S33 is calculated as −(S11+S22). In order to have the order of magnitude of the S values be roughly commensurate with coordinates in Angstroms, the alignment tensor values must be multiplied by 10^5.

GIGJ Product of the nuclear "g" factors for this dipolar coupling restraint. These are related to the nuclear gyromagnetic rations by $\gamma_N = g_N \beta_N / \hbar$. Common values are ^{1}H = 5.5856, ^{13}C = 1.4048, ^{15}N = -0.5663, ^{31}P = 2.2632.

DIJ The internuclear distance for observed dipolar coupling. If a non-zero value is given, the distance is considered to be fixed at the given value. If a *dij* value is zero, its value is computed from the structure, and it is assumed to be a variable distance. For one-bond couplings, it is usually best to treat the bond distance as "fixed" to an effective zero-point vibration value [79].

DCUT Controls printing of calculated and observed dipolar couplings. Only values where abs(dobs(u,l) - dexp) is greater than dcut will be printed. Default is 0.1 Hz. Set to a negative value to print all dipolar restraint information.

FREEZEMOL If this is set to *.true.*, the molecular coordinates are not allowed to vary during dynamics or minimization: only the elements of the alignment tensor will change. This is useful to fit just an alignment tensor to a given structure. Default is *.false.*.

5.13. Free energies using thermodynamic integration.

Sander has the capability of doing simple thermodynamic free energy calculations, using either PME or generalized Born potentials.

ICFE When this is set to 1, information useful for doing thermodynamic integration estimates of free energy changes will be computed. For the default value of zero, nothing is done, and you use the usual prmtop file created with *saveAmberParm*. If this option is set to 1, you must read in a "perturbation" prmtop file, created with the LEaP command *saveAmberParmPert*. Then a mixing parameter λ is used (see Eqs. 4 and 5, below) to interpolate between the "unperturbed" and "perturbed" potential functions.

CLAMBDA The value of λ for this run, as in Eqs. (4) and (5), below. Zero corresponds to the unperturbed Hamiltonian in the prmtop file; λ=1 corresponds to the perturbed Hamiltonian defined in LEaP. (Note that this is the opposite to the convention used in *gibbs*, but agrees with standard conventions in the literature.) Default is 0; that is, the system is run with the unperturbed Hamiltonian.

KLAMBDA The exponent in Eq. (5), below.

Unlike *gibbs*, the program itself does not compute free energies; it is up to the user to combine the output of several runs (at different values of λ) and to numerically estimate the integral:

$$\Delta A \equiv A(\lambda = 1) - A(\lambda = 0) = \int_0^1 \langle \partial V/\partial \lambda \rangle_\lambda \, d\lambda \tag{1}$$

If you understand how free energies work, this should not be at all difficult. However, since the actual values of λ that are needed, and the exact method of numerical integration, depend upon the problem and upon the precision desired, we have not tried to pre-code these into the program.

The simplest numerical integration is to evaluate the integrand at the midpoint:

$$\Delta A \approx \langle \partial V/\partial \lambda \rangle_{\frac{1}{2}} \tag{2}$$

This might be a good first thing to do to get some picture of what is going on, but is only expected to be accurate for very smooth or small changes, such as changing just the charges on some atoms. Gaussian quadrature formulas of higher order are generally more useful:

$$\Delta A \approx \sum_{i=1}^{n} w_i \langle \partial V/\partial \lambda \rangle_{\lambda_i} \tag{3}$$

Some weights and quadrature points are given in the accompanying table; other formulas are possible [80], but the Gaussian ones listed there are probably the most useful. The formulas are always symmetrical about λ = 0.5, so that λ_i^a and λ_i^b both have the same weight. For example, if you wanted to use 5-point quadrature, you would need to run five *sander* jobs, setting λ to 0.04691, 0.23076, 0.5, 0.76923, and 0.95308 in turn. (Each value of λ should have an equilibration period as well as a sampling period; this can be achieved by setting the *ntave* parameter.) You would then multiply the values of $< \partial V/\partial \lambda >$ by the weights listed in the Table, and compute the sum.

When *icfe=1* and *klambda* has its default value of 1, the simulation uses the mixed potential function:

$$V(\lambda) = (1 - \lambda)V_0 + \lambda V_1 \tag{4}$$

Abcissas and weights for Gaussian integration			
n	λ_i^a	λ_i^b	w_i
1	0.50000		1.00000
2	0.21132	0.78867	0.50000
3	0.11270	0.88729	0.27777
	0.50000		0.44444
5	0.04691	0.95308	0.11846
	0.23076	0.76923	0.23931
	0.50000		0.28444
7	0.02544	0.97455	0.06474
	0.12923	0.87076	0.13985
	0.29707	0.70292	0.19091
	0.50000		0.20897
9	0.01592	0.98408	0.04064
	0.08198	0.91802	0.09032
	0.19331	0.80669	0.13031
	0.33787	0.66213	0.15617
	0.50000		0.16512
12	0.00922	0.99078	0.02359
	0.04794	0.95206	0.05347
	0.11505	0.88495	0.08004
	0.20634	0.79366	0.10158
	0.31608	0.68392	0.11675
	0.43738	0.56262	0.12457

where V_0 is the potential with the original Hamiltonian, and V_1 is the potential with the perturbed Hamiltonian. The program also computes and prints $\partial V/\partial\lambda$ and its averages; note that in this case, $\partial V/\partial\lambda = V_1 - V_0$. This is referred to as linear mixing, and is often what you want unless you are making atoms appear or disappear. If some of the perturbed atoms are "dummy" atoms (with no van der Waals terms, so that you are making these atoms "disappear" in the perturbed state), the integrand in Eq. (1) diverges at $\lambda = 1$; this is a mild enough divergence that the overall integral remains finite, but it still requires special numerical integration techniques to obtain a good estimate of the integral [81]. *Sander* implements one simple way of handling this problem: if you set *klambda* > 1, the mixing rules are:

$$V(\lambda) = (1-\lambda)^k V_0 + [1-(1-\lambda)^k]V_1 \tag{5}$$

where k is given by *klambda*. Note that this reduces to Eq. (4) when $k = 1$, which is the default. If *klambda* ≥ 4, the integrand remains finite as $\lambda \to 1$ [81]. We have found that setting *klambda* = 6 with disappearing groups as large as tryptophan works well. Note that the behavior of $\partial V/\partial\lambda$ as a function of λ is not monotonic when *klambda* > 1. You may need a fairly fine quadrature to get converged results for the integral, and you may want to sample more carefully in regions where

$\partial V/\partial\lambda$ is changing rapidly.

Notes:

(1) This capability in *sander* is implemented by calling the force() routine twice on each step, once for λ=0 and once for λ=1. This increases the cost of the simulation, but involves extremely simple coding.

(2) Eq. (5) is designed for having dummy atoms in the perturbed Hamiltonian, and "real" atoms in the regular Hamiltonian. You must ensure that this is the case when you set up the system in LEaP. There is currently no facility to have dummy atoms in both V_0 and in V_1.

(3) In the GB model there is no provision for mutating the Born radii or the GB screening parameters. Hence, the principal application for GB simulations will probably be to charging free energies, where just the atomic charges are varying. One prime example would be for pK_a calculations, where the charges are mutated from the protonated to the deprotonated form. Since H atoms bonded to oxygen already have zero van der Waals radii (in the Amber force fields and in TIP3P water), once their charge is removed (in the deprotonated form) they are really then like dummy atoms. [Note that, for this special situation, there is no need to use *klambda* > 1: since the van der Waals terms are missing from both the perturbed and unperturbed states, the proton's position can never lead to the large contributions to $< V_1 - V_0 >$ that can occur when one is changing from a zero van der Waals term to a finite one.]

5.14. Targeted MD

The targeted MD option adds an additional term to the energy function based on the mass-weighted root mean square deviation of a set of atoms in the current structure compared to a reference structure. The reference structure is specified using the *-ref* flag in the same manner as is used for Cartesian coordinate restraints (NTR=1). Targeted MD can be used with or without positional restraints. If positional restraints are not applied (ntr=0), *sander* performs a best-fit of the reference structure to the simulation structure based on selection in *tgtfitmask* and calculates the RMSD for the atoms selected by *tgtrmsmask*. The two masks can be identical or different. This way, fitting to one part of the structure but calculating the RMSD (and thus restraint force) for another part of the structure is possible. If targeted MD is used in conjunction with positional restraints (ntr=1), only *tgtrmsmask* should be given in the control input because the molecule is 'fitted' implicitly by applying positional restraints to atoms specified in *restraintmask*.
The energy term has the form:

```
E = 0.5 * TGTMDFRC * NATTGTRMS * (RMSD-TGTRMSD)**2
```

The energy will be added to the RESTRAINT term. Note that the energy is weighted by the number of atoms that were specified in the *tgtrmsmask* (NATTGTRMS). The RMSD is the root mean square deviation and is mass weighted. The force constant is defined using the *tgtmdfrc* variable (see below). This option can be used with molecular dynamics or minimization. When targeted MD is used, *sander* will print the current values for the actual and target RMSD to the energy summary in the output file.

ITGTMD

= 0 no targeted MD (default)

= 1 use targeted MD

TGTRMSD Value of the target RMSD. The default value is 0. This value can be changed during the simulation by using the weight change option (see Section Two of the *sander* manual).

TGTMDFRC This is the force constant for targeted MD. The default value is 0, which will result in no penalty for structure deviations regardless of the RMSD value. Note that this value can be negative, which would force the coordinates AWAY from the reference structure.

TGTRMSMASK Define the atoms that will be used for the rms superposition between the current structure and the reference structure. Syntax is in Chapter 11.5.

TGTFITMASK Define the atoms that will be used for the rms difference calculation (and hence the restraint force), as outlined above. Syntax is in Chapter 11.5.

One can imagine many uses for this option, but a few things should be kept in mind. Since there is currently only one reference coordinate set, there is no way to force the coordinates to any specific structure other than the reference. To move a structure toward a reference coordinate set, one might use an initial *tgtrmsd* value corresponding to the actual RMSD between the input and reference (*inpcrd* and *refc*). Then the weight change option could be used to decrease this value to 0 during the simulation. To move a structure away from the reference, one can increase *tgtrmsd* to values larger than zero. The minimum for this energy term will then be at structures with an RMSD value that matches *tgtrmsd*. Keep in mind that many different structures may have similar RMSD values to the reference, and therefore one cannot be sure that increasing *tgtrmsd* to a given value will result in a particular structure that has that RMSD value. In this case it is probably

wiser to use the final structure, rather than the initial structure, as the reference coordinate set, and decrease *tgtrmsd* during the simulation. A negative force constant *tgtmdfrc* can be used, but this can cause problems since the energy will continue to decrease as the RMSD to the reference increases.

Also keep in mind that phase space for molecular systems can be quite complex, and this method does not guarantee that a low energy path between initial and target structures will be followed. It is possible for the simulation to become unstable if the restraint energies become too large if a low-energy path between a simulated structure and the reference is not accessible.

Note also that the input and reference coordinates are expected to match the *prmtop* file and have atoms in the same sequence. No provision is made for symmetry; rotation of a methyl group by 120° would result in a non-zero RMSD value.

5.15. Potentials of mean force using umbrella sampling.

Another free energy quantity that is accessible within *sander* is the ability to compute potentials of mean force (at least for simple distance, angle, or torsion variables) using umbrella sampling. The basic idea is as follows. You add an artificial restraint to the system to bias it to sample some coordinate in a certain range of values, and you keep track of the distribution of values of this coordinate during the simulation. Then, you repeatedly move the minimum of the biassing potential to different ranges of the coordinate of interest, and carry out more simulations. These different simulations (often called "windows") must have some overlap; that is, any particular value of the coordinate must be sampled to a significant extent in more than one window. After the fact, you can remove the effect of the biassing potential, and construct a potential of mean force, which is the free energy profile along the chosen coordinate.

The basic ideas have been presented in many places [82-86], and will not be repeated here. The implementation in *sander* follows two main steps. First, restraints are set up (using the distance and angle restraint input of section FOUR) and the DUMPFREQ parameter is used to create "history" files that contain sampled values of the restraint coordinate. Second, a collection of these history files is analyzed (using the so-called "weighted histogram" or WHAM method [84-86]) to generate the potentials of mean force. As with thermodynamic integration, the *sander* program itself does not compute these free energies; it is up to the user to combine the output of several windows into a final result. For many problems, the programs prepared by Alan Grossfield (`http://dasher.wustl.edu/alan/`) are very convenient, and the *sander* output files are compatible with these codes.

A simple example. The input below shows how one window of a potential of mean force might be carried out. The coordinate of interest here is the chi angle of a base in an RNA duplex. Here is the *mdin* file:

```
test of umbrella sampling of a chi torsion angle
 &cntrl
   nstlim=50000, cut=20.0, igb=1, saltcon=0.1,
   ntpr=1, ntwr=100000, ntt=3, gamma_ln=0.2,
   ntx=5, irest=1,
   ntc=2, ntf=2, tol=0.000001,
   dt=0.001, ntb=0,
   nmropt=1,
 /
 &wt type='DUMPFREQ', istep1=10  /
 &wt type='END'   /
DISANG=chi.RST
DUMPAVE=chi_vs_t.170
```

The items in the *&cntrl* namelist are pretty standard, and not important here, except for specifying *nmropt=1*, which allows restraints to be defined. (The name of this variable is an historical artifact: distance and angle restraints were originally introduced to allow NMR-related structure calculations to be carried out. But they are also very useful for cases, like this one, that have nothing to do with NMR.) The DUMPFREQ command is used to request a separate file be created to hold values of the torsion angle; this will have the name *chi_vs_t.170* given in the DUMPAVE file redirection command.

The torsion angle restraint itself is given in the *chi.RST* file:

```
#  torsion restraint for chi of residue 2
 &rst  iat=39,40,42,43, r1=0., r2=170., r3=170., r4=360., rk2 = 30.,
       rk3 = 30.,   /
```

The *iat* variable gives the atom numbers of the four atoms that define the torsion of interest. We set *r2 = r3* and *rk2 = rk3* to obtain a harmonic biassing potential, with a minimum at 170^{o}. The values *r1* and *r4* should be far away from 170, so that the potential is essentially harmonic everywhere. (It is not required that biassing potentials be harmonic, but Dr. Grossfield's programs assume that they are, so we enforce that here.) Subsequent runs would change the minimum in the potential to values other than 170, creating other *chi_vs_t* files. These files would then be used to create potentials of mean force.

5.16. Poisson-Boltzmann dynamics.

The Poisson-Boltzmann (PB) method can be used for reaction field correction for water cap simulations (see section 5.6.12). It can be turned on by setting *igb* = 10. The water cap should be set up in Leap using the solvateCap option (see Chapter 3). Note that PB can also be used, in principle, as a pure implicit solvent just as GB (see section 5.6.5) for molecular dynamics simulations [10]. However, this has not been implemented in this release.

The PB method implemented here is based on an efficient finite-difference solver [9]. What is implemented in Amber 8 is a special case of the procedure described by Lu and Luo [10]. Specifically, the electrostatic energies and forces are determined as in Eqs. (20) and (21) of this paper, but the dielectric surface is fixed at the boundary of the CAP waters. That is, in regions of space that are less than cap radius from the cap center (both of these are set with the "solvateCap" command in LEaP), the dielectric is taken to be "epsin" (typically 1.0); otherwise, the dielectric is "epsout" (typically 80.)

This means that all electrostatic interactions are computed, and that the electrostatic cutoffs (*cutres* and *cutfd*, below) are just used to partition the electrostatic interactions into "short-range" and "long-range" contributions. (This is analogous to the way the "cut" variable is used in PME.) Covalent interactions are computed in the usual way, and the Lennard-Jones interactions are computed out to a distance *cutnb*, with no long-range correction for the missing terms.

It should be pointed out that "solvateCap" can be used to solvate either a small portion of a solute or all of a solute, depending on the center and radius of the water CAP. The two scenarios require very different implementations for efficiency even if the fundamental algorithm is the same. The implementation in this release is for the situation where a solute is solvated completely by the water cap. If the water cap option is detected in the prmtop file, i.e. *ifcap* > 0, the PB will ignore whatever atoms outside the water cap for its dielectric setup.

Because PB treats regions outside the water cap (augmented by a buffer) as continuum, the explicit water molecules should stay inside the water cap throughout a simulation. Thus a strong restraining harmonic potential should be used, the recommended value for *fcap* is 10 kcal/mol-Å^2 Note that the restraining force is only turned on when a water molecule moves outside the water cap, so that its interference to the solute dynamics is small. Incidentally, this is also why the water cap is augmented by a buffer in the definition of low dielectric region.

The accuracy of PB is related to the finite-difference grid spacing, the convergence criterion for the PB solver, and the cutoff distance used for correction of finite difference Coulombic interactions. See the descriptions below for more information. PB calculations are memory intensive for macromolecules. Thus, the efficiency of PB depends on how much memory is allocated for it. The option directly related its memory allocation is the finite-difference grid spacing. See the descriptions below.

More background on the Poisson-Boltzmann model is given in Chap. 12.

All PB options described below can be defined in the `&pb` namelist, which is read immediately after the `&cntrl` namelist. We have tried hard to make the defaults for these parameters appropriate for solvated simulations. Please take care in changing any values. Note that it is not necessary to use the `&pb` namelist at all to turn on PB as long as *igb* = 10. Of course, this means that you only want to use default options for PB. The `&pb` namelist has the following variables:

EPSIN Sets the dielectric constant of the solute region, i.e. the region within the water cap; default is 1.0.

EPSOUT Sets the implicit solvent dielectric constant, i.e. the region outside the water cap; default is 80.

ISTRNG	Sets the ionic strength (in mM) for the Poisson-Boltzmann solvent; default is 0 mM.
PBTEMP	Temperature used for the PB equation, needed to compute the Boltzmann factor for salt effects; default is 300 K.
SPACE	Sets the grid spacing for the finite difference solver; default is 0.7 Å.
NBUFFER	Sets how far (in grid units) the boundary of the finite difference grid is away from the water cap; default is 9 grids.
ACCEPT	Sets the convergence criterion (relative) for the finite difference solver; default is 0.001.
MAXITN	Sets the maximum number of iterations for the finite difference solver, default to 100. If maxitn is reached during a simulation (with an accept value of 0.001), it usually indicates there is something wrong in the installation of the program.
NPBGRID	Sets how often the finite difference grid is regenerated; default is 1000 steps. Note that the PB solver effectively takes advantage of the fact that the electrostatic potential distribution varies very slowly during dynamics simulations. This requires that the FDPB grid be fixed in space for the code to be efficient. However, molecules do move freely in simulations. Thus, it is necessary to set up the FDPB grid once in a while to make sure a molecule is well within the grid.
NSNBR	Sets how often residue-based pairlist is generated; default is 25 steps.
NSNBA	Sets how often atom-based pairlist is generated; default is 5 steps.
CUTRES	Residue-based cutoff distance; default is 12 Å. The residue-based nonbonded list is used to make the generation of the atom-based cutoff list efficient.
CUTFD	Atom-based cutoff distance to remove short-range finite-difference Coulombic interactions; default is 14 grid units.
CUTNB	Atom-based cutoff distance for van der Waals interactions; default is 10 Å.
NPBVERB	When set to 1, turns on verbose mode for PB calculations; default is 0.

5.17. QM/MM calculations

Amber now has the option of allowing part of the system to be described by a semiempirical Hamiltonian, and the remainder by a molecular mechanics force field; this is referred to as a "coupled potential" calculation. Several additional variables in the `&cntrl` namelist control this option; additional information is provided in formatted input following the namelist group, and in separate files. This QM/MM capability is only in the *sander.QMMM* program; it is limited now to single-cpu runs, and does not support periodic boundary conditions.

A discussion of the basic ideas behind these coupled potential calculations is given in Chap. 12.

5.17.1. Coupled Potential Namelist Variables

The following variables may be present in the `&cntrl` namelist:

IFQT — Flag for coupled potential run; if set, you must also define NQT and MODCHG. When you set this variable, several additional formatted input lines must also be included after the namelist input along with several separate files within the directory where the job is running. These files and formatted lines are described below.

- = 0 normal run; no QM atoms (default)
- = 1 coupled potential run

NQT — Number of quantum mechanical atoms. No default; must be specified if IFQT is set to one.

IDC — Flag for divide and conquer calculation. This should be consistent with keywords defined in "divcon.in".

- = 0 standard calculation (default).
- = 1 divide and conquer calculation.

MODCHG — Number of MM atoms that are going to have their charges modified. Typically MM atoms around the link atoms in a QM/MM simulation have the default value of the charge changed. This variable defines how many such atoms are included in the calculation as these will be read in during the formatted input section. Default = 0.

IDOCHG — Couples MM charges to lambda so that their influence can be removed during a simulation. Normally used as the first step in an "electrostatically decoupled" free energy perturbation calculation.

- = 0 normal run no coupling (default)
- = 1 option activated charges coupled to lambda

IDOVDW — Couples QM vdW parameters to lambda so that their contribution to the energy of a system can be calculated. Normally used as the second step in a two step "electrostatically decoupled" free energy of "solvation" type calculation.

- = 0 normal run no coupling (default)
- =1 option activated charges coupled to lambda

IDOPMF Flag for PMF run. If IDOPMF has been set to 1 then the definition of the PMF system is read in during the formatted reads (described below). The atoms defined are the ones moved slowly apart over the run. They are moved in the direction specifically defined in this same section of formatted input.

= 0 normal run no PMF (default)

=1 option activated (program will look for PMF input)

CHGLAM Initial charge scaling factor used with IDOCHG. This value is changed at each step by DCHG. Default = 1.0

DCHG Value by which CHGLAM is changed at each step during the FEP simulation. Default = 0.0

VLAMBI Initial value of lambda for slow growth runs (only available for charge and vdW scaling and not available for PMFs). Default = 1.0

VLAMBF Final value of lambda for slow growth runs (only available for charge and vdW scaling and not available for PMFs). Default = 0.0

NPERT The number of groups of atoms that will be moved in the PMF. Must be specified if a PMF run is being done.

NEQUIL Number of equilibration steps. Must be specified for any coupled potential run. NOTE: The variable NSTLIM should be set larger than NEQUIL + NSAMPL. Also important is that coupled potential jobs should normally be done within one "run" (NRUN=1). While this is not an absolute requirement it is suggested.

NSAMPL Number of sampling steps. Must be specified for any run where IDOVDW, IDOCHG, or IDOPMF is equal to one. Default = 0.

5.17.2. Coupled Potential Formatted Input

Following the *&cntrl* namelist comes some additional input, in two sections. See the files in *amber8/test/qmmm* for some examples.

5.17.2.1. Lines for PMF runs

The next five lines are repeated NPERT times to define each PMF group *only if* a PMF run is being done. If this is not the case, then these lines should be omitted.

IABOND If the PMF is for a bond, then IABOND should be the atom in group i that participates in the bond. If the PMF is for the movement of the center of mass, then this should be zero.

SXCM, SYCM, SZCM

These are the X,Y,and Z components of starting center of mass for group i, or, if IABOND(i) is not 0, the starting coordinates of the atom IABOND(i).

PX,PY,PZ The X,Y, and Z components of step for moving group i. It is assumed that this corresponds to separating the groups.

IATMS(1),IATMS(2),*etc.*

The next call reads in the list of atoms in group i. The list may take up as many lines as necessary and ranges of atoms may be indicated with a slash, e.g., 1/5 means 1,2,3,4,5. Terminate list with a blank line.

Blank specifier The above 4 lines must be followed by a "blank" line. Due to unfortunate machine dependence problems, the line must actually say "blank" instead of just having nothing on it. The word blank must appear just as it does here "blank" with no capital letters and appearing in columns 1-5.

5.17.2.2. Lines for all coupled potential runs

QM atoms are defined in the next series of lines along with the identities of MM atoms which are to have their charges modified.

LABELS(1), LABELS(2), *etc.*
The LABELS are the QM atom numbers

ICHG, TMPCHG (This line is repeated MODCHG times.) ICHG is the atom number for which charge is to be changed, and TMPCHG is the new charge. NOTE: Any belly atom input follows directly after the last line of input (CP-2) in the normal AMBER group format.

5.17.3. Control Files for Coupled Potential Simulations

divcon.in To run a QM/MM job it is necessary to include a file named "divcon.in" within the directory in which the job is running. This file contains DIVCON specific commands which would normally be included on the DIVCON command line. Please refer to DIVCON manual (in *amber8/doc/DivConLite.pdf*) for keywords.

cntrl.dat It is possible to calculate the ESP charges for the QM atoms within the field of charges of the MM atoms. The only input necessary to cause the calculation of ESP charges is the inclusion of the file "cntrl.dat" within the directory which the job is running. The calculation of ESP charges is fairly computationally intensive depending on the size of the system and speed of the workstation used. For a large QM system with many MM atoms on an average workstation, several hours of computational time might be expected. The file "cntrl.dat" needs only to include two integers on the first line of the file (free format). These integers correspond to:

INESP First MD step on which to calculate ESP charges. Required

ISTESP Number of MD steps between ESP charge calculation. Required

move1.dat The presence of this file acts as little more than a flag as it needs not contain any data. If the file "move1.dat" is present in the directory in which the job is being run, then the internal coordinate constraints will be enforced by moving only the first atom in each internal coordinate list. This is in contrast to the normal algorithm which strives to achieve a constraint value by moving both ends of the internal coordinate equally.

cutoff.dat It is possible to define a secondary cutoff for QM atoms within a coupled potential run. As discussed in the theory section of the manual, the energy of the QM potion of the system is highly dependent on the cutoff used. The energy can also be greatly effected by residues moving in and out of a residue based cutoff (particularly charged residues). It is therefore often advisable to use a longer cutoff for the QM residues to minimize this effect while not greatly increasing the cost of the calculation. To specify a secondary cutoff the file "cutoff.dat" should be included in the directory in which the job is being run. This file has free format

lines for each residue for which there is to be a secondary cutoff. Each line should contain the residue number and the new cutoff (*i.e.*, ICUTI,RCUTI).

ICUTI the residue number for which there is to be an increased cutoff

RCUTI the new cutoff for that residue

constraint.dat Geometry constraints can be imposed on the QM portion of the system. This is done by reading in geometric constraint data for the QM portion of the system from the file 'constraint.dat'. If the file is not present, then no QM constraints are assumed. The routine returns with ierror=0 if constraint data is in the proper format. Otherwise ierror=1. A check is also made for the file "move1.dat". If this file is present (it need not contain any data) Then the internal coordinate constraints will be enforced by moving only the first atom in each internal coordinate list.

Each constraint takes up one line in 'constraint.dat'. For the I'th constraint, the I'th line in the file should have one of the three following (free-field) formats:

```
(IABCD(J,I),J=1,4), ICODE(I)  (ICODE(I)=0)
   or
(IABCD(J,I),J=1,4), ICODE(I), QCNSTR(I)  (ICODE(I)=1)
   or
(IABCD(J,I),J=1,4), ICODE(I), (IABCD0(J,I),J=1,4) (ICODE(I)>1)
```

Here, an internal coordinate is defined by the atoms IABCD(1,I), IABCD(2,I), IABCD(3,I), AND IABCD(4,I):

```
IABCD(1,I)-IABCD(2,I) = bond length specifier
IABCD(1,I)-IABCD(2,I)-IABCD(3,I) = bond angle specifier
IABCD(1,I)-IABCD(2,I)-IABCD(3,I)-IABCD(4,I) = Dihedral angle specifier
```

If the internal coordinate is a bond length or bond angle, then zeroes should be entered for IABCD(3,I) and/or IABCD(4,I). The value of ICODE(I) determines whether the first, second, or third format is used. If ICODE(I)=0 then the internal coordinate is constrained to be equal to its initial value for the duration of the simulation. If ICODE(I)=1, then the coordinate is constrained to be equal to CONSTR(I). If ICODE(I) is greater than one, the coordinate is constrained according to a reference coordinate defined by atoms (IABCD0(J,I),I=1,4). Here again, the manner in which the constraint is defined depends on the value of ICODE(I):

```
ICODE(I) = 2 ---> IABCD COORDINATE = IABCD0 COORDINATE
ICODE(I) = 3 ---> IABCD DIHEDRAL = NEGATIVE OF IABCD0 DIHEDRAL
ICODE(I) = 4 ---> IABCD DIHEDRAL = IABCD0 DIHEDRAL + 180 DEGREES
ICODE(I) = 5 ---> IABCD DIHEDRAL = IABCD0 DIHEDRAL + 120 DEGREES
ICODE(I) = 6 ---> IABCD DIHEDRAL = IABCD0 DIHEDRAL - 120 DEGREES
```

NOTE: that the atom numbering should correspond to the AMBER convention and bond lengths are specified in angstroms, while bond angles and dihedrals are in degrees.

5.18. Replica-Exchange.

In the one dimensional replica-exchange method noninteracting copies of the system, replicas, are simulated concurrently at different values of some independent variable, such as temperature. Replicas are subjected to Monte Carlo move evaluation periodically, thus, effecting exchange between values of the independent variable. The replica-exchange method enables simulation in a generalized ensemble --- one in which states may be weighted by non-Boltzmann probabilities. (However, one advantage of replica-exchange is the simplicity inherent in its use of Boltzmann factors.) Consequently, local potential energy wells may not dominate traversal through phase space because a replica trapped in a local minimum can escape via exchange to a different value of the independent variable [87]. *Sander* implements two ways of carrying out replica-exchange simulations. One (called *multisander REM*) has the different replicas as parts of an overall MPI job. In the second, the MMTSB Toolkit is used as a "driver" program, which spawns off independent *sander* jobs, and takes care of collecting the results and making the exchanges. These two options are outlined in the next sections.

5.18.1. Multisander REM

Multiple *sander* jobs can run concurrently under a single MPI program. This can be used to just run unconnected parallel jobs, but it is more useful to use this as a platform for the *replica exchange method.*

The replica exchange method (REM) in temperature space for molecular dynamics [87] has been implemented on top of the framework that multisander provides. N non-interacting replicas are simultaneously simulated in N separate MPI groups, each of which has its own set of input and output files. One process from each MPI group is chosen to form another MPI group (called the master group), in which exchanges are attempted.

In this REM implementation, the sander subroutine is repeatedly called by each replica. After each call, instantaneous temperature, instantanous potential energy, and target temperatures of all replicas are collected by the master group, which carries out the exchange part of the calculation. The N replicas are first automatically sorted in an array by their target temperatures. Half of the N replicas (replicas with even array indices) are chosen to be exchange initiators. These initiators pair with their right and left neighbors alternatively after each sander call. Topologically, the N temperature-sorted replicas form a loop, in which the first and the last replicas are neighbors. Therefore, N/2 exchanges are attempted in each iteration. If the exchange is allowed between the pair, the target temperatures for the two replicas are swapped before the next sander call. The velocities of each replica involved in successful exchange are then adjusted by a scaling factor related to the previous and new target temperatures.

Before starting a replica exchange simulation, an optimal set of target temperatures should be determined so that the exchange ratio is roughly a constant. These target temperatures determine the probability of exchange among the replicas, and the user is referred to the literature for a more complete description of the influence of various factors on the exchange probability.

By default, REM is not compiled in sander (although multisander is supported for running multiple unconnected parallel jobs). REM can be enabled in the executable by recompiling the sander executable (in $AMBERHOME/src/sander) and turning on the -DREM flag via: make clean; make AMBERBUILDFLAGS=’-DREM’ parallel. See the "Installing Non-Standard Features" section in Chapter 1 for further information.

Each replica requires (for input files) or generates (for output files) its own mdin, inpcrd, mdout, mdcrd, restrt, mdinfo, and associated files. By default, a common base file name is used among all replicas that is appended with a 3-column number separated from the basename by a

".", starting with 000, to form the actual filename used by *sander*. However, only the basename is specified in the sander command line. This can be overridden through the specification of a *groupfile* on the command line with the *-groupfile groupfile* option. The *groupfile* file contains a separate command line for each of the replicas or multisander instances, one per line (with no extra lines except for comments which have a '#' in the first column). To choose the number of replicas or multisander instances, the *-ng N* command line option is used (in this case to specify *N* separate instances. If the number of processors (for the MPI run) is larger than *N* (and also a multiple of *N*), each replica or multisander instance will run on a number of processors equal to the total specified on the command line divided by *N*. Note that in the *groupfile*, the *-np* option is currently ignored, *i.e.* each replica or multisander instance is currently hardcoded to run on an equivalent number of processors.

For example, an 8-replica REM job will need 8 mdin and 8 inpcrd files. In the default case, these will be named mdin.00[0-7] and inpcrd.00[0-7], and generate 8 mdout, mdcrd, restrt, and mdinfo, named mdout.00[0-7], mdcrd.00[0-7], etc. These names correspond to sander defaults. The base named (mdin, mdout) may be changed on the sander command line as per normal usage. Alternatively, and overridding the sander command line file names, these names can be specified for each instance in the *-groupfile groupfile*. For example, the groupfile might look like this:

```
#
# multisander or replica exchange group file
#
# replica 1
-O -i mdin.rep1 -o mdout.rep1 -c inpcrd.rep1 -r restrt.rep1 -x mdcrd.rep1
# replica 2
-O -i mdin.rep2 -o mdout.rep2 -c inpcrd.rep2 -r restrt.rep2 -x mdcrd.rep2
# replica 3
-O -i mdin.rep3 -o mdout.rep3 -c inpcrd.rep3 -r restrt.rep3 -x mdcrd.rep3
# replica 4
-O -i mdin.rep4 -o mdout.rep4 -c inpcrd.rep4 -r restrt.rep4 -x mdcrd.rep4
# replica 5
-O -i mdin.rep5 -o mdout.rep5 -c inpcrd.rep5 -r restrt.rep5 -x mdcrd.rep5
# replica 6
-O -i mdin.rep6 -o mdout.rep6 -c inpcrd.rep6 -r restrt.rep6 -x mdcrd.rep6
# replica 7
-O -i mdin.rep7 -o mdout.rep7 -c inpcrd.rep7 -r restrt.rep7 -x mdcrd.rep7
# replica 8
-O -i mdin.rep8 -o mdout.rep8 -c inpcrd.rep8 -r restrt.rep8 -x mdcrd.rep8
```

Note that the mdin and inpcrd files are *not* required to be ordered by their target temperatures since the temperatures of the replicas will not remain sorted during the simulation. Sorting is performed automatically at each REM iteration as described above. Thus one can restart REM simulations without modifying the restart files from the previous REM run (see below for more information about restarting REM).

It is important to ensure that the target temperature (specified using *temp0*) is the only difference among the mdin files for the replicas, otherwise the outcome of an REM MD simulation may be unpredictable since each replica may be performing a different type of simulation.

However, in order to accommodate advanced users, the input files are not explicitly compared.

Addition of the -DREM replica exchange code leads to some changes in the default behavior and output files since REM calls the sander routine multiple times during a single REM MD simulation, with output files combined from *each* REM iteration. Thus, the multiple iterations of a single REM MD simulation will generate a single set of output files. The trajectory outputs (mdcrd) of each sander call (each REM iteration) are combined by enabling file appending.

5.18.1.1. Restarting REM simulations

It is recommended that each REM MD run generate a new set of output files (such as mdcrd), but for convenience one may use -A in the command line in order to append output to existing output files. This can be a useful option when restarting REM simulations. As noted above, file appending is always used after the first iteration (first exchange attempt) in REM. The use of -A on the command line only affects how sander treats any existing files during the first iteration of REM. If -A is used, files that were present before starting the REM simulation are appended to throughout the new simulation. Note that this can seriously affect performance on systems where the file writing becomes rate limiting. If -O is used, any files present are overwritten during the first iteration, and then subsequent iterations append to these new files.

At the end of a REM simulation, the target temperatures of each replica are most likely not the same as they were at the start of the simulation (due to exchanges). If one wishes to continue this simulation, sander will need to know that the target temperatures have changed. Since the target temperature is normally specified in the mdin file (using temp0), the previous mdin files would all need to be modified to reflect changes in target temperature of each replica. In order to simplify this process, the program will write the current target temperature as additional information in the restart files during an REM simulation. When an REM simulation is started, the program will check to see if the target temperature is present in the restart file. If it is present, this value will override the target temperature specified using temp0 in the mdin file. In this manner, one can restart the simulation from the set of restart files and the program will automatically update the target temperature of each replica to correspond to the final target temperature from the previous run. If the target temperature is not present (as would be the case for the first REM run), the correct values should be present in the mdin files.

5.18.1.2. Content of the output files

Standard (non-REM) sander simulations produce a significant amount of data at the start and end of the simulation. Since sander is called after each REMD iteration (possibly each 100-1000 MD steps), the output files will become quite large. In order to maintain an energy archive while saving disk space, the roles of mdout and mdinfo are therefore switched during REM simulation. The mdout file is created and overwritten for each sander call, and only contains information pertinent to this call. The energy archive for the entire set of REM iterations is placed into the mdinfo file. In non-REM simulations, mdinfo only contains the most recent energy information.

An important user-controllable option exists for writing output files (repcrd `&cntrl` namelist variable, see below). For mdinfo and mdcrd, two methods of file writing are permitted. An individual mdcrd or mdinfo file (for example, mdcrd.001) can contain the history of a single replica (which samples multiple temperatures), or it can contain all of the information for a single target temperature (regardless of the index of the replica that employed that target temperature. In the former case (files containing data for a single replica index), the files contain a continuous trajectory in phase space but discontinuous in target temperature. Thermodynamic analysis will most likely require postprocessing of these files such that data for each temperature can be analyzed

separately. The information required for this postprocessing (the target temperature of each replica during the simulation) can be found in the replica log file. One should note, however, that the current sander trajectory format does not store any information in the trajectory files other than coordinates (such as time index or target temperature). Thus it can be difficult (or impossible) to postprocess the data if any file corruption has occurred.

For this reason, the default behavior is to write mdcrd and mdinfo files that represent a particular target temperature rather than replica index. Thus no postprocessing is required for thermodynamic analysis, but the trajectories do not represent a path that was physically sampled. If desired, a continuous trajectory can be obtained by the postprocessing described above.

As mentioned earlier, mdin and inpcrd files need not be ordered by target temperature. If the user requests that mdcrd, mdinfo and mdout files correspond to a single temperature, these files will be ordered by target temperature. For example, mdcrd.000 would correspond to the coordinate archive for the lowest temperature, mdcrd.001 would be the next highest temperature, and so on. Note that this reordering only occurs if temperature-based output files are requested. If replica-based output files are requested, each output file will have the same index value as the corresponding input files for that replica.

5.18.1.3. Major Changes from sander

Within an MPI job, as discussed above, it is now possible to run multiple sander jobs at once, such that each job gets a subset of the total processors. To run multisander (or to specify the number of replicas to use in a REM run), a new command line argument:

-ng specifies the number of sander runs to perform concurrently. Note that at present, the number of sander runs to perform must be a divisor of the total number of processors (specified by the MPI run command).

In the -DREM compiled code enabling replica-exchange, the following additional options are available and changes in behavior from standard sander are present. First, there are two new command line arguments:

-rem specifies the type of replica exchange simulation. Two options are currently available. 0, no replica exchange (standard MD) (default behavior if -rem is not specified on command line); 1, regular replica exchange (requires -ng, see the multisander section of this manual).

-remlog specifies the filename of a log file. This file records from left to right, for every replica and every exchange attempt, the velocity scaling factor (negative if the exchange attempt failed), current actual temperature, current potential energy, current target temperature, and the new target temperature. The default value is rem.log.

Next, there are three new variables in the `&cntrl` namelist:

REPCRD REM output file (mdinfo and mdcrd) specifications. If 0, output files contain data for a particular temperature (default); if 1, output files contain data for a particular replica.

NUMEXCHG The number of exchange attempts, default 0.

NSTLIM the number of MD steps between exchange attempts. The total length of the REM simulation will therefore be nstlim*numexchg time steps.

5.18.1.4. Example

Below is an example of an 8-replica REM MD run on 16 processors, assuming that relevant environment variables have been properly set.

```
$MPIRUN -np 16 sander -O -ng 8 -rem 1 -remlog rem.log
-i rem.in -p prmtop -c inpcrd -o rem.out -x rem.x -inf
reminfo -r rem.r
```

This input specifies that REM should be used (-rem 1), with 8 replicas (-ng 8) and 2 processors per replica (-np 16). Note that the total number of processors should always be a multiple of the number of replicas. Here is a section of the rem.log:

```
# replica exchange log file

# Replica #, Velocity Scaling Factor, T, Eptot, Temp0, NewTemp0

# exchange 1

1 1.03 256.57 -534.74 267.00 283.00
2 0.97 272.54 -533.89 283.00 267.00
3 -1.00 287.43 -507.27 300.00 300.00
4 -1.00 310.18 -502.66 318.00 318.00
5 -1.00 329.41 -467.64 338.00 338.00
6 -1.00 347.79 -444.70 358.00 358.00
7 0.98 388.38 -445.28 380.00 403.00
8 1.02 375.30 -396.27 403.00 380.00

# exchange 2

1 -1.00 271.38 -522.06 283.00 283.00
2 -1.00 271.39 -552.22 267.00 267.00
3 -1.00 301.86 -516.23 300.00 300.00
4 0.99 323.10 -504.94 318.00 338.00
5 1.01 319.03 -462.42 338.00 318.00
6 -1.00 328.79 -448.31 358.00 358.00
7 -1.00 424.86 -433.82 403.00 403.00
8 -1.00 380.06 -433.69 380.00 380.00
```

5.18.2. Multiscale Modeling Tools in Structural Biology

The Multiscale Modeling Tools in Structural Biology (MMTSB), *http://mmtsb.scripps.edu/*, suite of software provides a generic user interface to various protein structure modeling packages [88]. Intended applications include protein structure prediction, loop modeling, structure refinement, and structure evaluation/scoring. The MMTSB Tool Set interfaces with AMBER and CHARMM for all-atom modeling and with MONSSTER for lattice-based low-resolution modeling. It provides conversion routines between high and low resolution models and implements efficient ensemble-based sampling techniques. The MMTSB Tool Set source package can be downloaded from the web. It is available free of charge for academic use; registration is required.

Sander is a viable molecular dynamics engine for the MMTSB which implements replica-exchange via a client-server architecture. The client software runs independent sander jobs that communicate replica information to the server via sockets. The server software performs exchange of replicas. In addition to AMBERHOME two environment variables must be defined. SANDEREXEC must be assigned to the full path of the *sander* executable. MMTSBDIR must be assigned to the the root of the MMTSB installation. Furthermore, $MMTSBDIR/perl and $MMTSBDIR/bin must be included in your path. aarexAmber.pl is the name of the MMTSB perl interface to *sander*. Its command line options and arguments are used to create the *mdin* control files and the *sander* command lines. The Amber test cases for MMTSB, in the directory $AMBERHOME/test/mmtsb, contain annotated examples of the usage of aarexAmber.pl.

To build sander with support for MMTSB use the -mmtsb option of configure. For example,

```
cd $AMBERHOME/src
make clean
./configure -mmtsb ifort
cd sander
make sander
mv -i sander ../../exe
```

To test the resulting sander, do the following:

```
cd $AMBERHOME/test
make test.mmtsb
```

See Chapter 1 for general information on installation and testing.

Two variables in the *&cntrl* namelist of *sander* are directly related to MMTSB. Both temperature and lambda replica-exchange are available. The velocities are randomized whenever either a temperature or a lambda exchange occurs.

MMTSB_SWITCH

The control of MMTSB replica-exchange.

- = 0 Do not perform MMTSB replica-exchange. This is the default value.
- = 1 Perform MMTSB temperature replica-exchange.
- = 2 Perform MMTSB lambda replica-exchange in conjunction with thermodynamic free energy calculations.

MMTSB_ITERATIONS

The number of molecular dynamics iterations between possible replica exchanges. The default value is 100.

5.19. Constant pH calculations

The constant pH molecular dynamics method has been implemented in *sander* by John Mongan. A paper by Mongan, Case and McCammon describing the theory behind this work is in preparation. Constant pH is limited to implicit solvent simulations. Using the constant pH method requires minor modifications to the process of generating the prmtop file, as well as generation of a second input file from the prmtop file, describing the titrating residues.

5.19.1. Background

Traditionally, molecular dynamics simulations have employed constant protonation states for titratable residues. This approach has many drawbacks. First, assigning protonation states requires knowledge of pK_a values for the protein's titratable groups. Second, if any of these pK_a values are near the solvent pH there may be no single protonation state that adequately represents the ensemble of protonation states appropriate at that pH. Finally, since protonation states are constant, this approach decouples the dynamic dependence of pK_a and protonation state on conformation.

The constant pH method implemented in *sander* addresses these issues through Monte Carlo sampling of the Boltzmann distribution of protonation states concurrent with the molecular dynamics simulation. The nature of the distribution is affected by solvent pH, which is set as an external parameter. Residue protonation states are changed by changing the partial charges on the atoms.

5.19.2. Preparing a system for constant pH

AMBER provides definitions for titrating side chains of ASP, GLU, HIS, LYS and TYR. See below if you need other titrating groups.

Begin by preparing your PDB file as you normally would for use with LEaP. Edit the PDB file, replacing all histidine residue names (HIS, HID, or HIE) with HIP. Change all ASP and ASH to AS4 and all GLU and GLH to GL4. This ensures that the prmtop file will have a hydrogen defined at every possible point of protonation.

Run leap, and enter the following commands:

```
source leaprc.ff99
loadAmberParams frcmod.mod_phipsi.1
set default PBRadii mbondi2
loadoff constph.lib
loadamberparams frcmod.constph
```

This loads constph.lib, which contains residue definitions for AS4 and GL4 (aspartate and glutamate residues with syn and anti hydrogens on each carboxyl oxygen), and frcmod.constph which defines improper torsions to keep the syn and anti protons on AS4 and GL4 from rotating into the same position. Now load your edited PDB file and proceed as usual to create the prmtop and prmcrd files.

Once you have the prmtop file, you need to generate a cpin file. The cpin file describes which residues should titrate, and defines the possible protonation states and their relative energies. A perl script, *cpinutil.pl*, is provided to generate this file. It takes a PDB file as input, either

on the command line or on STDIN, and writes the cpin file to STDOUT. Note that you *must* generate this PDB file from the prmtop file; do not use your original PDB file. Since LEaP has inserted extra hydrogens, the atom numbering in your original PDB file will not correspond to the prmtop file. Here is an example of generating the PDB file and using it to create the cpin file in a single step:

```
ambpdb -p prmtop < prmcrd | cpinutil.pl > cpin
```

The *cpinutil.pl* program accepts a number of flags that modify its behavior. By default, all residues start in protonation state 0: deprotonated for ASP and GLU, protonated for LYS and TYR, doubly protonated for HIS (i.e. HIP). Initial protonation states can be specified using the -states flag followed by a comma delimited list of initial protonation states (see below for more about protonation state definitions) as follows:

```
ambpdb -p prmtop < prmcrd | cpinutil.pl -states 1,3,0,0,0,1 > cpin
```

The -system flag can be used to provide a name for the titrating system. If experimental pKa values have been defined for the system (see below), they will be written into the cpin file. Note that experimental pKa values are used only by the analysis scripts to calculate pKa prediction error; they are not used in any way by *sander* and do not need to be included.

```
ambpdb -p prmtop < prmcrd | cpinutil.pl -system HEWL > cpin
```

A number of flags are available for filtering which residues are included in the cpin file. All residues in the cpin file, and only the residues in the cpin file, will be titrated. In general it is safe to exclude TYR and LYS for acidic simulations and GL4 and AS4 for basic simulations. HIP should be included in all except very acidic simulations. Note that there is currently no support for titrating N or C terminal residues. If you have an N or C terminal residue with a titratable sidechain, you should explicitly exclude it from the cpin file. The -resnum flag may be used to specify which residue numbers should be retained; all others are deleted. Conversely, the -notresnum flag can be used to specify which residue numbers are deleted; all others are retained. Residue number refers to the numbering in the PDB file, not the index number among titrating residues. Similarly, -resname and -notresname can be used to filter by residue type. For instance, -notresname TYR,LYS would eliminate basic residues from the cpin file. If experimental pKa values are known through use of the -system flag, the -minpka and -maxpka flags can be used to filter residues by experimental pK_a values.

cpinutil.pl can also take an existing cpin file as input, allowing modification or further filtering of existing cpin files. See *cpinutil.pl -h* for a summary of options and flags.

5.19.3. Running at constant pH

Running constant pH under *sander* has few differences from normal operation. In the mdin file, you must set *icnstph*=1 to turn on constant pH. *solvph* is used to set the solvent pH value. You must also specify the period for Monte Carlo steps, *ntcnstph* (for period *n*, a Monte Carlo step is performed every *n* steps). Note that only one residue is examined on each step, so you should decrease the step period as the number of titrating residues increases to maintain a constant effective step period for each residue. We have seen good results with fairly short periods,

in the neighborhood of 100 fs effective period for each residue (e.g. *ntcnstph*=5, *dt*=0.002 with about 10 residues titrating).

In order to avoid having to calculate non-electrostatic contributions to protonation state transition energies, this method uses correction factors based on the relative energy differences of the different protonation states in the AMBER force field. These relative energies were calculated under the following parameters:

```
cut=30.0, scee=1.2, igb=2, saltcon=0.1,
ntb=0, dt=0.002, nrespa=1,
ntt=1, tempi=300.0, temp0 = 300., tautp=2.0,
ntc=2, ntf=2, tol=0.000001,
```

Deviations from these parameters, or from the force field or GB radii specified above may affect the relative energies of the protonation states, which will cause erroneous results. If you must deviate from these settings, you can test whether your changes will cause problems by running long (multiple ns) titrations of the model compounds, with solvent pH equal to the model compound pKa value. The model compounds are ACE-X-NME, where X is AS4, GL4, HIP, LYS or TYR. If these titrations predict the model pKa value (4.0, 4.4, 6.5, 10.4 and 9.6, respectively), then the parameter set is probably OK. If not, you must either change the parameter set or recalculate the relative energies (see section below).

Some additional command line flags have been added to *sander* to support constant pH operation. The cpin file must be specified using the -cpin option. Additionally, a history of the protonation states sampled is written to the filename specified by -cpout. Finally, a constant pH restart file is written to the filename specified by -cprestrt. This is used to ensure that titrating residues retain the same protonation state across restarts. The constant pH restart file is a cpin-format file, and should be used as the cpin file when restarting the run. It will generally be longer than the original cpin file, as it contains some amount of zeroed data, due to limitations in the Fortran namelist implementation. The excess zero data can be removed by filtering it through *cpinutil.pl*, e.g.

```
cpinutil.pl cprestrt > cpin2
```

5.19.4. Analyzing constant pH simulations

As the simulation progresses, the protonation states that are sampled are written to the cpout file. A section of a cpout file is included here:

```
Solvent pH:  2.00000
Monte Carlo step size:        2
Time step:        0
Time:      0.000
Residue    0 State:  1
Residue    1 State:  0
Residue    2 State:  1
Residue    3 State:  0
Residue    4 State:  1
```

```
Residue    5 State:  0

Residue    2 State:  0

Residue    4 State:  0

Residue    0 State:  3

Residue    1 State:  0

Residue    0 State:  0
```

One record is written on each Monte Carlo step. Each record is terminated by a blank line. There are two types of records, full records (at the top of the file) and delta records (single lines, remainder of file). Full records are written before the run begins, on timesteps where restart files are written, and on the final time step (assuming these are Monte Carlo steps); delta records are written in all other cases. The full record specifies the protonation state of each residue, along with some additional information, while the delta records give only the protonation state for the residue selected on the corresponding Monte Carlo step. Note that in some cases, the protonation state for a delta record may be the same as that in an earlier record: this indicates that the Monte Carlo protonation move was rejected. The residue numbers in cpout are indices over the titrating residues included in the cpin file; cpout must be analyzed in conjunction with cpin to map these indices back to the original system.

The Perl script *calcpka.pl* is provided as an example parser for the cpout format, and as a utility for calculating predicted pK_a values from cpout files. It takes a cpin file as its first argument and any number of cpout files for its remaining arguments. For instance:

```
calcpka.pl cpin cpout1 cpout2 cpout3
```

Output contains one line for each titrating residue in the system. *Offset* is the difference between the predited pK_a and the system pH. *Pred* is the predicted pK_a. Note that predictions are calculated assuming Henderson-Hasselbalch titration curves. Predictions are most accurate when the absolute value of the offset is less than 2.0. If experimental pK_a values have been defined for the system (see following), then experimental and error values are also printed. *Frac Prot* is the fraction of time the residue spends protonated and *Transitions* gives the number of accepted protonation state transitions. Note that transitions between states with the same total protonation (e.g. *syn* and *anti* protonated states of a carboxylic acid) are not included in this total. *Average total molecular protonation* is the sum of the fractional protonations. It ranges between zero and the number of titrating residues, and gives the average protonation of the molecule as a whole.

5.19.5. Extending constant pH to additional titratable groups

There are two major components to defining a new titrating group for constant pH. First you must define the partial charges for each atom in the residue for each protonation state. Then you must set the relative energies of each state.

5.19.5.1. Defining charge sets

Partial charges are most easily calculated using Antechamber and Gaussian. You must set up a model to calculate charges for each protonation state. If the titrating group you are defining is a polymer subunit (e.g. amino acid residue), you must adjust the charges on atoms that have bonded interactions (including 1-4) with atoms in neighboring residues. The charges on these atoms must be changed so they are constant across all protonation states − otherwise relative energies of protonation states become sequence dependent. For an amino acid, this means that all backbone atoms must have constant charges. For the residues defined here, we arbitrarily selected the backbone charges of the protonated state to be used across all protonation states. The total charge difference between states should remain 1; we achieved this by adjusting the charge on the beta carbon.

5.19.5.2. Calculating relative energies

Relative energies are used to calibrate the method such that when a model compound is titrated at pH equal to its pK_a, the energies (and thus populations) of the protonated and deprotonated states are equal. Relative energies of the different protonation states are calculated using thermodynamic integration of a model compound between the charge sets defined for the different protonation states. The model compound should be a small molecule that mimics the bonded environment of the titratable group of interest, and for which experimental pK_a data are available. For instance, the model compound for an amino acid X is generally ACE-X-NME; the model compound for a ligand might be the free ligand. The thermodynamic integration calculations must be performed using exactly the same parameters and force field as you plan to use in your constant pH simulations. Once the relative energies of the states are calculated by thermodynamic integration, the energy difference must be adjusted to account for the pK_a: the energy of the more protonated states should be increased by $pK_a RT \ln(10)$.

For example suppose one were developing a model for an artificial amino acid, ART, with pK_a 3.5 and two protonation states: ARP, having one proton and ARD having zero protons. After calculating partial charges as above, you would construct a model compound having the sequence ACE-ARP-NME and generate a prtprmtop file where the ARP charges were perturbed to the ARD values. You would then use sander to perform thermodynamic integration between ARP and ARD. Suppose that this showed that the energy of ARD relative to ARP was -6.3 kcal/mol. You would assign a relative energy of -6.3 to ARD and a relative energy of 3.5RTln(10) to ARP.

5.19.5.3. Testing the titratable group definitions

Prior to large scale use of your new titratable group definition, it's a good idea to test it by performing a constant pH simulation on your model compound, with pH set to the model pK_a. Doing this requires generation of a cpin file, so this is a good point to modify the table of titratable group definitions used by *cpinutil.pl*. These tables are found near the end of CPin.pm. The table is a perl hash of 2D arrays. Each hash entry is an array of states that define a titratable group. Each state array consists of the relative energy, the relative protonation, and the partial charges for the state, in that order. An entry for the example given above might look like (charge list shortened for brevity):

```
"ARP" => [
          # State 0, ARP
```

```
 [3.5 * 1.3818, # Relative energy (300K)
  1, # Relative protonation
  -0.4157,
  0.2719,
  -0.0014,
  0.0876,
  -0.0152,
  0.0295,
 ],
 # State 1, ARD
 [-6.3, # Energy
  0, # Protonation
  -0.4157,
  0.2719,
  -0.0014,
  0.0876,
  -0.0858,
  0.019,
 ]
]
```

Below this table is another table of experimental pK_a values. Entries for new systems can be created following the example already present for HEWL (the keys are residue numbers, the values are their pK_a values). As discussed above, this is optional and does not affect the constant pH simulations − these data are used only by *calcpka.pl* and *cpinutil.pl*.

Having added your titratable group definition to the table, you should be able to prepare a cpin file as described above, run your simulation and calculate the predicted pK_a using *calcpka.pl*. Since the model compound is usually very small, runs of tens of nanoseconds are easily accessible for these tests. In general, the run to run variation of predicted pK_a values is a few hundredths of a pK_a unit for long runs with pH near pK_a. In most cases, the thermodynamic integration procedure described above yields acceptable results, but if your predicted pK_a differs significantly from the model pK_a, you may want to adjust your relative energies, regenerate your cpin file and rerun the test until you achieve good predictions.

5.20. Overview of NMR refinement using SANDER.

We find the *sander* module to be a flexible way of incorporating a variety of restraints into a optimization procedure that includes energy minimization and dynamical simulated annealing. The "standard" sorts of NMR restraints, derived from NOE and J-coupling data, can be entered in a way very similar to that of programs like DISGEO, DIANA or X-PLOR; an aliasing syntax allows for definitions of pseudo-atoms, connections with peak numbers in spectra, and the use of "ambiguous" constraints from incompletely-assigned spectra. More "advanced" features include the use of time-averaged constraints, use of multiple copies (LES) in conjunction with NMR refinement, and direct refinement against NOESY intensities, paramagnetic and diamagnetic chemical shifts, or residual dipolar couplings. In addition, a key strength of the program is its ability to carry out the refinements (usually near the final stages) using an explicit-solvent representation that incorporates force fields and simulation protocols that are known to give pretty accurate results in many cases for unconstrained simulations; this ability should improve predictions in regions of low constraint density and should help reduce the number of places where the force field and the NMR constraints are in "competition" with one another.

Since there is no generally-accepted "recipe" for obtaining solution structures from NMR data, the comments below are intended to provide a guide to some commonly-used procedures. Generally speaking, the programs that need to be run to obtain NMR structures can be divided into three parts:

(1) *front-end* modules, which interact with NMR databases that provide information about assignments, chemical shifts, coupling constants, NOESY intensities, and so on. We have tried to make the general format of the input straightforward enough so that it could be interfaced to a variety of programs. At TSRI, we generally use the FELIX and NMRView codes, but the principles should be similar for other ways of keeping track of a database of NMR spectral information. As the flow-chart on the next page indicates, there are only a few files that need to be created for NMR restraints; these are indicated by the solid rectangles. The primary distance and torsion angle files have a fairly simple format that is largely compatible with the DIANA programs; if one wishes to use information from ambiguous or overlapped peaks, there is an additional "MAP" file that makes a translation from peak identifiers to ambiguous (or partial) assignments. Finally, there are some specialized (but still pretty straightforward) file formats for chemical shift or residual dipolar coupling restraints.

There are a variety of tools, besides the ones described below, that can assist in preparing input for structure refinement in Amber. The SANE (Structure Assisted NOE Evaluation) package,

```
http://garbanzo.scripps.edu/nmrgrp/wisdom/sane/sane.html
```

is widely used at The Scripps Research Institute [89]. If you use Bruce Johnson's NmrView package, you might also want to look at the TSRI additions to that:

```
http://garbanzo.scripps.edu/nmrgrp/wisdom/pipe/tips_scripts.html
```

In particular, the *xpkTOupl* and *starTOupl* scripts there convert NmrView peak lists into the "7-column" needed for input to makeDIST_RST.

Users of the MARDIGRAS programs from UCSF can use the *mardi2amber* program to

do conversion to Amber format:

```
http://picasso.ucsf.edu/mardihome.html
```

(2) *restrained molecular dynamics*, which is at the heart of the conformational searching procedures. This is the part that *sander* itself handles.

(3) *back-end* routines that do things like compare families of structures, generate statistics, simulate spectra, and the like. For many purposes, such as visualization, or the running of procheck-NMR, the "interface" to such programs is just the set of pdb-format files that contain the family of structures to be analyzed. These general-purpose structure analysis programs are available in many locations and are not discussed here. The principal *sander*-specific tool is *sviol*, which prepares tables and statistics of energies, restraint violations, and the like.

5.20.1. Preparing restraint files for Sander

Figure 1 shows the general information flow for auxiliary programs that help prepare the restraint files. Once the restraint files are made, Figure 2 shows a flow-chart of the general way in which *sander* refinements are carried out.

The basic ideas of this scheme owe a lot to the general experience of the NMR community over the past decade. Several papers outline procedures in the Scripps group, from which a lot of the NMR parts of *sander* are derived [89-95]. They are by no means the only way to proceed. We hope that the flexibility incorporated into *sander* will encourage folks to experiment with refinement protocols.

5.20.2. Preparing distance restraints: makeDIST_RST.

The *makeDIST_RST* program converts a simplified description of distance bounds into a detailed input for *sander*. A variety of input and output filenames may be specified on the command line:

```
input:
   -upb <filename>          7-col file of upper distance bounds, OR
   -ual <filename>          8-col file of upper and lower bounds, OR
   -vol <filename>          7-col file of NOESY volumes

   -pdb <filename>          Brookhaven format file
   -map <filename>          MAP file  (default:map.DG-AMBER)
   -les <filename>          LES atom mappings, made by addles

output:
   -dgm <filename>          DGEOM95 restraint format
   -rst <filename>          SANDER restraint format
   -svf <filename>          Sander Volume Format, for NOESY refinement

other options:
   -help        (gives you this explanation, overrides other parameters)
```

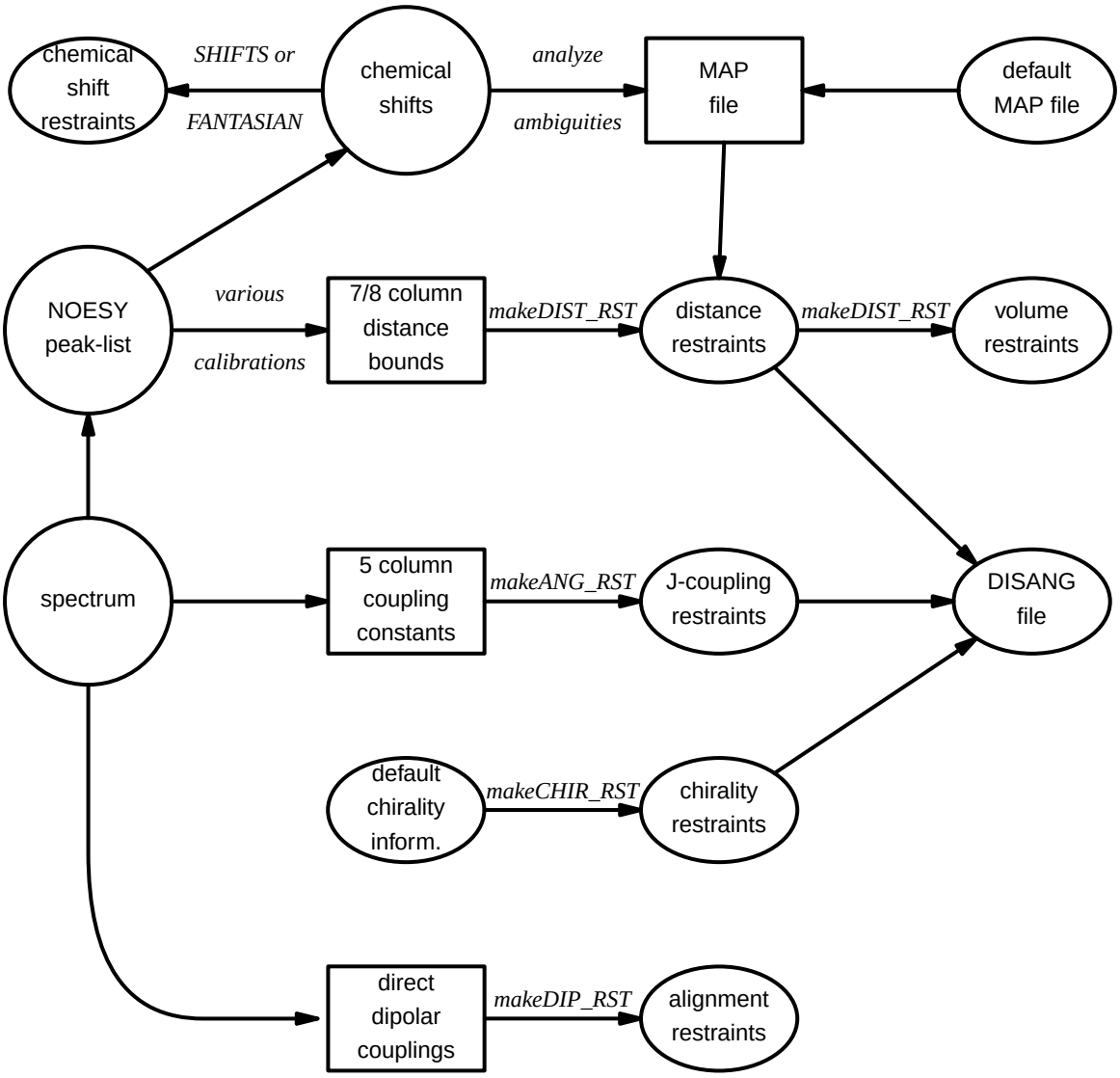


Fig. 1. Notation: *circles* represent logical information, whose format might differ from one project to the next; *solid rectangles* are in a specific format (largely compatible with DIANA and other programs), and are intended to be read and edited by the user; *ellipses* are specific to *sander*, and are generally not intended to be read or edited manually. The conversion of NOESY volumes to distance bounds can be carried out by a variety of programs such as *mardigras* or *xpk2bound* that are not included with Amber. Similarly, the analysis and partial assignment of ambiguous or overlapped peaks is a separate task; at TSRI, these are typically carried out using the programs *xpkasgn* and *filter.pl*.

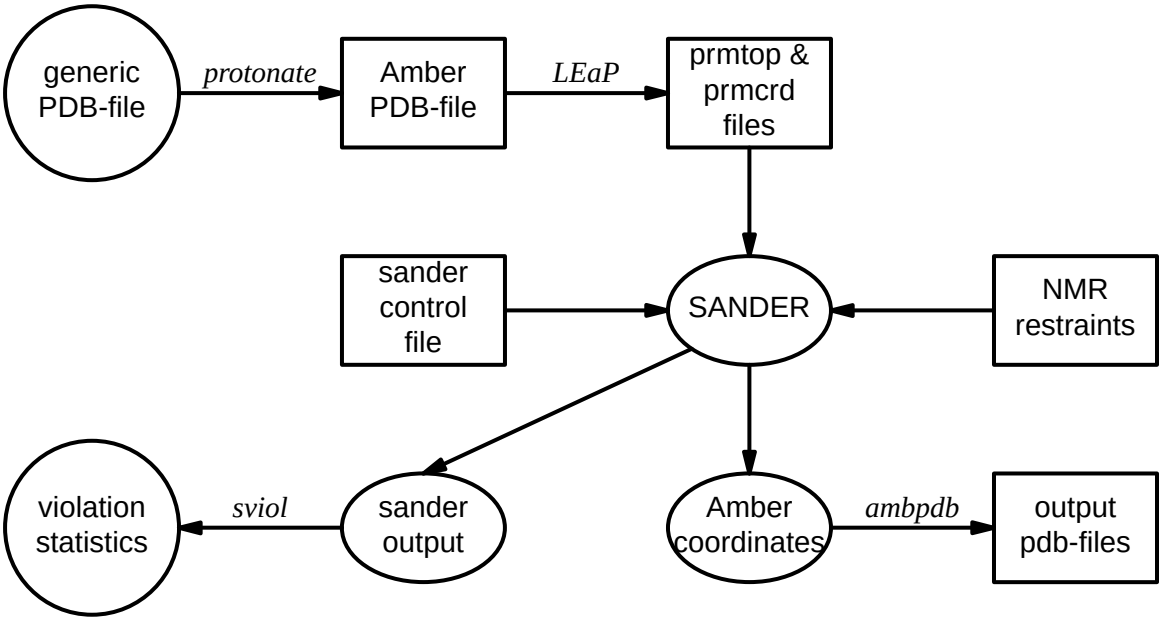


Fig. 2

`-report`	*(gives you short runtime diagnostic output)*
`-nocorr`	*(do not correct upper bound for r**-6 averaging)*
`-altdis`	*(use alternative form for the distance restraints)*

The *7/8 column distance bound* file is essentially that used by the DIANA or DISGEO programs. It consists of one-line per restraint, which would typically look like the following:

```
23   ALA   HA      52  VAL  H    3.8  # comments go here
```

The first three columns identify the first proton, the next three the second proton, and the seventh column gives the upper bound. Only the first three letters of the residue name are used, so that DIANA files that contain residues like "ASP−" will be correctly interpreted. An alternate, 8-column, format has both upper and lower bounds as the seventh and eighth columns, respectively. A typical line might in an "8-col" file might look like this:

```
23   ALA   HA      52  VAL  H   3.2   3.8  # comments go here
```

Here the lower bound is 3.2 Å and the upper bound is 3.8 Å. Comments typically identify the spectrum and peak-number or other identification that allow cross-referencing back to the appropriate spectrum. If the comment contains the pattern "<integer>:<integer>", then the first integer is treated as a peak-identifier, and the second as a spectrum-identifier. These identifiers go into the *ixpk* and *nxpk* variables, and will later be printed out in *sander*, to facilitate going back to the original spectra to track down violations, etc.

The format for the *-vol* option is the same as for the *-upb* option except that the seventh column holds a peak intensity (volume) value, rather than a distance upper bound.

The input pdb file must exactly match the Amber *prmtop* file that will be used; use the `ambpdb -aatm` command to create this.

If all peaks involved just single protons, and were fully assigned, this is all that one would need. In general, though, some peaks (especially methyl groups or fast-rotating aromatic rings) represent contributions from more than one proton, and many other peaks may not be fully assigned. *Sander* handles both of these situations in the same way, through the notion of an "ambiguous" peak, that may correspond to several assignments. These peaks are given two types of special names in the 7/8-column format file:

(1) Commonly-occurring ambiguities, like the lack of stereospecific assignments to two methylene protons, are given names defined in the default MAP file. These names, also more-or-less consistent with DIANA, are like the names of "pseudo-atoms" that have long been used to identify such partially assigned peaks, e.g. "QB" refers to the (HB2,HB3) combination in most residues, and "MG1" in valine refers collectively to the three methyl protons at position CG1, etc.

(2) There are generally also molecule-specific ambiguities, arising from potential overlap in a NOESY spectrum. Here, the user assigns a unique name to each such ambiguity or overlap, and prepares a list of the potential assignments. The names are arbitrary, but might be constructed, for example, from the chemical shifts that identify the peak, e.g. "p_2.52" might identify the set of protons that could contribute to a peak at 2.52 ppm. The chemical shift list can be used to prepare a list of potential assignments, and these lists can often be pruned by comparison to approximate or initial structures.

The default and molecule-specific MAP files are combined into a single file, which is used, along with the 7-column restraint file, the the program *makeDIST_RST* to construct the actual *sander* input files. You should consult the help file for makeDIST_RST for more information. For example, here are some lines added to the MAP file for a recent TSRI refinement:

```
AMBIG n2:68  =  HE 86 HZ 86
AMBIG n2:72  =  HE 24 HD 24 HZ 24
AMBIG n2:73  =  HN 81 HZ 13 HE 13 HD 13 HZ 24
AMBIG n2:78  =  HN 76 HZ 13 HE 13 HZ 24
AMBIG n2:83  =  HN 96 HN 97 HD 97 HD 91
AMBIG n2:86  =  HD1 66 HZ2 66
AMBIG n2:87  =  HN 71 HH2 66 HZ3 66 HD1 66
```

Here the spectrum name and peak number were used to construct a label for each ambiguous peak. Then, an entry in the restraint file might look like this:

```
123 GLY HN  0   AMB n2:68   5.5
```

indicating a 5.5 Å upper bound between the amide proton of Gly 123 and a second proton, which might be either the HE or HZ protons of residue 86. (The "zero" residue number just serves as a placeholder, so that there will be the same number of columns as for non-ambiguous restraints.) If it is possible that the ambiguous list might not be exhaustive (e.g. if some protons have not been assigned), it is safest to set *ialtd*=1, which will allow "mistakes" to be present in the constraint list. On the other hand, if you want to be sure that every violation is "active", set *ialtd*=0.

If the *-les* flag is set, the program will prepare distance restraints for multiple copies (LES) simulations. In this case, the input pdb file is one *without* LES copies, i.e. with just a single copy of the molecule. The "lesfile" specified by this flag is created by the *addles* program, and contains a mapping from original atom numbers into the copy numbers used in the multiple-copies simulation.

The *-rst* and *-svf* flags specify outputs for *sander*, for distance restraints and NOESY restraints, respectively. In each case, you may need to hand-edit the outputs to add additional parameters. You should make it a habit to compare the outputs with the descriptions given earlier in this chapter to make sure that the restraints are what you want them to be.

It is common to run makeDIST_RST several times, with different inputs that correspond to different spectra, different mixing times, etc. It is then expected that you will manually edit the various ouptut files to combine them into the single file required by *sander*.

5.20.3. Preparing torsion angle restraints: makeANG_RST

There are fewer "standards" for representing coupling constant information. We have followed the DIANA convention in the program *makeANG_RST*. This program takes as input a five-column torsion angle constraint file along with an AMBER pdb file of the molecule. It creates as output (to standard out) a list of constraints in RST format that is readable by AMBER.

```
Usage: makeANG_RST -help
       makeANG_RST -pdb ambpdb_file [-con constraint] [-lib libfile]
                [-les lesfile ]
```

The input torsion angle constraint file can be read from standard in or from a file specified by the -con option on the command line. The input constraint file should look something like this:

```
1   GUA   PPA     111.5   144.0
2   CYT   EPSILN   20.9   100.0
2   CYT   PPA     115.9   134.2
3   THY   ALPHA    20.4   35.6
4   ADE   GAMMA    54.7   78.8
5   GLY   PHI      30.5   60.3
6   ALA   CHI      20.0   50.0
   ....
```

Lines beginning with "#" are ignored. The first column is the residue number; the second is the residue name (three letter code, or as defined in your personal torsion library file). Only the first three letters of the residue name are used, so that DIANA files that contain residues like "ASP-" will be correctly interpreted. Third is the angle name (taken from the torsion library described below).The fourth column contains the lower bound, and the fifth column specifies the upper bound. Additional material on the line is (presently) ignored.

Note: It is assumed that the lower bound and the upper bound define a region of allowed conformation on the unit circle that is swept out in a clockwise direction from $lb \rightarrow ub$. If the number in the *lb* column is greater than the the number in the *ub* column, 360^o will successively be subtracted from the *lb* until $lb < ub$. This preserves the clockwise definition of the allowed conformation space, while also making the number that specifies the lower bound less than the

number that specifies the upper bound, as is required by AMBER. If this occurs, a warning message will be printed to *stderr* to notify the user that the data has been modified.

The angles that one can constrain in this manner are defined in the library file that can be optionally specified on the command line with the -lib flag, or the default library "tordef.lib" (written by Garry P. Gippert) will be used. If you wish to specify your own nomenclature, or add angles that are not already defined in the default file, you should make a copy of this file and modify it to suit your needs. The general format for an entry in the library is:

```
LEU   PSI     N    CA   C    N+
```

where the first column is the residue name, the second column is the angle name that will appear in the input file when specifying this angle, and the last four columns are the atom names that define the torsion angle. When a torsion angle contains atom(s) from a preceding or succeeding residue in the structure, a "-" or "+" is appended to those atom names in the library, thereby specifying that this is the case. In the example above, the atoms that define PSI for LEU residues are the N, CA, and C atoms of that same LEU and the N atom of the residue after that LEU in the primary structure. Note that the order of atoms in the definition is important and should reflect that the torsion angle rotates about the two central atoms as well as the fact that the four atoms are bonded in the order that is specified in the definition.

If the first letter of the second field is "J", this torsion is assumed to be a J-coupling constraint. In that case, three additional floats are read at the end of the line, giving the A,B and C coefficients for the Karplus relation for this torsion. For example:

```
ALA JHNA    H    N    CA   HA   9.5  -1.4  0.3
```

will set up a J-coupling restraint for the HN-HA 3-bond coupling, assuming a Karplus relation with A,B, C as 9.5, -1.4 and 0.3. (These particular values are from Brüschweiler and Case, JACS 116: 11199 (1994).)

This program also supports pseudorotation phase angle constraints for prolines and nucleic acid sugars; each of these will generate restraints for the 5 component angles which correspond to the *lb* and *ub* values of the input pseudorotation constraint. In the torsion library, a pseudorotation definition looks like:

```
PSEUDO        CYT     PPA     NU0     NU1     NU2     NU3     NU4
CYT   NU0     C4'     O4'     C1'     C2'
CYT   NU1     O4'     C1'     C2'     C3'
CYT   NU2     C1'     C2'     C3'     C4'
CYT   NU3     C2'     C3'     C4'     O4'
CYT   NU4     C3'     C4'     O4'     C1'
```

The first line describes that a PSEUDOrotation angle is to be defined for CYT that is called PPA and is made up of the five angles NU0-NU4. Then the definition for NU0-NU4 should also appear in the file in the same format as the example given above for LEU PSI.

PPA stands for Pseudorotation Phase Angle and is the angle that should appear in the input constraint file when using pseudorotation constraints. The program then uses the definition of that PPA angle in the library file to look for the 5 other angles (NU0-NU4 in this case) which it then generates restraints for. PPA for proline residues is included in the standard library as well as for the DNA nucleotides.

If the *-les* flag is set, the program will prepare torsion angle restraints for multiple copies (LES) simulations. In this case, the input pdb file is one *without* LES copies, i.e. with just a single copy of the molecule. The "lesfile" specified by this flag is created by the *addles* program, and contains a mapping from original atom numbers into the copy numbers used in the multiple-copies simulation.

Torsion angle constraints defined here cannot span two different copy sets, i.e., there cannot be some atoms of a particular torsion that are in one multiple copy set, and other atoms from the same torsion that are in other copy sets. It *is* OK to have some atoms with single copies, and others with multiple copies in the same torsion. The program will create as many duplicate torsions as there are copies.

A good alternative to interpreting J-coupling constants in terms of torsion angle restraints is to refine directly against the coupling constants themselves, using an appropriate Karplus relation. See the discussion of the variable `RJCOEF`, above.

5.20.4. Chirality restraints: makeCHIR_RST

```
Usage:  makeCHIR_RST <pdb-file> <output-constraint-file>
```

We also find it useful to add chirality constraints and *trans*-peptide ω constraints (where appropriate) to prevent chirality inversions or peptide bond flips during the high-temperature portions of simulated annealing runs. The program *makeCHIR_RST* will create these constraints. Note that you may have to edit the output of this program to change *trans* peptide constraints to *cis*, as appropriate.

5.20.5. Direct dipolar coupling restraints: makeDIP_RST

For simulations with residual dipolar coupling restraints, the *makeDIP_RST.protein*, *makeDIP_RST.dna* and *makeDIP_RST.diana* are simple codes to prepare the input file. Use *-help* to obtain a more detailed description of the usage. For now, this code only handles backbone NH and CαH data. The header specifying values for various parameters needs to be manually added to the output of *makeDIP_RST*.

Use of residual dipolar coupling restraints is new both for AMBER and for the general NMR community. Refinement against these data should be carried out with care, and the optimal values for the force constant, penalty function, and initial guesses for the alignment tensor components are still under investigation. Here are some suggestions from the experiences so far:

(1) Beware of overfitting the dipolar coupling data in the expense of AMBER force field energy. These dipolar coupling data are very sensitive to tiny changes in the structure. It is often possible to drastically improve the fitting by making small distortions in the backbone angles. We recommend inclusion of explicit angle restraints to enforce ideal backbone geometry, especially for those residues that have corresponding residual dipolar coupling data.

(2) The initial values for the Cartesian components of the alignment tensor can influence the final structure and alignment if the structure is not fixed (ibelly = 0). For a fixed structure (ibelly = 1), these values do not matter. Therefore, the current "best" strategy is to fit the experimental data to the fixed starting structure, and use the alignment tensor[s] obtained from this fitting as the initial guesses for further refinement.

(3) AMBER is capable of simultaneously fitting more than one set of alignment data. This allows the use of individually obtained datasets with different alignment tensors. However, if the different sets of data have equal directions of alignment but different magnitudes, using an overall scaling factor for these data with a single alignment tensor could greatly reduce the number of fitting parameters.

(4) Because the dipolar coupling splittings depend on the square root of the order parameters ($0 \leq S2 \leq 1$), these order parameters describing internal motion of individual residues are often neglected (N. Tjandra and A. Bax, *Science* **278**, 1111-1113, 1997). However, the square root of a small number can still be noticeably smaller than 1, so this may introduce undesirable errors in the calculations.

5.20.6. Getting summaries of NMR violations

If you specify `LISTOUT=POUT` when running *sander*, the output file will contain a lot of detailed information about the remaining restraint violations at the end of the run. When running a family of structures, it can be useful to process these ouput files with *sviol*, which takes a list of *sander* output files on the command line, and sends a summary of energies and violations to STDOUT. If you have more than 20 or so structures to analyze, the output from *sviol* becomes unwieldy. In this case you may also wish to use *sviol2*, which prints out somewhat less detailed information, but which can be used on larger families of structures. The *senergy* script gives a more detailed view of force-field energies from a series of structures. (We thank the TSRI NMR community for helping to put these scripts together, and for providing many useful suggestions.)

5.20.7. Time-averaged restraints.

The model of the previous sections involves the "single-average-structure" idea, and tries to fit all constraints to a single model, with minimal deviations. A generalization of this model treats distance constraints arising from from NOE crosspeaks (for example) as being the average distance determined from a trajectory, rather than as the single distance derived from an average structure. Time-averaged bonds and angles are calculated as

$$\bar{r} \;=\; (1/C)\;\left\{\;\int_0^t e^{(t'-t)/\tau} r(t')^{-i}\,dt'\;\right\}^{-1/i} \qquad (1)$$

where

$\bar{r}$ = time-averaged value of the internal coordinate (distance or angle)

t = the current time

τ = the exponential decay constant

$r(t')$ = the value of the internal coordinate at time t'

i = average is over internals to the inverse of i. Usually i = 3 or 6 for NOE distances, and −1 (linear averaging) for angles and torsions.

C = a normalization integral.

Time-averaged torsions are calculated as

$$< \phi > \;=\; \tan^{-1}(< \sin(\phi) > / < \cos(\phi) >) \qquad (2)$$

where ϕ is the torsion, and $< \sin(\phi) >$ and $< \cos(\phi) >$ are calculated using the equation above with

$\sin(\phi(t'))$ or $\cos(\phi(t'))$ substituted for r(t').

Forces for time-averaged restraints can be calculated either of two ways. This option is chosen with the DISAVI / ANGAVI / TORAVI commands (Section 1). In the first (the default),

$$\partial E/\partial x \;=\; (\partial E/\partial \bar{r})\,(\partial \bar{r}/\partial r(t))\,(\partial r(t)/\partial x) \tag{3}$$

(and analogously for y and z). The forces then correspond to the standard flat-bottomed well functional form, with the instantaneous value of the internal replaced by the time-averaged value. For example, when $r_3 < \bar{r} < r_4$,

$$E \;=\; k_3(\bar{r} - r_3)^2 \tag{4}$$

and similarly for other ranges of $\bar{r}$.

When the second option for calculating forces is chosen (IINC = 1 on a DISAVI, ANGAVI or TORAVI card), forces are calculated as

$$\partial E/\partial x \;=\; (\partial E/\partial \bar{r})\,(\partial r(t)/\partial x) \tag{5}$$

For example, when $r_3 < \bar{r} < r_4$,

$$\partial E/\partial x \;=\; 2\;k_3\,(\bar{r} - r_3)\,(\partial r(t)/\partial x) \tag{6}$$

Integration of this equation does not give Equation (4), but rather a non-intuitive expression for the energy (although one that still forces the bond to the target range). The reason that it may sometimes be preferable to use this second option is that the term $\partial \bar{r}/\partial r(t)$, which occurs in the exact expression [Eq. (3)], varies as $(\bar{r}/r(t))^{1+i}$. When i=3, this means the forces can be varying with the fourth power the distance, which can possibly lead to very large transient forces and instabilities in the molecular dynamics trajectory. [Note that this will not be the case when linear scaling is performed, i.e. when i=−1, as is generally the case for valence and torsion angles. Thus, for linear scaling, the default (exact) force calculation should be used].

It should be noted that forces calculated using Equation (5) are not conservative forces, and would cause the system to gradually heat up, if no velocity rescaling were performed. The temperature coupling algorithm should act to maintain the average temperature near the target value. At any rate, this heating tendency should not be a problem in simulations, such as fitting NMR data, where MD is being used to sample conformational space rather than to extract thermodynamic data.

This section has described the methods of time-averaged restraints. For more discussion, the interested user is urged to consult studies where this method has been used [96-100].

5.20.8. Multiple copies refinement using LES

NMR restraints can be made compatible with the multiple copies (LES) facility; see the following chapter for more information about LES. To use NMR constraints with LES, you need to do two things:

(1) Add a line like "`file wnmr name=(lesnmr) wovr`" to your input to *addles*. The filename (`lesnmr` in this example) may be whatever you wish. This will cause *addles* to output an additional file that is needed at the next step.

(2) Add "`-les lesnmr`" to the command line arguments to *makeDIST_RST*. This will read in the file created by *addles* containing information about the copies. All NMR restraints will then be interpreted as "ambiguous" restraints, so that if any of the copies satisfies the restraint, the penalty goes to zero.

Note that although this scheme has worked well on small peptide test cases, we have yet not used it extensively for larger problems. This should be treated as an experimental option, and users should use caution in applying or interpreting the results.

5.20.9. Some sample input files

The next few pages contain excerpts from some sample NMR refinement files used at TSRI. The first example just sets up a simple (but often effective) simulated annealing run. You may have to adjust the length, temperature maximum, etc. somewhat to fit your problem, but these values work well for many "ordinary" NMR problems.

1. Simulated annealing NMR refinement

```
 15ps simulated annealing protocol
 &cntrl
    nstlim=15000, ntt=1,          (time limit, temp. control)
    scee=1.2,                     (scee must be set - 1-4 scale factor)
    ntpr=500, pencut=0.1,         (control of printout)
    ipnlty=1, nmropt=1,           (NMR penalty function options)
    vlimit=10,                    (prevent bad temp. jumps)
    ntb=0,                        (non-periodic simulation)
 /
 &ewald
    eedmeth=5,                    (use r dielectric)
 /
#
# Simple simulated annealing algorithm:
#
#  from steps      0 to  1000: raise target temperature 10->1200K
#  from steps  1000 to  3000: leave at 1200K
#  from steps  3000 to 15000: re-cool to low temperatures
#
 &wt type='TEMP0', istep1=0,istep2=1000,value1=10.,
            value2=1200.,    /
 &wt type='TEMP0', istep1=1001, istep2=3000, value1=1200.,
            value2=1200.0,     /
 &wt type='TEMP0', istep1=3001, istep2=15000, value1=0.,
            value2=0.0,     /
#
# Strength of temperature coupling:
#   steps      0 to  3000: tight coupling for heating and equilibration
#   steps  3000 to 11000: slow cooling phase
#   steps 11000 to 13000: somewhat faster cooling
#   steps 13000 to 15000: fast cooling, like a minimization
#
 &wt type='TAUTP', istep1=0,istep2=3000,value1=0.2,
            value2=0.2,     /
 &wt type='TAUTP', istep1=3001,istep2=11000,value1=4.0,
            value2=2.0,     /
 &wt type='TAUTP', istep1=11001,istep2=13000,value1=1.0,
            value2=1.0,     /
 &wt type='TAUTP', istep1=13001,istep2=14000,value1=0.5,
            value2=0.5,    /
 &wt type='TAUTP', istep1=14001,istep2=15000,value1=0.05,
            value2=0.05,    /
```

(continued on next page)

1. Simulated annealing NMR refinement *(continued)*

```
#
# "Ramp up" the restraints over the first 3000 steps:
#
 &wt type='REST', istep1=0,istep2=3000,value1=0.1,
            value2=1.0,  /
 &wt type='REST', istep1=3001,istep2=15000,value1=1.0,
            value2=1.0,  /

 &wt type='END'  /
LISTOUT=POUT                    (get restraint violation list)
DISANG=RST.f                    (file containing NMR restraints)
```

The next example just shows some parts of the actual `RST` file that *sander* would read. This file would ordinarily *not* be made or edited by hand; rather, run the programs *makeDIST_RST*, *makeANG_RST* and *makeCHIR_RST*, combining the three outputs together to construct the `RST` file.

2. Part of the RST.f file referred to above

```
# first, some distance constraints prepared by makeDIST_RST:
#     (comment line is input to makeRST, &rst namelist is output)
#
#(   proton 1          proton 2          upper bound)
#---------------------------------------------
#
# 2 ILE HA       3 ALA HN         4.00
#
 &rst iat=  23,  40, r3= 4.00, r4= 4.50,
      r1 = 1.3, r2 = 1.8, rk2=0.0, rk3=32.0, ir6=1, /
#
# 3 ALA HA       4 GLU HN         4.00
#
 &rst iat=  42,  50, r3= 4.00, r4= 4.50,  /
#
# 3 ALA HN       3 ALA MB         5.50
#
 &rst iat=  40,  -1, r3= 6.22, r4= 6.72,
      igr1=   0,   0,   0,   0, igr2=  44,  45,  46,   0,   /
#
# .....etc.....
```

2. Part of the RST.f file referred to above *(continued)*

```
#
#  next, some dihedral angle constraints, from makeANG_RST:
#
 &rst iat= 213, 215, 217, 233, r1=-190.0,
      r2=-160.0, r3= -80.0, r4= -50.0, /

 &rst iat= 233, 235, 237, 249, r1=-190.0,
      r2=-160.0, r3= -80.0, r4= -50.0, /

# .......etc.......
#
#  next, chirality and omega constraints prepared by makeCHIR_RST:
#
#
#  chirality for residue  1  atoms:   CA CG HB2 HB3
 &rst iat= 3 , 8 , 6 , 7 ,
   r1=10., r2=60., r3=80., r4=130., rk2 = 10., rk3=10.,  /
#
#  chirality for residue  1  atoms:   CB SD HG2 HG3
 &rst iat= 5 , 11 , 9 , 10 , /
#
#  chirality for residue  1  atoms:   N C HA CB
 &rst iat= 1 , 18 , 4 , 5 , /
#
#  chirality for residue  2  atoms:   CA CG2 CG1 HB
 &rst iat= 22 , 26 , 30 , 25 , /
#
 ......etc.........

# trans-omega constraint for residue  2
 &rst iat= 22 , 20 , 18 , 3 ,
   r1=155., r2=175., r3=185., r4=205., rk2 = 80., rk3=80.,  /
#
# trans-omega constraint for residue  3
 &rst iat= 41 , 39 , 37 , 22 , /
#
# trans-omega constraint for residue  4
 &rst iat= 51 , 49 , 47 , 41 , /
#
# ......etc.........
#
```

The next example is an input file for volume-based NOE refinement. As with the distance/angle `RST` file shown above, the user would generally not construct this file, but create it from a "7-column" file using the makeDIST_RST program. Hand-editing might be used at the top of the file, to change the correlation times, etc.

3. Sample NOESY intensity input file

```
# A part of a NOESY intensity file:

 &noeexp
  id2o=1,                  (exchangeable protons removed)
  oscale=6.21e-4,          (scale between exp. and calc. intensity units)
  taumet=0.04,             (correlation time for methyl rotation, in ns.)
  taurot=4.2,              (protein tumbling time, in ns.)
  NPEAK = 13*3,            (three peaks, each with 13 mixing times)
  EMIX = 2.0E-02,  3.0E-02,  4.0E-02,  5.0E-02,  6.0E-02,
     8.0E-02,  0.1,  0.126,  0.175,  0.2,  0.25,  0.3,  0.35,
                           (mixing times, in sec.)
  IHP(1,1) = 13*423,  IHP(1,2) = 13*1029,  IHP(1,3) = 13*421,
                           (number of the first proton)
  JHP(1,1) = 78*568,  JHP(1,2) = 65*1057,  JHP(1,3) = 13*421,
                           (number of the second proton)
  AEXP(1,1) = 5.7244,  7.6276,  7.7677,  9.3519,
              10.733,  15.348,  18.601,
              21.314,  26.999,  30.579,
              33.57,  37.23,  40.011,
                           (intensities for the first cross-peak)
  AEXP(1,2) =  8.067,  11.095,  13.127,  18.316,
              22.19,   26.514,  30.748,
              39.438,  44.065,  47.336,
              54.467,  56.06,   60.113,
  AEXP(1,3) =  7.708,  13.019,  15.943,  19.374,
              25.322,  28.118,  35.118,
              40.581,  49.054,  53.083,
              56.297,  59.326,  62.174,
 /
SUBMOL1
RES 27 27 29 29 39 41 57 57 70 70 72 72 82 82     (residues in this submol)
END
END
```

Next, we illustrate the form of the file that holds residual dipolar coupling restraints. Again, this would generally be created from a human-readable input using the program *makeDIP_RST*.

5. Residual dipolar restraints, prepared by makeDIP_RST:

```
&align
  ndip=91, dcut=-1.0, gigj = 37*-3.1631, 54*7.8467,
  s11=3.883, s22=53.922, s12=33.855, s13=-4.508, s23=-0.559,
  id(1)=188,   jd(1)=189,  dobsu(1)=  6.24, dobsl(1)= 6.24,
  id(2)=208,   jd(2)=209,  dobsu(2)= -10.39, dobsl(1)= -10.39,
  id(3)=243,   jd(3)=244,  dobsu(3)= -8.12, dobsl(1)= -8.12,

  ....
  id(91)=1393,   jd(91)=1394,  dobsu(91)= -19.64,  dobsl(91) = -19.64,
/
```

Finally, we show how the detailed input to *sander* could be used to generate a more complicated restraint. Here is where the user would have to understand the details of the `RST` file, since there are no "canned" programs to create this sort of restraint. This illustrates, though, the potential power of the program.

5. A more complicated constraint

```
# 1)   Define two centers of mass. COM1 is defined by
#      {C1 in residue 1; C1 in residue 2; N2 in residue 3; C1 in residue 4}.
#      COM2 is defined by {C4 in residue 1; O4 in residue 1;  N* in residue 1}.
#      (These definitions are effected by the igr1/igr2 and grnam1/grnam2
#      variables; You can use up to 200 atoms to define a center-of-mass
#      group)
#
# 2)   Set up a distance restraint between COM1 and COM2 which goes from a
#      target value of 5.0A to 2.5A, with a force constant of 1.0, over steps 1-5000.
#
# 3)   Set up a distance restraint between COM1 and COM2 which remains fixed
#      at the value of 2.5A as the force slowly constant decreases from
#      1.0 to 0.01 over steps 5001-10000.
#
# 4)   Sets up no distance restraint past step 10000, so that free (unrestrained)
#      dynamics takes place past this step.
#

 &rst iat=-1,-1, nstep1=1,nstep2=5000,
    iresid=1,irstyp=0,ifvari=1,ninc=0,imult=0,ir6=0,ifntyp=0,
    r1=0.00000E+00,r2=5.0000,r3=5.0000,
       r4=99.000,rk2=1.0000,rk3=1.0000,
    r1a=0.00000E+00,r2a=2.5000,r3a=2.5000,
       r4a=99.000,rk2a=1.0000,rk3a=1.0000,
    igr1 = 2,3,4,5,0,
    grnam1(1)='C1',grnam1(2)='C1',grnam1(3)='N2',grnam1(4)='C1',
    igr2 = 1,1,1,0,
    grnam2(1)='C4',grnam2(2)='O4',grnam2(3)='N*',
 /
 &rst iat=-1,-1, nstep1=5001,nstep2=10000,
    iresid=1,irstyp=0,ifvari=1,ninc=0,imult=0,ir6=0,ifntyp=0,
    r1=0.00000E+00,r2=2.5000,r3=2.5000,
       r4=99.000,rk2=1.0000,rk3=1.0000,
    r1a=0.00000E+00,r2a=2.5000,r3a=2.5000,
       r4a=99.000,rk2a=1.0000,rk3a=0.0100,
    igr1 = 2,3,4,5,0,
    grnam1(1)='C1',grnam1(2)='C1',grnam1(3)='N2',grnam1(4)='C1',
    igr2 = 1,1,1,0,
    grnam2(1)='C4',grnam2(2)='O4',grnam2(3)='N*',
 /
```

5.21. Getting debugging information

The debug options in *sander* are there principally to help developers test new options or to test results between two machines or versions of code, but can also be useful to users who want to test the effect of parameters on the accuracy of their ewald or pme calculations. If the debug options are set, *sander* will exit after performing the debug tasks set by the user.

To access the debug options, include a `&debugf` namelist. Input parameters are:

DO_DEBUGF Flag to perform this module. Possible values are zero or one. Default is zero. Set to one to turn on debug options.

One set of options is to test that the atomic forces agree with numerical differentiation of energy.

ATOMN Array of atom numbers to test atomic forces on. Up to 25 atom numbers can be specified, separated by commas.

NRANATM number of random atoms to test atomic forces on. Atom numbers are generated via a random number generator.

RANSEED seed of random number generator used in generating atom numbers default is 71277

NEGLGDEL negative log of delta used in numerical differentiating; e.g. 4 means delta is 10^-4 Angstroms. Default is 5. *Note*: In general it does no good to set nelgdel larger than about 6. This is because the relative force error is at best the square root of the numerical error in the energy, which ranges from 10^-15 up to 10^-12 for energies involving a large number of terms.

CHKVIR Flag to test the atomic and molecular virials numerically. Default is zero. Set to one to test virials.

DUMPFRC Flag to dump energies, forces and virials, as well as components of forces (bond, angle forces etc.) to the file "forcedump.dat" This produces an ascii file. Default is zero. Set to one to dump forces.

RMSFRC Flag to compare energies forces and virials as well as components of forces (bond, angle forces etc.) to those in the file "forcedump.dat". Default is zero. Set to one to compare forces.

Several other options are also possible to modify the calculated forces.

ZEROCHG Flag to zero all charges before calculating forces. Default zero. Set to one to remove charges.

ZEROVDW Flag to remove all van der Waals interactions before calculating forces. Default zero. Set to one to remove van der Waals.

ZERODIP Flag to remove all atomic dipoles before calculating forces. Only relevant when polarizability is invoked.

CONST,DO_CAP
DO_DIR,DO_REC,DO_ADJ,DO_SELF,DO_BOND,DO_CBOND,DO_ANGLE,DO_EPHI,DOX- These are flags which turn on or off the subroutines they refer to. The defaults are one. Set to zero to prevent a subroutine from running. For example, set do_dir=0 to turn off the direct sum interactions (van der Waals as well as electrostatic). Thes options, as well as the zerochg, zerovdw,zerodip flags, can be used to fine tune a test of forces, accuracy etc.

EXAMPLES:

This input list tests the reciprocal sum forces on atom 14 numerically, using a delta of 10^-4.

```
&debugf
 neglgdel=4, nranatm = 0, atomn = 14,
 do_debugf = 1,do_dir = 0,do_adj = 0,do_rec = 1, do_self = 0,
 do_bond = 1,do_angle = 0,do_ephi = 0, zerovdw = 0, zerochg = 0,
 chkvir = 0,
 dumpfrc = 0,
 rmsfrc = 0,
/
```

This input list causes a dump of force components to "forcedump.dat". The bond, angle and dihedral forces are not calculated, and van der Waals interactions are removed, so the total force is the Ewald electrostatic force, and the only non-zero force components calculated are electrostatic.

```
&debugf
 neglgdel=4, nranatm = 0, atomn = 0,
 do_debugf = 1,do_dir = 1,do_adj = 1,do_rec = 1, do_self = 1,
 do_bond = 0,do_angle = 0,do_ephi = 0, zerovdw = 1, zerochg = 0,
 chkvir = 0,
 dumpfrc = 1,
 rmsfrc = 0,
/
```

In this case the same force components as above are calculated, and compared to those in "forcedump.dat". Typically this is used to get an RMS force error for the Ewald method in use. To do this, when doing the force dump use ewald or pme parameters to get high accuracy, and then normal parameters for the force compare:

```
&debugf
 neglgdel=4, nranatm = 0, atomn = 0,
 do_debugf = 1,do_dir = 1,do_adj = 1,do_rec = 1, do_self = 1,
 do_bond = 0,do_angle = 0,do_ephi = 0, zerovdw = 1, zerochg = 0,
 chkvir = 0,
 dumpfrc = 0,
 rmsfrc = 1,
/
```

For example, if you have a 40x40x40 unit cell and want to see the error for default pme options (cubic spline, 40x40x40 grid), run 2 jobs------ (assume box params on last line of inpcrd file)

Sample input for 1st job:

```
&cntrl
 dielc =1.0, scee   = 1.2,
 cut    = 11.0,     nsnb   = 5,      ibelly = 0,
 ntx    = 7,       irest  = 1,
 ntf    = 2,       ntc    = 2,       tol    = 0.0000005,
 ntb    = 1,       ntp    = 0,       temp0 = 300.0, tautp = 1.0,
nstlim = 1,    dt = 0.002,  maxcyc = 5,      imin = 0,  ntmin = 2,
 ntpr   = 1,     ntwx = 0, ntt = 0, ntr = 0,
          jfastw = 0, nmrmax=0, ntave = 25,
/
&debugf
 do_debugf = 1,do_dir = 1,do_adj = 1,do_rec = 1, do_self = 1,
 do_bond = 0,do_angle = 0,do_ephi = 0, zerovdw = 1, zerochg = 0,
 chkvir = 0,
 dumpfrc = 1,
 rmsfrc = 0,
/
&ewald
  nfft1=60,nfft2=60,nfft3=60,order=6, ew_coeff=0.35,
/
```

Sample input for 2nd job:

```
&cntrl
 dielc =1.0, scee   = 1.2,
 cut    = 8.0,     nsnb   = 5,      ibelly = 0,
 ntx    = 7,       irest  = 1,
 ntf    = 2,       ntc    = 2,       tol    = 0.0000005,
 ntb    = 1,       ntp    = 0,       temp0 = 300.0, tautp = 1.0,
nstlim = 1,    dt = 0.002,  maxcyc = 5,      imin = 0,  ntmin = 2,
 ntpr   = 1,     ntwx = 0, ntt = 0, ntr = 0,
          jfastw = 0, nmrmax=0, ntave = 25,
/
&debugf
 do_debugf = 1,do_dir = 1,do_adj = 1,do_rec = 1, do_self = 1,
 do_bond = 0,do_angle = 0,do_ephi = 0, zerovdw = 1, zerochg = 0,
 chkvir = 0,
 dumpfrc = 0,
 rmsfrc = 1,
/
&ewald
 ew_coeff=0.35,
/
```

Note that an Ewald coefficient of 0.35 is close to the default error for an 8 Angstrom cutoff. However, the first job used an 11 Angstrom cutoff. The direct sum forces calculated in the 2nd job are compared to these, giving the RMS error due to an 8 Angstrom cutoff, with this value of ew_coeff. The reciprocal sum error calculated in the 2nd job is with respect to the pme reciprocal

forces in the 1st job considered as "exact".

Note further that if in these two jobs you had not specified "ew_coeff" *sander* would have calculated ew_coeff according to the cutoff and the direct sum tolerance, defaulted to 10^-5. This would give 2 different ewald coefficients. Under these circumstances the direct, reciprocal and adjust energies and forces would not agree well between the two jobs. However the total energy and forces should agree reasonably, (forces to within about 5x10^-4 relative RMS force error) Since the totals are invariant to the coefficient.

Finally, note that if other force components are calculated, such as van der Waals, bond, angle etc. The total force will include these, and the relative RMS force errors will be with respect to this total force in the denominator.

6. PMEMD (Particle Mesh Ewald Molecular Dynamics)

6.1. Introduction.

PMEMD is a new version of Sander that has been written with the major goal of improving performance in Particle Mesh Ewald molecular dynamics simulations and minimizations. PMEMD supports a subset of the functionality present in Sander, with the focus being on Particle Mesh Ewald molecular dynamics and minimization. For the supported functionality, the input required and output produced are intended to exactly replicate Sander 8 within the limits of roundoff errors. PMEMD just runs more rapidly, scales better on parallel processors using MPI, can be used profitably on significantly higher numbers of processors, and uses about half the resident memory. Dynamic memory allocation is used so memory configuration is not required. PMEMD is ideal for molecular dynamics simulations of large solvated systems for long periods of time, especially if supercomputer resources are available.

PMEMD accepts Amber 8 Sander input files (prmtop, inpcrd, restrt, mdin), and is also backward compatible in regard to input to the same extent that Sander 8 is. As is the case with Sander 8, some of the more obscure mdin options used in earlier versions of Sander have been dropped and use of these may produce namelist errors at runtime. All options documented in the Sander section of this manual should be properly parsed. Support for previous PMEMD modes invoked with the options amber7_compat and use_cit has been dropped.

6.2. Functionality

As mentioned above, PMEMD is not a complete implementation of Sander 8. Instead, it is intended to be a fast implementation of the functionality most likely to be used by someone doing simulations on large solvated systems.

The following functionality is missing entirely:

1	*igb != 0*	Only explicit solvent models are supported; other models such as Generalized Born that are specified with a nonzero value of igb are not supported.
2	*ipol != 0*	Polarizable force field simulations are not currently supported.
3	*Extra Points*	PRMTOP files containing extra points data are not supported, and the frameon option should not be specified in mdin with any value.
4	*nmropt = 2*	A variety of NMR-specific options such as NOESY restraints, chemical shift restraints, pseudocontact restraints, and direct dipolar coupling restraints are not supported.
5	*imin = 5*	Trajectory analysis is not supported.
6	*idecomp != 0*	Energy decomposition options, used in conjunction with mm_pbsa, are not supported.
7	*itgtmd != 0*	Targeted molecular dynamics is not supported.
8	*ntmin > 2*	LMOD minimization methods are not supported.
9	*icfe != 0*	Calculation of free energies via thermodynamic integration is not supported.
10	*Water Caps*	Water cap simulations are not supported, since such simulations are nonperiodic.
11	*LES*	A LES-supporting version of PMEMD is not available.
12	*debugf namelist*	Use of the debugf namelist and options it contains is not supported. This functionality is nice for developers but not very useful for production.
13		The new format for specifying frozen or restrained atoms, which uses the restraint_wt, restraintmask, and bellymask options, is not supported. This functionality is still supported through use of the Amber 6/7 GROUP format instead.

The following options are supported, but only with the indicated default values:

1	*ew_type = 0*	Only Particle Mesh Ewald calculations are supported. ew_type = 1 (regular Ewald calculations) must be done in Sander 8.
2	*eedmeth = 1*	Only a cubic spline switch function (eedmeth = 1) for the direct sum Coulomb interaction is supported. This is the default, and most widely used setting for eedmeth.
3	*order = 4*	Only the default order of 4 for the PME B-spline interpolation is supported.
4	*cutoff + skinnb >= 6.d0*	In PMEMD an assumption is made that all nonbonded force adjustments will be found within a distance of cutoff + skinnb. To insure that this is a safe assumption, a minimum length check on cutoff + skinnb is made.

I would strongly suggest that new PMEMD users simply take an existing Sander 8 mdin file and attempt a short 10-30 step run. The output will tell you whether or not PMEMD will handle the particular problem at hand.

6.3. New variables

A minimum of new variables have been introduced into the mdin namelists in PMEMD. The new variables are:

use_axis_opt In &ewald. For parallel runs, the most favorable orientation of an orthogonal unit cell is with the longest side in the Z direction. Starting with pmemd 3.00, we were actually reorienting internal coordinates to take advantage of this, and in high processor count runs on oblong unit cells, using axis optimization can improve performance on the order of 10%. However, if a system has hotspots, the results produced with axes oriented differently may vary by on the order of 0.05% relatively quickly. This effect has to do with the fact that axis optimization changes the order of LOTS of operations and also the fft slab layout, and under mpi if the system has serious hotspots, shake will come up with slightly different coordinate sets. This is really only a problem in pathological situations, and then it is probably mostly telling you that the situation is pathological, and neither set of results is more correct (typically the ewald error term is also high). In routine regression testing with over a dozen tests, axis reorientation has no effect on results. Nonetheless, we have changed defaults recently to be in favor of higher reproducibility of results. Now, axis optimization is only done for mpi runs in which an orthogonal unit cell has an aspect ratio of at least 3 to 2. It is turned off for all minimization runs and for runs in which velocities are randomized (ntt = 2 or 3). If you want to force axis optimization, you may set use_axis_opt = 1 in the &ewald namelist. If you set it to 0, you will force it off in scenarios where it would otherwise be used.

mdinfo_flush_interval

In &ctrl, this variable can be used to control the minimum time in integer seconds

between "flushes" of the mdinfo file. PMEMD DOES NOT use file flush() calls at all because flush functionality is broken in some versions of the SGI compilers/libraries and in some versions of the Intel Fortran Compiler (its actually worse - SGI changed the flush() interface, and depending on your compilers/libraries version, a call to flush() may corrupt the stack). Thus, PMEMD does an open/close cycle on mdinfo at a default minimum interval of 60 seconds. This interval can be changed with this variable if desired. Note that mdinfo under PMEMD simply serves as a heartbeat for the simulation at mdinfo_flush_interval, and mdinfo probably will not be updated with the last step data at the end of a run. If mdinfo_flush_interval is set to 0, then both mdinfo and mdin will be reopened and closed at each step.

dbg_atom_redistribution

In &ctrl, the default value (0) of this variable does nothing. If dbg_atom_redistribution is set to 1, then the atom redistribution code basically runs at every list build, and also causes the image redistribution code to run frequently. This capability is provided for testing purposes, since this code may not normally run for several thousand steps, well past the point where roundoff errors make it impossible to spot problems that have a small impact on results. If you have questions about results, you may want to enable this functionality and do a short run, looking for problems in the first 100 steps. Under no circumstances should this switch be set routinely, as it has a fairly negative impact on performance. It is only relevant for parallel runs. We have tested thoroughly for problems related to this code, but include the switch as a user assurance feature.

Slightly changed functionality:

An I/O optimization has been introduced into PMEMD. The NTWR default value (frequency of writing the restart file) has been modified such that the default minimum is 500 steps, and this value is increased incrementally for multiprocessor runs. In general, frequent writes of restrt, especially in runs with a high processor count, is wasteful. Also, if the mden file is being written, it is always written as formatted output, regardless of the value of IOUTFM. Sander 6 was broken in that it would die attempting to do a binary write of mden. There is little or no reason to ever want to write out mden as a binary file.

6.4. Installation

The build process for PMEMD is similar to the build process for the rest of Amber 8, but must be invoked separately in the src/pmemd directory. There is a configure script that generates a config.h header that is used in the build process. Generation of config.h files is dependendent on use of information in the src/pmemd/config_data database. This system is similar to the old Amber MACHINE files configuration system, but is a bit more automated in that the configure script will set up config.h for a lot of common machine setups. The PMEMD installation process has remained separate from the Amber 8 installation process because PMEMD does not support all systems that can be automatically configured by Amber 8, and vice versa. Also, there is an emphasis on performance in PMEMD, and there was a desire to be able to fine tune the optimization process to a larger extent than was possible with the Amber 8 configuration process. Finally, user definition of configuration files in the src/config_data database is a fairly simple process, and this allows users to easily target new machines or machines with unusual configuration requirements. For more PMEMD installation details, please read src/pmemd/README.

6.5. Acknowledgements

This code was developed by Dr. Robert Duke in Prof. Lee Pedersen's Lab at UNC-Chapel Hill, starting from the version of Sander in Amber 6. I would like to thank Prof. Pedersen for his support in the development of this code, and would also like to acknowledge funding support from NIH grant HL-06350 (PPG) and NSF grant 2001-0759-02 (ITR/AP). I would also like to acknowledge Drs. Lalith Perera and Divi Venkateswarlu in the Pedersen Lab for helpful conversations and a willingness to actually use early releases of PMEMD, as well as Dr. Tom Darden of NIEHS for helpful conversations. This work required the availability of large piles of processors. The North Carolina Supercomputer Center was instrumental in making the work possible by making available a 720 processor IBM SP3 for my research efforts. The folks at the Edinburgh Parallel Computing Centre have also been most helpful and generous in providing me access to their HPCx facility (a total of 1280 IBM SP4 processors) for performance testing. Thanks also to Vance Shaffer of IBM who put me in touch with the EPCC folks and also did some benchmarking on other IBM systems. The folks at the Pittsburgh Supercomputing Center have also been most generous in making several systems available for test and benchmarking.

When citing PMEMD (Particle Mesh Ewald Molecular Dynamics) in the literature, please use the Amber Version 8 citation given in the Amber 8 manual.

7. LES

The LES functionality for sander and gibbs was written by Carlos Simmerling, based on his thesis work and the experiences of the Elber group. It basically functions by modifying the *prmtop* file using the program `addles`. The modified *prmtop* file is then used with a slightly modified version of sander called `sander.LES`.

Background material for LES can be found in Chapter 12.

7.1. Preparing to use LES with AMBER

The first decision that must be made is whether LES is an appropriate technique for the system that you are studying. For further guidance, you may wish to consult published articles to see where LES has proven useful in the past. Several examples will also be given at the end of this section in order to provide models that you may wish to follow.

There are three main issues to consider before running the ADDLES module of AMBER.

(1) What should be copied?

(2) How many copies should be used?

(3) How many regions should be defined?

A brief summary of my experience with LES follows.

(1) You should make copies of flexible regions of interest. This sounds obvious, and in some cases it is. If you are interested in determining the conformation of a protein loop, copy the loop region. If you need to determine the position of a side chain in a protein after a single point mutation, copy that side chain. If the entire biomolecule needs refinement, then copy the entire molecule. Some other cases may not be obvious- you may need to decide how far away from a particular site structural changes may propagate, and how far to extend the LES region.

(2) You should use as few copies as are necessary. While this doesn't sound useful, it illustrates the general point--too few copies and you won't get the full advantages of LES, and too many will not only increase your system size unnecessarily but will also flatten the energy surface to the point where minima are no longer well defined and a wide variety of structures become populated. In addition, remember that LES is an approximation, and more copies make it more approximate. Luckily, published articles that explore the sensitivity of the results to the number of copies show that 3-10 copies are usually reasonable and provide similar results, with 5 copies being a good place to start.

(3) Placing the divisions between regions can be the most difficult choice when using LES. This is essentially a compromise between surface smoothing and copy independence. The most effective surface-smoothing in LES takes places between LES regions. This is because Na copies in region A interact with all Nb copies in region B, resulting in Na*Nb interactions, with each scaled by 1/(Na*Nb) compared to the original interaction. This is better both from the statistics of how many different versions of this interaction contribute to the LES average, and how much the barriers are reduced. Remember that since the copies of a given region do not interact with different copies of that same region, interactions inside a region are only scaled by 1/N.

The other thing to consider is whether these enhanced statistics are actually helpful. For example, if the copies cannot move apart, you will obtain many copies of the same conformation--obviously not very helpful. This will also result in less effective reduction in barriers, since the average energy barriers will be very similar to the non-average barrier. The independence of the copies is also related to how the copies are attached. For example, different copies of an

amino acid side chain are free to rotate independently (at least within restrictions imposed by the surroundings and intrinsic potential) and therefore each side chain in the sequence could be placed into a separate LES region. If you are interested in backbone motion, however, placing each amino acid into a separate region is not the best choice. Each copy of a given amino acid will be bonded to the neighbor residues on each side. This restriction means that the copies are not very independent, since the endpoints for each copy need to be in nearly the same places. A better choice is to use regions of 2-4 amino acids. As the regions get larger, each copy can start to have more variety in conformation- for example, one segment may have some copies in a helical conformation while others are more strand-like or turn-like. The general rule is that larger regions are more independent, though you need to consider what types of motions you expect to see.

The best way to approach the division of the atoms that you wish to copy into regions is to make sure that you have several LES regions (unless you are copying a very small region such as a short loop or a small ligand). This will ensure plenty of inter-copy averaging. Larger regions permit wider variations in structure, but result in less surface smoothing. A subtle point should be addressed here- the statistical improvement available with LES is not a benefit in all cases and care must be taken in the choice of regions. For example, consider a ligand exiting a protein cavity in which a side chain acts as a *gate* and needs to move before the ligand can escape. If we make multiple copies of the gate, and do not copy the ligand, the ligand will interact in an average way with the *gates*. If the gate was so large that even the softer copies can block the exit, then the ligand would have to wait until ALL of the gate copies opened in order to exit. This may be more statistically difficult than waiting for the original, single gate to open despite the reduced barriers. Another way to envision this is to consider the ligand trying to escape against a true probability distribution of the gate- if it was open 50% of the time and closed 50%, then the exit may still be completely blocked. Continuum representations are therefore not always the best choice.

Specific examples will be given later to illustrate how these decisions can be made for a particular system.

7.2. Using the ADDLES program

The ADDLES module of AMBER is used to prepare input for simulations using LES. A non-LES prmtop and prmcrd file are generated using a program such as LEaP. This prmtop file is then given to ADDLES and replaced by a new prmtop file corresponding to the LES system. All residues are left intact- copies of atoms are placed in the same residue as the original atom, so that analysis based on sequence is preserved. Atom numbering is changed, but atom names are unchanged, meaning that a given residue may have several atoms with the same name. A different program is available for taking this new topology file and splitting the copies apart into separate residues, if desired. All copies are given the same coordinates as in the input coordinate file for the non-LES system.

Using addles:

```
addles < inputfile > outputfile
```

SAMPLE INPUT FILE:

```
~ a line beginning with ~ is a comment line.
~ all commands are 4 letters.
~ the maximum line length is 80 characters;
~ a trailing hyphen, "-", is the line continuation token.
```

```
~
~ use 'file' to specify an input/output file
~ then the type of file
~ 'rprm' means this is the file to read the prmtop
~ the 'read' means it is an input file
~
file rprm name=(solv200.topo) read
~
~ 'rcrd' reads the original coordinates- optional, only if you want
~ a set of coords for the new topology
~ you can also use 'rcvd' for coords+velocities, 'rcvb' for coords,
~ velos and box dimensions
~
file rcrd name=(501v200.coords) read
~
~ 'wprm' is the new topology file to be written. the 'wovr' means to
~ write over the file if it exists, 'writ' means don't write over.
~
file wprm name=(lesparm) wovr
~
~'wcrd is for writing coords, it will automatically write velo and box
~ if they were read in by 'rcvd' or 'rcvb'
~
file wcrd name=(lescrd) wovr
~
~ now put 'action' before creating the subspaces
~
action
~
~ the default behavior is to scale masses by 1/N.
~ omas leaves all masses at the original values
~
omas
~
~ now we specify LES subspaces using the 'spac' keyword, followed
~ by the number of copies to make and then a pick command to tell which
~ atom to copy for this subspace
~ 3 copies of the fragment consisting of monomers 1 and 2
~
spac numc=3 pick #mon 1 2 done
~
~ 3 copies of the fragment consisting of monomers 3 and 4
~
spac numc=3 pick #mon 3 4 done
~
~ 3 copies of the fragment consisting of residues 5 and 6
~
spac numc=3 pick #mon 5 6 done
~
```

```
~ 2 copies of the side chain on residue 1
~ note that this replaces each of the side chains ON EACH OF THE 3
~ COPIES MADE ABOVE with 2 copies - net 6 copies
~ each of the 3 copies of residue 1-2 has 2 side chain copies.
~ the '#sid' command picks all atoms in the residue except
~ C,O,CA,HA,N,H and HN.
~
spac numc=2 pick #sid 1 1 done
spac numc=2 pick *sid 2 2 done
spac numc=2 pick #sid 3 3 done
spac numc=2 pick #sid 4 4 done
spac numc=2 pick #sid 5 5 done
~
use the *EOD to end the input
*EOD
```

What this does: all of the force constants are scaled in the new prmtop file by 1/N for N copies, so that this scaling does not need to be done for each pair during the nonbond calculation. Charges and VDW epsilon values are also scaled. New bond, angle, torsion and atom types are created. Any of the original types that were not used are discarded. Since each LES copy should not interact with other copies of the SAME subspace, the other copies are placed in the exclusion list. If you define very large LES regions, the exclusion list will get large and you may have trouble with the fixed length for this entry in the prmtop file- currently 8 digits.

The coordinates are simply copied - that means that all of the LES copies initially occupy the same positions in space. In this setup, the potential energy should be identical to the original system- this is a good test to make sure everything is functioning properly. Do a single energy evaluation of the LES system and the original system, using the copied coordinate file. All terms should be nearly identical (to within machine precision and roundoff). With PME on non- neutral systems, all charges are slightly modified to neutralize the system. For LES, there are a different number of atoms than in the original system, and therefore this charge modification to each atom will differ from the non-LES system and electrostatic energies will not match perfectly.

IMPORTANT: After creating the LES system, the copies will all feel the same forces, and since the coordinates are identical, they will move together unless the initial velocities are different. If you are initializing velocities using INIT=3 and TEMPI>0, this is not a problem. In order to circumvent this problem, addles slightly (and randomly) modifies the copy velocities if they were read from the coordinate input file. If the keyword "nomodv" is specified, the program will leave all of the velocities in the same values as the original file. If you do not read velocities, make sure to assign an initial non-zero temperature to the system. You should think about this and change the behavior to suit your needs. In addition, the program scales the velocities by sqrt(N) for N copies to maintain the correct thermal energy (˜mv2), but only when the masses are scaled (not using omas option). Again, this requires some thought and you may want different behavior. Regardless of what options are used for the velocities, further equilibration should be carried out. These options are simple attempts to keep the system close to the original state [101].

It is important to understand that each subsequent pick command acts on the ORIGINAL particle numbers. Making a copy of a given atom number also makes copies of all copies of that atom that were already created. This was the simplest way to be able to have a hierarchical LES setup, but you can't make extra copies of part of one of the copies already made. I'm not sure why you would want to, or if it is even correct to do so, but you should be warned. Copies can be

anything -spanning residues, copies of fragments already copied, non-contiguous fragments, etc. Pay attention to the order in which you make the copies, and look carefully at the output to make sure you get what you had in mind. Addles will provide a list at the end of all atoms, the original parent atom, and how many copies were made.

There are array size limits in the file SIZE.h, I apologize in advance for the poor documentation on these. Mail *carlos.simmerling@stonybrook.edu* if you have any questions or problems.

7.3. More information on the ADDLES commands and options

```
file: open a file, also use one of
rcrd:  read coords from this file
rcvd:  read coords + velo from file
rcvb:  read coords, velo and box from file
wcrd:  write coords (and more if rcvd, rcvb) to file
wprm:  write new topology file

action:     start run, all of the following options must come AFTER action

nomodv:     do NOT slightly randomize the velocities of the copies

spac:   add a new subspace definition, using a pick command (see below).
            follow with numc=#  pickcmd where # is the number of copies to make
            and pickcmd is a pick command that selects the group of atoms
            to copy.

omas:   leave all masses at original values (otherwise scale 1/N)
```

Syntax for 'pick' commands

Currently, the syntax for picking atoms is somewhat limited. Simple Boolean logic is followed, but operations are carried out in order and parentheses are not allowed.

```
#prt A B         picks the atom range from A to B by atom number
#mon A B         picks the residue range from A to B by residue number
#cca A B         picks the residue range from A to B by residue number,
                     but dividing the residue between CA and C; the CO for A
                     is included, and the CO for monomer B is not.  See
                     Simmerling and Elber, 1994 for an example of where this
                     can be useful.
chem prtc A      picks all atoms named A, case sensitive
chem mono A      picks all residues named A, case sensitive
```

Completion wildcards are acceptable for names: H* picks H, HA, etc. Note that H*2 will select all atoms starting with H and ignore the 2.

Boolean logic:

```
|   or              atoms in either group are selected
```

```
&    and              atoms must be in both groups to be selected
!=   not              A != B will pick all atoms in A that are NOT in B
```

The user should carefully check the output file to ensure that the proper atoms were selected.

Examples:

```
pick commandatoms selected

pick #mon 4 19 done                        all atoms in residues 4 through 19
pick #mon 1 50 & chem mono GLY done        only GLY in residues 1 to 50
pick chem mono LYS | chem mono GLU done    any GLU or LYS residue
pick #mon 1 5 != #prt 1 3 done             residues 1 to 5 but not atoms 1 to 3
```

so, a full command to add a new subspace (LES region) with 4 copies of atoms 15 to 35 is:

```
spac numc=4 pick #prt 15 35 done
```

7.4. Using the new topology/coordinate files with SANDER

These topology files are ready to use in Sander with one exception: all of the FF parameters have been scaled by 1/N for N copies. This is done to provide the energy of the new system as an average of the energies of the individual copies (note that it is an average energy or force, not the energy or force from an average copy coordinate). However, one additional correction is required for interactions between pairs of atoms in the same LES region. Sander will make these corrections for you, and this information is just to explain what is being done. For example, consider a system where you make 2 copies of a sidechain in a protein. Each charge is scaled by 1/2. For these atoms interacting with the rest of the system, each interaction is scaled by 1/2 and there are 2 such interactions. For a pair of particles inside the sub-space, however, the interaction is scaled by 1/2*1/2=1/4, and since the copies do not interact, there are only 2 such interactions and the sum does not correspond to the correct average. Therefore, the interaction must be scaled up by a factor of N. When the PME technique is requested, this simple scaling cannot be used since the entire charge set is used in the construction of the PME grid and individual charges are not used in the reciprocal space calculation. Therefore, the intra-copy energies and forces are corrected in a separate step for PME calculations. Sander will print out the number of correction interactions that need to be calculated, and very large amounts of these will make the calculation run more slowly. PME also needs to do a separate correction calculation for excluded atom pairs (atoms that should not have a nonbonded interaction, such as those that are connected by a bond). Large LES regions result in large numbers of excluded atoms, and these will result in a larger computational penalty for LES compared to non-LES simulations. For both of these reasons, it is more efficient computationally to use smaller LES regions- but see the discussion above for how region size affects simulation efficiency. These changes are included in the LES version of Sander (sander.LES). Each particle is assigned a LES 'type' (each new set of copies is a new type), and for each pair of types there is a scaling factor for the nonbond interactions between LES particles of those types. Most of the scaling factors are 1.0, but some are not - such as the diagonal terms which correspond to interactions inside a given subspace, and also off-diagonal terms where only some of the copies are in common. An example of this type is the side chain example given

above- each of the 3 backbone copies has 2 sidechains, and while interactions inside the side chains need a factor of 6, interactions between the side chain and backbone need a factor of 3. This matrix of scaling factors is stored in the new topology file, along with the type for each atom, and the number of types. The changes made in sander relate to reading and using these scale factors.

7.5. Using LES with the Generalized Born solvation model

LES simulations can be performed using the GB solvent model, with some limitations. It is strongly recommended that the user read the article describing the derivation of the GB+LES approach [102]. The current code only allows igb=1 when using LES. Surface area calculations are not yet supported with LES. Only a single LES region is permitted for GB+LES simulations. A new namelist variable was introduced (RDT) in sander to control the compromise of speed and accuracy for GB+LES simulations. The article referenced above provides more detail on the function of this variable. RDT is the effective radii deviation threshold. When using GB+LES, non-LES atoms require multiple effective Born radii for an exact calculation. Using these multiple radii can significantly increase calculation time required for GB calculations. When the difference between the multiple radii for a non-LES atom is less than RDT, only a single effective radius will be used. A value of 0.01 has been found to provide a reasonable compromise between speed and accuracy, and is the default value.

7.6. Case studies: Examples of application of LES

7.6.1. Enhanced sampling for individual functional groups: Glucose.

The first example will deal with enhancing sampling for small parts of a molecule, such as individual functional groups or protein side chains. In this case we wanted to carry out separate simulations of α and β (not converting between anomers, only for conversions involving rotations about bonds) glucose, but the 5 hydroxyl groups and the strong hydrogen bonds between neighboring hydroxyls make conversion between different rotamers slow relative to affordable simulation times. The eventual goal was to carry out free energy simulations converting between anomers, but we need to ensure that each window during the Gibbs calculation would be able to sample all relevant orientations of hydroxyl groups in their proper Boltzmann-weighted populations. We were initially unsure how many different types of structures should be populated and carried out non-LES simulations starting from different conformations. We found that transitions between different conformations were separated by several hundred picoseconds, far too long to expect converged populations during each window of the free energy calculation. We therefore decided to enhance conformational sampling for each hydroxyl group by making 5 copies of each hydroxyl hydrogen and also 5 copies of the entire hydroxymethyl group. Since the hydroxyl rotamer for each copy should be relatively independent, we decided to place each group in a different LES region. This meant that each hydroxyl copy interacted with all copies of the neighboring groups, with a total of 5*5*5*5*5 or 3125 structural combinations contributing to the LES average energy at each point in time. The input file is given below.

```
file rprm name=(parm.solv.top) read
file rcvb name=(glucose.solv.equ.crd) read
file wprm name=(les.prmtop) wovr
file wcrd name=(glucose.les.crd) wovr
```

```
action
~
omas
~
~ 5 copies of each hydroxyl hydrogen- copying oxygen will make no difference
~ since they will not be able to move significantly apart anyway
~
spac numc=5 pick chem prtc HO1 done
spac numc=5 pick chem prtc HO2 done
spac numc=5 pick chem prtc HO3 done
spac numc=5 pick chem prtc HO4 done
~
~ take the entire hydroxy methyl group
~
spac numc=5 pick #prt 20 24 done
*EOD
```

This worked quite well, with transitions now occurring every few ps and populations that were essentially independent of initial conformation [103].

7.6.2. Enhanced sampling for a small region: Application of LES to a nucleic acid loop

In this example, we consider a biomolecule (in this case a single RNA strand) for which part of the structure is reliable and another part is potentially less accurate. This can be the case in a number of different modeling situations, such as with homologous proteins or when the experimental data is incomplete. In this case two different structures were available for the same RNA sequence. While both structures were hairpins with a tetraloop, the loop conformations differed, and one was more accurate. We tested whether MD would be able to show that one structure was not stable and would convert to the other on an affordable timescale.

Standard MD simulations of several ns were not able to undergo any conversion between these two structures (the initial structure was always retained). Since the stem portion of the RNA was considered to be accurate, LES was only applied to the tetraloop region. In this case, both of the ends of the LES region would be attached to the same locations in space, and there was no concern about copies diffusing too far apart to re-converge to the same positions after optimization. The issues that need to be addressed once again are the number of copies to use, and how to place the LES region(s). I usually start with the simplest choices and used 5 LES copies and only a single LES region consisting of the entire loop. If each half of the loop was copied, then it might become too *crowded* with copies near the base-pair hydrogen bonds and conformational changes that required moving a base through this regions could become even more difficult (see the background section for details). Therefore, one region was chosen, and the RNA stem, counterions and solvent were not copied. The ADDLES input file is given below.

```
~
file rprm name=(prm.top) read
file rcvb name=(rna.crd) read
file wprm name=(les.parm) wovr
```

```
file wcrd name=(les.crd) wovr
action
~
omas
~
~ copy the UUCG loop region- residues 5 to 8.
~ pick by atom number, though #mon 5 8 would work the same way
~
spac numc=5 pick #prt 131 255  done
*EOD
```

Subsequent LES simulations were able to reproducibly convert from what was known to be the incorrect structure to the correct one, and stay in the correct structure in simulations that started there. Different numbers of LES copies as well as slightly changing the size of the LES region (from 4 residues to 6, extending 1 residue beyond the loop on either side) were not found to affect the results. Fewer copies still converted between structures, but on a slower timescale, consistent with the barrier heights being reduced roughly proportional to the number of copies used. See Simmerling, Miller and Kollman, 1998, for further details.

7.6.3. Improving conformational sampling in a small peptide

In this example, we were interested not just in improving sampling of small functional groups or even individual atoms, but in the entire structure of a peptide. The peptide sequence is AVPA, with ACE and NME terminal groups. Copying just the side chains might be helpful, but would not dramatically reduce the barriers to backbone conformational changes, especially in this case with so little conformational variety inherent in the Ala and Pro residues. We therefore apply LES to all atoms. If we copied the entire peptide in 1 LES regions, the copies could float apart. While this would not be a disaster, it would make it difficult to bring all of the copies back together if we were searching for the global energy minimum, as described above. We therefore use more than one LES region, and need to decide where to place the boundaries between regions. A useful rule of thumb is that regions should be at least two amino acids in size, so we pick our two regions as Ace-Ala-Val and Pro-Ala- Nme. If we make five LES copies of each region and each copy does not interact with other copies of the same regions, each half the peptide will be represented by five potentially different conformations at each point in time. In addition, since each copy interacts with all copies of the rest of the system, there are 25 different combinations of the two halves of the peptide that contribute at each point in time. This statistical improvement alone is valuable, but the corresponding barriers are also reduced by approximately the same factors. When we place the peptide in a solvent box the solvent interacts in an average way with each of the copies. The input file is given below, and all of the related files can be found in the test directory for LES.

```
~
~ all file names are specified at the beginning, before "action"
~
~ specify input prmtop
~
file rprm name=(prmtop) read
~
```

```
~ specify input coordinates, velocities and box (this is a restart file)
~
file rcvb name=(md.solv.crd) read
~
~ specify LES prmtop
~
file wprm name=(LES.prmtop) wovr
~
~ specify LES coordinates (and velocities and box since they
~ were input)
~
file wcrd name=(LES.crd) wovr
~
~ now the action command reads the files and tells addles to
~ process commands
~
action
~
~ do not scale masses of copied particles
~
omas
~
~ divide the peptide into 2 regions.
~ use the CCA option to place the division between carbonyl and
~ alpha carbon
~ use the "or" to make sure all atoms in the terminal residues
~ are included since the CCA option places the region division at C/CA
~ and we want all of the terminal residue included on each end
~
~ make 5 copies of each half
~
~ "spac" defines a LES subspace (or region)
~
spac numc=5 pick #cca 1 3 | #mon 1 1 done
~
spac numc=5 pick #cca 4 6 | #mon 6 6 done
~
~ the following line is required at the end
*EOD
```

This example brings up several important questions:

(1) should I make LES copies before or after adding solvent? Since LEaP is used to add solvent, and LEaP will not be able to load and understand a LES structure, you must run ADDLES after you have solvated the peptide in LEaP. ADDLES should be the last step before running SANDER.

(2) which structure should be used as input to ADDLES? If you will also be carrying out non-LES simulations, then you can equilibrate the non-LES simulation and carry out any amount of production simulation desired before taking the structure and running

ADDLES. At the point you may switch to only LES simulations, or continue both LES and non-LES from the same point (using different versions of SANDER). Typically I equilibrate my system without LES to ensure that it has initial stability and that everything looks OK, then switch to LES afterward. This way I separate any potential problems from incorrect LES setup from those arising from problems with the non-LES setup, such as in initial coordinates, LEaP setup, solvent box dimensions and equilibration protocols.

(3) how can I analyze the resulting LES simulation? This is probably the most difficult part of using LES. With all of the extra atoms, most programs will have difficulty. For example, a given amino acid with LES will have multiple phi and psi backbone dihedral angles. There are basically two options: first, you can process your trajectory such that you obtain a single structure (non-LES). This might be just extracting one of the copies, or it might be one by taking the average of the LES copies. After that, you can proceed to traditional analysis but must keep in mind that the average structure may be non-physical and may not represent any actual structure being sampled by the copies, especially if they move apart significantly. A better way is to use LES-friendly analysis tools, such as those developed in the group of Carlos Simmerling. The visualization program MOIL-View (*http://morita.chem.sunysb.edu/~carlos/moil-view.html*) is one example of these programs, and has many analysis tools that are fully LES compatible. Read the program web page or manual for more details. A version of MOIL-View is included on the Amber 8 CD.

7.7. Unresolved issues with LES in AMBER

(1) Sander can't currently maintain groups of particles at different temperatures (important for dynamics, less so for optimization.) [104,105] Users can set *temp0les* to maintain all LES atoms at a temperature that is different from that for the system as a whole, but all LES atoms are then coupled to the same bath.

(2) Initial velocity issues as mentioned above- works properly, user must be careful.

(3) Analysis programs may not be compatible. See *http://morita.chem.sunysb.edu/~carlos/moil- view.html* for an LES-friendly analysis and visualization program.

(4) Visualization can be difficult, especially with programs that use distance-based algorithms to determine bonds. See #3 above.

(5) Water should not be copied- the fast water routines have not been modified. For most users this won't matter.

(6) Copies should not span different 'molecules' for pressure coupling and periodic imaging issues. Copies of an entire 'molecule' should result in the copies being placed in new, separate molecules- currently this is not done. This would include copying things such as counterions and entire protein or nucleic acid chains.

(7) Copies are placed into the same residue as the original atoms- this can make some residues much larger than others, and may result in less efficient parallelization with algorithms that assign nonbond workload based on residue numbers.

8. ptraj

The current version of *ptraj* is really two programs:

rdparm: a program to read, print (and modify) Amber prmtop files
usage: `rdparm` *prmtop*

ptraj: a program to process coordinates/trajectories
usage: `ptraj` *prmtop script*

Which code is used at runtime depends on the name of the executable (note that both *rdparm* and *ptraj* are created by default from the same source code when the programs are compiled with the supplied Makefile). If the executable name contains the string "rdparm", then the *rdparm* functionality is obtained. *rdparm* is semi-interactive (type `?` or `help` for a list of commands) and requires specification of an Amber prmtop file (this *prmtop* is specified as a filename typed on the command line; note that if no filename is specified you will be prompted for a filename).

If the executable name does not contain the string "rdparm", *ptraj* is run instead. *ptraj* also requires specification of parameter/topology information, however it currently supports both the Amber prmtop format and (I know, sacrilege!) CHARMM psf files. Note that the *ptraj* program can also be accessed from *rdparm* by typing *ptraj*.

The commands to *ptraj* can either be piped in through standard input or supplied in a file, where the filename (*script*) is passed in as the second command line argument. Note that if the *prmtop* filename is absent, the user will be prompted for a filename.

The code is written in ANSI compliant C and is fairly extensively documented and meant to be extended by able users!. Along with this code is distributed public domain C code from the Computer Graphics Lab at UCSF for reading and writing PDB files. Note that this program is updated more frequently than the general Amber release and that new versions and documentation may be obtained through links on the Amber WWW page.

Usage: `ptraj` *prmtop script*

ptraj is a program to process and analyze sets of 3-D coordinates read in from a series of input coordinate files (in various formats as discussed below). For each coordinate set read in, a sequence of events or *ACTIONS* is performed (in the order specified) on each of the configurations (set of coordinates) read in. After processing all the configurations, a trajectory file and other supplementary data can be optionally written out.

To use the program it is necessary to (1) read in a parameter/topology file, (2) set up a list of input coordinate files, (3) optionally specify an output file and (4) specify a series of actions to be performed on each coordinate set read in.

(1) **reading in a parameter/topology file**:

This is done at startup and currently either an Amber prmtop or CHARMM psf file can be read in. The type of the file is autodetected. The information in these files is used to setup the global *STATE* (ptrajState *) which gives information about the number of atoms, residues, atom names, residue names, residue boundaries, *etc*. This information is used to guide the reading of input coordinates which MUST match the order specified by the state, otherwise garbage may be obtained (although this may be detected by the program for some file

formats, leading to a warning to the user). In other words, when reading a pdb file, the atom order must correspond exactly to that of the parameter/topology information; in the pdb the names/residues are ignored and only the coordinates are read in based.

(2) **set up a list of input coordinate files**:

This is done with the `trajin` command (described in more detail below) which specifies the name of a coordinate file and optionally the start, stop and offset for reading coordinates. The type of coordinate file is detected automatically and currently the following input coordinate types are supported:

- Amber trajectory
- Amber restart (or inpcrd)
- PDB
- CHARMM (binary) trajectory
- Scripps "binpos" binary trajectory

(3) **optionally specify an output trajectory file**:

This is done with the `trajout` command (discussed in more detail below). Trajectories can currently be written in Amber trajectory (default), Amber restrt, Scripps binpos, PDB or CHARMM trajectory (in little or big endian binary format).

(4) **specify a list of actions**:

There are a variety of coordinate analysis/manipulation **actions** provided and each of these specified-- note that each action can be repeated-- is applied sequentially to the coordinates in the order listed by the user in the input file.

As mentioned above, input to `ptraj` is in the form of commands listed in a script (or if absent, from text on standard input). An example input file to `ptraj` follows:

```
trajin traj1.Z 1 20
trajin traj2.Z 1 100
trajin restrt.Z
trajout fixed.traj
rms first out rms @CA,C,N
center :1-20
image
strip :WAT
go
```

This reads in three files of coordinates (which can be compressed and the type is autodetected), a trajectory file is output (by default to Amber trajectory format), rms fitting is performed to the first coordinate frame using atoms names CA, C and N (storing the RMSd values to a file named "rms"), the center of geometry of residues 1-20 is placed at the origin, the coordinates are imaged (which requires periodic boundary conditions) to move coordinates outside the periodic box back in, and then the coordinates of all the residues named "WAT" are deleted.

8.1. ptraj command prerequisites

Before going into the details of each of the commands, some prerequisites are necessary to describe the command flow and the standard argument types. Effectively, all the commands are

processed from the input file in the order listed, except for the input/output commands. Input is the first step and involves reading in all the coordinates sets from each file specified, in the order specified, a single coordinate set at a time. For each coordinate set read in, all of the actions specified are applied and then the potentially modified coordinates are output. Not all of the actions actually modify the coordinates and some of the commands simply change the state (such as **solvent** which just changes the definition of what the solvent molecules are). Some of the actions just accumulate data (such as distances, angles and sugar puckers). Writing out of any accumulated data is deferred until all of the coordinate sets have been read in. Some of the actions load up contiguous sets of coordinates into main memory; with large coordinate sets this may require large amounts of memory. In these cases, such as with the command **2dRMS**, it may be useful only to "save" the necessary coordinates by performing a **strip** of unnecessary coordinates prior to the **2dRMS** call.

In the discussion that follows commands are listed in **bold** type. Words in *italics* are values that need to be specified by the user, and words in standard text are keywords to specify an option (which may or may not be followed by a value). In the specification of the commands, arguments in square brackets (`[]`'s) are optional and the "`|`" character represents "or". Arguments that are not in square brackets are required. In general, if there is an error in processing a particular action, that action will be ignored and the user warned (rather than terminating the program), so check the printed WARNING's carefully... In what follows is listed a few standard argument types:

mask: this is an atom or residue mask; it represents the list of active atoms. The current parser recognizes a simplified midas style format for picking atoms and residues. The "`@`" character represents an atom selection and the "`:`" character represents a residue selection. Either the atom and residue names or numbers can be specified. The "`-`" character represents a continuation. The "`~`" represents "not" and in this naive implementation, if this character is specified anywhere in the string, the "not" flag will be turned on. The "`*`" character is a wild card and will match all the atoms if specified alone. When specified in atom or residue name specifications, sometimes it will correctly work as a wildcard. The "`?`" character is also a wildcard, however only one character is matched. Note that the current parser is not very sophisticated. Until this is "fixed", check the output very carefully; note that whenever an atom mask is used, a summary of the atoms selected is printed, so check this out...

filename: this refers to the full path to a file and note that no checking is done for existing files, i.e. data will be overwritten if you attempt to write to an existing file.

8.2. ptraj input/output commands

trajin *filename* [*start stop offset*]

Load the trajectory file specified by *filename* using only the frames starting with *start* (default 1) and ending with (and including) *stop* (default, the final configuration) using an offset of *offset* (default 1) if specified. Amber trajectory, restrt/inpcrd, PDB, Scripps BINPOS and CHARMM binary trajectory files are all currently supported and the type of file is auto-detected (including the CHARMM binary file byte ordering). Compressed files (filenames with an appended `.Z` or `.gz` are also recognized and treated appropriately). Note that the coordinates *must* match the names/ordering of the parameter/topology information previously read in.

reference *filename*

Load up a the first coordinate set from the trajectory specified by the file named *filename* and save this for use as a reference structure. Currently only the **rms** command potentially uses this reference structure. Note that as the state is modified (for example by **strip** or **closestwaters**), the reference coordinates are also modified internally.

trajout *filename* [*format*] [nobox] [little | big] [dumpq | parse] [nowrap]

Specify the name of the file of output coordinates to write (*filename*) and the format (*format*). Currently supported formats are "trajectory" (or Amber trajectory, the default), "restart" (Amber restart), "binpos" (Scripps binary format), "pdb" (PDB), or "charmm" (CHARMM binary trajectory). With the CHARMM files, it is possible to specify the byte ordering as "little" or "big" endian, with the default being that which the first CHARMM trajectory file was read in as, or if none was read in, big endian. With the PDB output, it is possible to include charges and radii in higher precision temperature/occupancy columns with the additional keyword "dumpq" (to dump Amber charges and radii, assuming a Amber prmtop has been previously read in) or "parse" (to dump charges and parse radii). By default (and differing from earlier versions of `ptraj`), atom names are wrapped in the PDB file to put the 4th letter of the atom name first. If you want to avoid this behavior, specify "nowrap"; the former is more consistent with standard PDB usage but departs from the format written in previous versions of this program. Note that if more than one coordinate set is to be output, with the pdb and restrt/inpcrd formats, extensions (based on the current configuration number) will be appended to the filenames and therefore only one coordinate set will be written per file. The optional keyword "nobox" will prevent box coordinates from being dumped to Amber trajectory files; this is useful if one is stripping the solvent from a trajectory file and you don't want that pesky box information cluttering up the trajectory and messing with other programs... Note that if periodic box information is present in the CHARMM trajectory file, when a new CHARMM trajectory file is written (in versions > 22) the symmetric box information will be *very* slightly different due to numerical issues in the diagonalization procedure; this will not effect analysis but shows up if diffing the binary files.

8.3. ptraj commands that modify the state

These commands change the state of the system, such as to delete atoms.

box [x *value*] [y *value*] [z *value*] [alpha *value*] [beta *value*] [gamma *value*]
[fixx] [fixy] [fixz] [fixalpha] [fixbeta] +[fixgamma]

This command allows specification and optionally fixing of the periodic box (unit cell) dimensions. This can be useful when reading PDB files that do not contain box information. In the standard usage, without the "fix*N*" keywords, if the box information is not already present in the input trajectory (such as the case with restart files or trajectory files) this command can be used to set the default values that will be applied. If you want to force a particular box size or shape, the "fixx", "fixy", etc

commands can be used to override any box information already present in the input coordinate files.

solvent [byres | bytype | byname] *mask1* [*mask2*] [*mask3*] ...

This command can be used to override the solvent information specified in the Amber prmtop file or that which is set by default (based on residue name) upon reading a CHARMM psf. Applying this command overwrites any previously set solvent definitions. The solvent can be selected by residue with the "byres" modifier using all the residues specified in the one or more atom masks listed. The byname option searches for solvent by residue name (where the mask contains the name of the residue), searching over all residues. The "bytype" option is intended for use in selecting solvents that span multiple residues, however it is not yet implemented since I haven't found a case where it is necessary (and setting the solvent information in the code is a real nightmare).

As an example, say you want to select the solvent to be all residues from 20-100, then you would do

```
solvent byres :20-100
```

Note that if you don't know the final residue number of your system offhand, yet you do know that the solvent spans all residues starting at residue 20 until the end of the system, just chose an upper bound and the program will reset accordingly, *i.e.*

```
solvent byres :20-9999
```

To select all residues named "WAT" and "TIP3" and "ST2":

```
solvent byname WAT TIP3 ST2
```

Note that if you just want to peruse what the current solvent information is (or more generally get some information about the current state), specify **solvent** with no arguments and a summary of the current state will be printed.

Other commands which also modify the state are **strip** and **closestwaters**. These commands are described in the next section since they also modify the coordinates.

8.4. **ptraj** *action* commands

The following are commands that involve an *action* performed on each coordinate set as it is read in. The commands are listed in alphabetical order. Note that in the script the commands are applied in the order specified and some may change the overall state (more on this later). All of the actions can be applied repeatedly. Note that in general (except where otherwise mentioned) imaging in non-orthorhombic systems is now supported, however note that this code has not been extensively tested.

angle *name mask1 mask2 mask3* [out *filename*] [time *interval*]

Calculate the angle between the three atoms listed, each specified in a mask, *mask1* through *mask3*. If more than one atom is listed in each mask, then the center of mass of the atoms in that mask is used at the position. The results are saved internally with the name *name* (which must be unique) on the `scalarStack` for later processing (with the **analyze** command). Data will be dumped to a file named *filename* if "out" is specified (with a time interval between configurations of *interval* if "time" is listed). All the angles are stored in degrees.

atomicfluct [out *filename*] [*mask*] [start *start*] [stop *stop*] [offset *offset*]
[byres | byatom | bymask] [bfactor]

Compute the atomic positional fluctuations for all the atoms; output is performed only for the atoms in *mask*. If "byatom" is specified, dump the calculated fluctuations by atom (default). If "byres" is specified, dump the average (mass-weighted) for each residue. If "bymask" is specified, dump the average (mass-weighted) over all the atoms in the original *mask*. If "out" is specified, the data will be dumped to *filename* (otherwise the values will be dumped to the standard output). The "start", "stop" and "offset" keywords can be used to specify the range of coordinates processed (as a subset of all of those read in across all input files, not to be confused with the individual specification in each **trajin** command). If the keyword "bfactor" is specified, the data is output as B-factors rather than atomic positional fluctuations (which simply means multiplying the results by (8/3)pi**2).

So, to dump the mass-weighted B-factors for the protein backbone atoms, by residue:

```
atomicfluct out back.apf @C,CA,N byres bfactor
```

average *filename* [*mask*] [start *start*] [stop *stop*] [offset *offset*] [pdb [parse | dumpq]
[nowrap] | binpos | rest] [nobox] [stddev]

Compute the average structure over all the configurations read in (subject to start, stop and offset if set) dumping the results to a file named *filename*. If the keyword "stddev" is present, save the standard deviations (fluctuations) instead of the average coordinates. Output is by default to an Amber trajectory, however can also be to a pdb, binpos or restrt file (depending on the keyword chosen). The "nobox" keyword will suppress box coordinates, and with the PDB format, it is possible to dump charges and radii (with the "dumpq" keyword for Amber radii and charges or the "parse" for parse radii and Amber charges) and prevent atom name wrapping "nowrap". The optional *mask* trims the output coordinates (but does not change the state). This command does not alter the coordinates as they are processed. If you want to alter the coordinates by averaging (for use by actions further on), use the **runningaverage** command.

center [*mask*] [origin] [mass]

If we are in periodic boundary conditions, center all the atoms based on the center of geometry of the atoms in the *mask* to the center of the periodic box or the origin if the optional argument "origin" is specified. If the trajectory is not a periodic boundary trajectory, then the molecule is implicitly centered to the origin. If no *mask* is specified, centering is relative to all the atoms. If "mass" is specified, center with respect to the center of mass instead.

checkoverlap [*mask*] [min *value*] [max *value*] [noimage]

Look for pair distances in the selected atoms (all by default) that are less than the specified minimum value (in angstroms, 0.95 by default) apart or greater than the maximum value (if specified). This command is rather computationally demanding, particularly if imaging is turned on (by default), but it is extremely useful for diagnosing problems in input coordinates related to poor model building.

closest *total mask* [oxygen | first] [noimage]

Retain only *total* solvent molecules (using the solvent information specified, see **solvent** above) in each coordinate set. The solvent molecules saved are those which are closest to the atoms in the *mask*. If "oxygen" or "first" are specified, only the distance to the first atom in the solvent molecule (to each atom in the mask) is measured. This command is rather time consuming since many distances need to be measured. Note that imaging is implicitly performed on the distances and this gets extremely expensive in non-orthorhombic systems due to the need to possibly check all the distances of the nearest images (up to 26!). Imaging can be disabled by specifying the "noimage" keyword.

Note that the behavior of this command is slightly different than in previous `ptraj` versions; now the solvent molecules are ordered at output such that the closest solvent is first and the PDB file residue numbers no longer represent the identity of the water in the original coordinate set. This command should now work with non-sequential solvent molecules and be independent of where the water is located. Like the **strip** command, this modifies the current state (i.e. pars down the size of the trajectory which is useful in cases where subsets of a trajectory may be loaded into memory). A restriction of this command is that each of the solvent molecules must have the same number of atoms; this leads to a fixed size "configuration" in each coordinate set output which is necessary for most of the file formats and to avoid really complicating the code.

Of course, say you have two solvents of differing sizes and you want to perform closest to each of these, this can be done sequentially. Say we have both ethanol ":ETH" and water ":WAT" present, and you want to save the closest 50 of each to residues :1-20

```
solvent byres :WAT
closestwater 50 :1-20 first
```

```
solvent byres :ETH
closestwater 50 :1-20 first
```

dihedral *name mask1 mask2 mask3 mask4* [out *filename*]

Calculated the dihedral angle for the four atoms listed in *mask1* through *mask4* (representing rotation about the bond from *mask2* to *mask3*). If more than one atom is listed in each mask, treat the position of that atom as the center of mass of the atoms in the mask. The results are saved internally with the name *name* (which must be unique) and the data is stored on the `scalarStack` for later processing. Data will be dumped to a file if "out" is specified (with a *filename* appended). All the angles are listed in degrees.

diffusion *mask time_per_frame* [average] [*filenameroot*]

Compute a mean square displacement plot for the atoms in the *mask*. The time between frames in picoseconds is specified by *time_per_frame*. If "average" is specified, then the average mean square displacement is calculated and dumped (only). If "average" is not specified, then the average and individual mean squared displacements are dumped. They are all dumped to a file in the format appropriate for xmgr (dumped in multicolumn format if necessary, i.e. use xmgr -nxy). The units are displacements (in angstroms**2) vs time (in ps). The *filenameroot* is used as the root of the filename to be dumped. The average mean square displacements are dumped to "filenameroot_r.xmgr", the x, y and z mean square displacements to "filenameroot_x.xmgr", etc and the total distance traveled to "filenameroot_a.xmgr".

This will fail if a coordinate moves more than 1/2 the box in a single step. Also, this command implicitly unfolds the trajectory (in periodic boundary simulations) hence will currently only work with orthorhombic unit cells.

dipole *filename nx x_spacing ny y_spacing nz z_spacing mask1* origin | box [max *max_percent*]

Same as **grid** (see below) except that dipoles of the solvent molecules are binned. Dumping is to a grid in a format for Chris Bayly's discern delegate program that comes with Midas/Plus.

distance *name mask1 mask2* [out *filename*] [noimage]

This command will calculate a distance between the center of mass of the atoms in *mask1* to the center of mass of the atoms in *mask2* and store this information into an array with *name* as the identifier (a name which must be unique and which is placed on the `scalarStack` for later processing) for each frame in the trajectory. If the optional keyword "out" is specified, then the data is dumped to a file named *filename*. The distance is implicitly imaged (for both orthorhombic and non-orthorhombic unit cells) and the shortest imaged distance will be saved (unless the "noimage" keyword is specified which disables imaging).

tion]

grid *filename nx x_spacing ny y_spacing nz z_spacing mask1* [origin | box] [negative] [max *frac-*

Create a grid representing the histogram of atoms in *mask1* on the 3D grid that is "*nx* * *x_spacing* by *ny* * *y_spacing* by *nz* * *z_spacing* angstroms (cubed). Either "origin" or "box" can be specified and this states whether the grid is centered on the origin or half box. Note that to provide any meaningful representation of the density, the solute of interest (about which the atomic densities are binned) should be rms fit, centered and imaged prior to the **grid** call. If the optional keyword "negative" is also specified, then these density will be stored as negative numbers. Output is in the format of a XPLOR formatted contour file (which can be visualized by the density delegate to Midas/Plus or other programs). Upon dumping the file, pseudo-pdb HETATM records are also dumped to standard out which have the most probable grid entries (those that are 80% of the maximum by default which can be changed with the max keyword, i.e. max .5 makes the dumping at 50% of the maximum).

Note that as currently implemented, since the XPLOR grids are integer based, the grid is offset from the origin (towards the negative size) by half the grid spacing.

image [origin] [center] *mask* [bymol | byres | byatom | bymask] *mask*
[triclinic | familiar [com *mask*]]

Under periodic boundary conditions, which particular unit cell a given molecule is in does not matter as long as, as a whole, all the molecules "image" into a single unit cell. In an MD simulation, molecules drift over time and may span multiple periodic cells unless "imaging" is enabled to shift molecules that leave back into the primary unit cell. In sander, the IWRAP variable controls this, with IWRAP=1 implying turning on imaging. This command, **image** allows post-processing of the imaging to force all the molecules into the primary unit cell.

If the optional argument "origin" is specified, then imaging will occur to the coordinate origin (like in SPASMS) rather than the center of the box (as is the Amber standard). By default all atoms are imaged *by molecule* based on the position of the first atom (or the center of mass of the molecule if "center" is specified; the latter is recommended). If the *mask* is specified, only the atoms in the *mask* will be imaged. It is now possible to image by atom (byatom), by residue (byres), by molecule (bymol, default) or by atom mask (where all the atoms in the mask are treated as belonging to a single molecule). The behavior of the "by molecule" imaging is different in CHARMM and Amber; with Amber the molecules are specified directly by the periodic box information whereas with the CHARMM parameter/topology, each segid is treated as a different molecule. With this newer implementation of the imaging code, it is possible to avoid breaking up double stranded DNA during imaging, *i.e.*:

```
image :1-20 bymask :1-20
image byres :WAT
```

[Of course this assumes that the coordinates of the two strands were not displaced during the dynamics as well!] Imaging only makes sense if there is periodic box information present.

Non-orthorhombic unit cells are now supported! Use of the triclinic imaging can be forced with the "triclinic" keyword. Note that this puts the box into the triclinic shape, not the more familiar, more spherical shapes one might expect for some of the unit cells (i.e. truncated octahedron). To get into the more familiar shape, specify the "familiar" keyword. In this case, to specify atoms that imaged molecules should be closest to, specify a center of the atoms in the mask specified with the "com" keyword. Note that imaging "familiar" is time consuming (but recommended) since each of the possible imaged distances (27) are checked to see which is closest to the center.

principal *mask* [dorotation]

Principal axis transformation to align the atoms specified in *mask*. This is reasonably functional as there are still issues with degenerate eigenvalues and unwanted coordinate swapping. To align whole system along the principal axes specify "dorotation".

pucker *name mask1 mask2 mask3 mask4 mask5* [out *filename*] [amplitude]
[altona | cremer] [offset *offset*]

Calculate the pucker for the five atoms specified in each of the mask's, *mask1* through *mask5*, associating *name* (which must be unique) with the calculated values. If more than one atom is specified in a given mask, the center of mass of all the atoms in that mask is used to define the position. If the "out" keyword is specified the data is dumped to *filename*. If the keyword "amplitude" is present, the amplitudes are saved rather than the pseudorotation values. If the keyword "altona" is listed, use the Altona & Sundarlingam conventions/algorithm (for nucleic acids) (the default) [see Altona & Sundaralingam, *JACS* 94, 8205-8212 (1972) or Harvey & Prabhakaran, *JACS* 108, 6128-6136 (1986).] In this convention, both the pseudorotation phase and amplitude are in degrees.

If "cremer" is specified, use the Cremer & Pople conventions/algorithm [see Cremer & Pople, *JACS* 97:6, 1354-1358 (1975).]

Note that to calculate nucleic acid puckers, specify C1' first, followed by C2', C3', C4' and finally O4'. Also note that the Cremer & Pople convention is offset from the Altona & Sundarlingam convention (with nucleic acids) by 90.0; to add in an extra 90.0 to "cremer" (offset -90.0) or subtract 90.0 from the "Altona" (offset 90.0) specify an *offset* with the offset keyword; this value is subtracted from the calculated pseudorotation value (or amplitude).

radial *root-filename spacing maximum solvent-mask* [*solute-mask*] [closest]
[density *value*] [noimage]

Compute radial distribution functions and store the results into files with *root-filename* as the root of the filename. Three files are currently produced, "*root-filename*_carnal.xmgr" (which corresponds to a carnal style RDF), "*root-filename*_standard.xmgr" (which uses the more traditional RDF with a density input by the user)

and "*root-filename*_volume.xmgr" (which uses the more traditional RDF and the average volume of the system). The total number of bins for the histogram is determined by the spacing between bins (*spacing*) and the range which runs from zero to *maximum*. If only a *solvent-mask* is listed (i.e. a list of atoms) then the RDF will be calculated for the interaction of every solute-mask atom with ALL the other solute-mask atoms.

If the optional *solute-mask* is specified, then the RDF will represent the interaction of each solute-mask atom with ALL of the solvent-mask atoms. If the optional keyword "closest" is specified, then the histogram will bin, over all the solvent-mask atoms, the distance of the closest atom in the solute mask. If the *solute-mask* and *solvent-mask* atoms are not mutually exclusive, zero distances will be binned (although this should not break the code). If the optional keyword "density", followed by the density *value* is specified, this will be used in the calculations. The default value is 0.033456 molecules/angstrom**3 which corresponds to a density of water equal to 1.0 g/mL. To convert a standard density in g/mL, multiply the density by "6.022 / (10 * weight)" where "weight" is the mass of the molecule in atomic mass units. This will only effect the "*root-filename*_standard.xmgr" file.

Note that although imaging of distances is performed (to find the shortest imaged distance unless the "noimage" keyword is specified), minimum image conventions are applied.

rms *mode* [*mass*] [out *filename*] [time *interval*] *mask* [name *name*] [nofit]

This will RMS fit all the atoms in the *mask* based on the current *mode* which is:

previous: fit to previous frame
first: fit to the "start" frame of the first trajectory specified.
reference: fit to a the reference structure (which must have been previously read in)

If the keyword "mass" is specified, then a mass-weighted RMSd will be performed. If the keyword "out" is specified (followed immediately by a *filename*), the RMSd values will be dumped to a file. If you want to specify an time interval between frames (used only when dumping the RMSd vs time), this can be done with the "time" keyword. To save the calculated values for later processing, associate a name with the "name" keyword (where the chosen *name must be unique and the data will be stored on the* `scalarStack` for later processing. If the keyword "nofit" is specified, then the coordinates are not modified, just the RMSd values are calculated (and stored or output if the name or out keywords were specified).

strip *mask* Strip all atoms matching the atoms in *mask*. This changes the state of the system such that all commands (actions) following the strip (including output of the coordinates which is done last) are performed on the stripped coordinates (*i.e.* if you strip all the waters and then on a later action try to do something with the waters, you will have trouble since the waters are gone). A benefit of stripping, beyong paring down trajectories is with the data intensive commands that read entire sections of the

trajectory into memory; with the strip to retain only selected atoms, it is much less likely that you will blow memory.

translate *mask* [x *x-value*] [y *y-value*] [z *z-value*]

Move the coordinates for the atoms in the *mask* in each direction by the offset(s) specified.

truncoct *mask distance* [prmtop *filename*]

Create a truncated octahedron box with solvent stripped to a distance *distance* away from the atoms in the *mask*. Coordinates are output to the trajectory specified with the **trajout** command. *Note that this is a special command* and will only really make sense if a single coordinate set is processed (*i.e.* any prmtop written out will only correspond to the first configuration!) and commands after the **truncoct** will have undefined behavior since the state will not be consistent with the modified coordinates. It is intended only as an aid for creating truncated octahedron restrt files for running in Amber.

The "prmtop" keyword can be used to specify the writing of a new prmtop (to a file named *filename*; this prmtop is only consistent with the first set of coordinates written. Moreover, this command will only work with Amber prmtop files and assumes an Amber prmtop file has previously been read in (rather than a CHARMM PSF). This command also assumes that all the solvent is located contiguously at the end of the file and that the solvent information has previously been set (see the **solvent** command).

watershell *mask filename* [lower *lower* upper *upper*] [*solvent-mask*] [noimage]

This option will count the number of waters within a certain distance of the atoms in the *mask* in order to represent the first and second solvation shells. The output is a file into *filename* (appropriate for xmgr) which has, on each line, the frame number, number of waters in the first shell and number of waters in the second shell. If *lower* is specified, this represents the distance from the *mask* which represents the first solvation shell; if this is absent 3.4 angstroms is assumed. Likewise, *upper* represents the range of the second solvation shell and if absent is assumed to be 5.0 angstroms. The optional *solvent-mask* can be used to consider other atoms as the solvent; the default is ":WAT". Imaging on the distances is done implicitly unless the "noimage" keyword has been specified.

8.5. Correlation and fluctuation facility

The *ptraj* program now contains several related sets of commands to analyze correlations and fluctuations, both from trajectories and from normal modes. These items replace the *correlation* command in previous versions of *ptraj*, and also replace what used to be done in the *nmanal* program. Some examples of command sequences are given at the end of this section.

vector *name mask* [principal [x|y|z] | dipole | box | corrplane | ired *mask2* |
corr *mask2* | corrired *mask2*] [out *filename*] [order *order*] [modes *modesfile*] [beg *beg*] [end *end*] [npair *npair*]

This command will keep track of a vector value (and its origin) over the trajectory; the data can be referenced for later use based on the *name* (which must be unique among the vector specifications). "Ired" vectors, however, may only be used in connection with the command "matrix ired". If the optional keyword "out" is specified (not valid for "ired" vectors), the data will be dumped to the file named *filename*. The format is frame number, followed by the value of the vector, the reference point, and the reference point plus the vector. What kind of vector is stored depends on the keyword chosen.

principal [x | y | z]: store one of the principal axis vectors determined by diagonalization of the intertia matrix from the coordinates of the atoms specified by the *mask*. If none of x | y | z are specified, then the principal axis (i.e. the eigenvector associated with the largest eigenvalue) is stored. The eigenvector with the largest eigenvalue is "x" (i.e. the hardest axis to rotate around) and the eigenvector with the smallest eigenvalue is "z" and if one of x | y | z are specified, that eigenvector will be dumped. The reference point for the vector is the center of mass of the *mask* atoms.

dipole: store the dipole and center of mass of the atoms specified in the *mask*. The vector is not converted to appropriate units, nor is the value well-defined if the atoms in the mask are not overall charge neutral.

box: store the box coordinates of the trajectory. The reference point is the origin or (0.0, 0.0, 0.0).

ired *mask2*: This defines ired vectors necessary to compute an ired matrix (see matrix command). The vectors must be defined *prior* to the matrix command.

corrplane: This defines a vector perpendicular to the (least-squares best) plane through a series of atoms given in *mask*, for which a time correlation function can be calculated subsequently with the command "analyze timecorr ...". *order* specifies the order of the Legendre polynomial used (0 <= order <= 2). It defaults to 2.

corr *mask2*: This defines a vector between the center of mass of *mask* and the one of *mask2*, for which a time correlation function can be calculated subsequently with the command "analyze timecorr ...". *order* specifies the order of the Legendre polynomial used (0 <= order <= 2). It defaults to 2.

corrired *mask2*: This defines a vector between the center of mass of *mask* and the one of *mask2*, for which a time correlation function according to the Isotropic Reorientational Eigenmode Dynamics (ired) approach [106] can be calculated. *order* specifies the order of the Legendre polynomial used (0 <= order <= 2). It defaults to 2. To calculate this vector, ired modes need to be provided by *modesfile*. They can be calculated by the commands "matrix ired ...", followed by "analyze matrix ...". Only modes <beg> to <end> are considered. Default is beg = 1, end = 50. To obtain meaningful results, it is important that the vector definition agrees with the one used

for calculation of the ired matrix (there is no internal check for this). Along these lines, *npair* needs to be specified, which relates to the position of this definition in the sequence of ired (not corrired!) vectors used to obtain the ired matrix.

matrix dist | covar | mwcovar | distcovar | correl | idea | ired [name *name*] [order *order*] [*mask1*] [*mask2*] [out *filename*]" [start *start*] [stop *stop*] [offset *offset*] [byatom | byres | bymask] [mass]

Compute DISTance, COVARiance, Mass-Weighted COVARiance, CORRELation, DISTance-COVARiance, Isotropically Distributed Ensemble Analysis [107], or Isotropic Reorientational Eigenmode Dynamics [106] matrices. Results are output to *filename* if given. Be aware, matrix dimension will be of the order of N x M for dist, correl, idea, and ired, 3N x 3M for covar and mwcovar, and N(N-1) x N(N-1) / 4 for distcovar (with N being the number of atoms in *mask1* and M being the number of atoms either in *mask1* or *mask2*).

"byatom" dumps the results by atom (default). This is the sole option for covar, mwcovar, distcovar, idea, and ired. In the case of dist or correl, "byres" calculates an average for each residue and "bymask" dumps the average over all atoms in the mask(s). With "mass", mass-weighted averages will be computed.

In the case of ired, mask information must not be given. Instead, "ired vectors" need to be defined *prior* to the matrix command by using the vector command. Otherwise, if no mask is given, all atoms against all are used. If only *mask1* is given, a symmetric matrix is computed. In the case of distcovar and idea, only *mask1* (or none) may be given. In the case of distcovar, mwcovar, and correl, if *mask1* and *mask2* is given, on output *mask1* atoms are listed column-wise while *mask2* atoms are listed row-wise. The number of atoms covered by *mask1* must be >= the number of atoms covered by *mask2* (this is also checked in the function).

The matrix may be stored internally on the matrixStack with the name *name* for latter processing (with the "analyze matrix" command). Since at the moment this only involves diagonalization, storing is only available for (symmetric) matrices generated with *mask1* (or no mask or ired matrices).

The *start*, *stop*, and *offset* parameters can be used to specify the range of coordinates processed (as a subset of all of those read in across all input files).

The *order* parameter chooses the order of the Legendre polynomial used to calculate the ired matrix.

analyze matrix *matrixname* [out *filename*] [thermo] [vecs *vecs*] [reduce]

Diagonalizes the matrix *matrixname*, which has been generated and stored before by the matrix command. This is followed by Principal Component Analysis (in cartesian coordinate space in the case of a covariance matrix or in distance space in the case of a distance-covariance matrix), or Quasiharmonic Analysis (in the case of a mass-weighted covariance matrix). Diagonalization of distance, correlation, idea,

and ired matrices are also possible. Eigenvalues are given in cm^{-1} in the case of a mass-weighted covariance matrix and in the units of the matrix elements in all other cases. In the case of a mass-weighted covariance matrix, the eigenvectors are mass-weighted.

Results [average coordinates (in the case of covar, mwcovar, correl), average distances (in the case of distcovar), main diagonal elements (in the case of idea and ired), eigenvalues, eigenvectors] are output to *filename*. *vecs* determines, how many eigenvectors and eigenvalues are calculated. The value must be >= 1, except if the "thermo" flag is given (see below). In that case, setting vecs = 0 results in calculating all eigenvalues, but no eigenvectors. This option is mainly intended for saving memory in the case of thermodynamic calculations. "reduce" (only possible for covar, mwcovar, and distcovar) results in reduced eigenvectors [Abseher & Nilges, *J. Mol. Biol.* **279,** 911, (1998)]. They may be used to compare results from PCA in distance space with those from PCA in cartesian-coordinate space.

"thermo" calculates entropy, heat capacity, and internal energy from the structure of a molecule (average coordinates, see above) and its vibrational frequencies using standard statistical mechanical formulas for an ideal gas. This option is only available for mwcovar matrices.

analyze modes fluct|displ|corr stack *stackname* | file *filename*
[beg *beg*] [end *end*] [bose] [factor *factor*] [out *outfile*] [maskp *mask1 mask2* [...]]

Calculates rms fluctuations ("fluct"), displacements of cartesian coordinates along mode directions ("displ"), or dipole-dipole correlation functions ("corr") for modes obtained from principal component analyses (of covariance matrices) or quasiharmonic analyses (of mass-weighted covariance matrices). Thus, a possible series of commands would be "matrix covar|mwcovar ..." to generate the matrix, "analyze matrix ..." to calculate the modes, and, finally, "analyze modes ...". Modes can be taken either from an internal stack, identified by their name on the stack, *stackname*, or can be read from a file *filename*. Only modes *beg* to *end* are considered. Default for *beg* is 7 (which skips the first 6 zero-frequency modes in the case of a normal mode analysis); for *end* it is 50. If "bose" is given, quantum (Bose) statistics is used in populating the modes. By default, classical (Boltzmann) statistics is used. *factor* is used as multiplicative constant on the amplitude of displacement. Default is factor = 1. Results are written to *outfile*, if specified, otherwise to stdout. In the case of "corr", pairs of atom masks (*mask1*, *mask2*; each pair preceded by "maskp" and each mask defining only a single atom) have to be given that specify the atoms for which the correlation functions are desired.

analyze timecorr vec1 *vecname1* [vec2 *vecname2*] [tstep *tstep*] [tcorr *tcorr*]
[drct] [dplr] [norm] out *filename*

Calculates time correlation functions for vectors *vecname1* (*vecname2*) of type "corr" or "corrired", using a fast Fourier method. If two different vectors are specified, the cross-correlation function is calculated. If the *drct* keyword is given, a direct approach is used instead of the FFT approach. Note that this is less efficient

than the FFT route. If *dplr* is given, in addition to the P_l correlation function, also correlation functions $C_l \equiv < P_l/(r(t)^3 r(t+\tau)^3) >$ and $< 1/(r(t)^3 r(t+\tau)^3) >$ are output. If *norm* is given, all correlation functions are normalized, i.e. $P_l(t = 0) = C_l(t = 0) = 1/(r(t)^3 r(t)^3)(t = 0) = 1.0$. Results are written to *filename*. *tstep* specifies the time between snapshots (default: 1.0), and *tcorr* denotes the maximum time for which the correlations functions are to be computed (default: 10000.0).

projection modes *modesfile* out *outfile* [beg *beg*] [end *end*] [*mask*]
[start *start*] [stop *stop*] [offset *offset*]

Projects snapshots onto modes obtained by diagonalizing covariance or mass-weighted covariance matrices. The modes are read from *modesfile*. The results are written to *outfile*. Only modes *beg* to *end* are considered. Default values are beg = 1, end = 2. *mask* specifies the atoms that will be projected. The user has to make sure that these atoms agree with the ones used to calculate the modes (i.e., if mask1 = @CA was used in the "matrix" command, mask = @CA needs to be set here as well). The *start*, *stop*, and *offset* parameters can be used to specify the range of coordinates processed (as a subset of all of those read in across all input files).

8.6. Examples

Please note that in most cases the trajectory needs to be aligned against a reference structure to obtain meaningful results. Use the "rms" command for this.

Calculating and analyzing matrices and modes

As a simple example, a distance matrix of all CA atoms is generated and output to distmat.dat.

```
matrix dist @CA out distmat.dat
```

In the following, a mass-weighted covariance matrix of all atoms is generated and stored internally with the name mwcvmat (as well as output). Subsequently, the matrix is analyzed by performing a quasiharmonic analysis, whereby 5 eigenvectors and eigenvalues are calculated and output to evecs.dat.

```
matrix mwcovar name mwcvmat out mwcvmat.dat
analyze matrix mwcvmat out evecs.dat vecs 5
```

Alternatively, the eigenvectors can be stored internally and used for calculating rms fluctuations or displacements of cartesian coordinates.

```
analyze matrix mwcvmat name evecs vecs 5
analyze modes fluct out rmsfluct.dat stack evecs beg 1 end 3
analyze modes displ out resdispl.dat stack evecs beg 1 end 3
```

Finally, dipole-dipole correlation functions for modes obtained from principle component analysis or quasiharmonic analysis can be computed.

```
analyze modes corr out cffromvec.dat stack evecs beg 1 end 3 ...
                    ... maskp @1 @2 maskp @3 @4 maskp @5 @6
```

Projecting snapshots onto modes

After calculating modes, snapshots can be projected onto these in an additional "sweep" through the trajectory. Here, snapshots are projected onto modes 1 and 2 read from evecs.dat (which have been obtained by the "matrix mwcovar", "analyze matrix" commands from above).

```
projection modes evecs.dat out project.dat beg 1 end 2
```

Calculating time correlation functions

Vectors between atoms 5 and 6 as well as 7 and 8 are calculated below, for which auto and cross time correlation functions are obtained.

```
vector v0 @5 corr @6 order 2
vector v1 @7 corr @8 order 2
analyze timecorr vec1 v0 tstep 1.0 tcorr 100.0 out v0.out
analyze timecorr vec1 v1 tstep 1.0 tcorr 100.0 out v1.out
analyze timecorr vec1 v0 vec2 v1 tstep 1.0 tcorr 100.0 out v0_v1.out
```

Similarly, a vector perpendicular to the plane through atoms 18, 19, and 20 is obtained and further analyzed.

```
vector v2 @18,@19,@20 corrplane order 2
analyze timecorr vec1 v3 tstep 1.0 tcorr 100.0 out v2.out
```

For obtaining time correlation functions according to the ired approach, two "sweeps" through the trajectory are necessary. First, ired vectors are defined and an ired matrix is calculated and analyzed. Ired eigenvectors are output to ired.vec.

```
vector v0 @5 ired @6
vector v1 @7 ired @8
 ...
vector v5 @15 ired @16
vector v6 @17 ired @18
matrix ired name matired order 2
analyze matrix matired vecs 6 out ired.vec
```

In a subsequent ptraj run, ired time correlation functions are calculated by projecting the snapshots onto the ired eigenvectors (read from ired.vec), which results in corrired vectors. Then, time correlation functions are computed. Please note that it is important that the corrired vector definition agrees with the one used for calculation of the ired matrix.

```
vector v0 @5 corrired @6 order 2 modes ired.vec beg 1 end 6 npair 1
vector v1 @7 corrired @8 order 2 modes ired.vec beg 1 end 6 npair 2
 ...
vector v5 @15 corrired @16 order 2 modes ired.vec beg 1 end 6 npair 6
vector v6 @17 corrired @18 order 2 modes ired.vec beg 1 end 6 npair 7
analyze timecorr vec1 v0 tstep 1.0 tcorr 100.0 out v0.out
 ...
analyze timecorr vec1 v6 tstep 1.0 tcorr 100.0 out v6.out
```

8.7. Hydrogen bonding facility

The *ptraj* program now contains a generic facility for keeping track of lists of pair interactions (subject to a distance and angle cutoff) useful for calculation hydrogen bonding or other interactions. It is designed to be able to track the interactions between a list of hydrogen bond "donors" and hydrogen bond "acceptors" that the user specifies.

donor *resname atomname* | mask *mask* | clear | print

This command sets the list of hydrogen bond donors. It can be specified repeatedly to add to the list of potential donors. The usage is either as a pair of residue and atom names or as a specified atom mask. The former usage,

```
donor ADE N7
```

would set all atoms named N7 in residues named ADE to be potential donors.

```
donor mask :10@N7
```

would set the atom named N7 in residue 10 to be a potential donor.

The keyword "clear" will clear the list of donors specified so far and the keyword "print" will print the list of donors set so far.

The acceptor command is similar except that both the heavy atom and the hydrogen atom are specified. If the same atom is specified twice (as might be the case to probe ion interactions) then no angle is calculated between the donor and acceptor.

acceptor *resname atomname atomnameH* | mask *mask maskH* | clear | print

The **donor** and **acceptor** commands do not actually keep track of distances but instead simply set of the list of potential interactions. To actually keep track of the distances, the **hbond** command needs to be specified:

hbond [distance *value*] [angle *value*] [solventneighbor *value*]
[solventdonor *donor-spec*] [solventacceptor *acceptor-spec*]
[nosort] [time *value*] [print *value*] [series *name*]

The optional "distance" keyword specifies the cutoff distance for the pair interactions and the optional "angle" keyword specifies the angle cutoff for the hydrogen bond. The default is no angle cutoff and a distance of 3.5 angstroms. To keep track of potential hydrogen bond interactions where we don't care *which* molecule of a given type is interaction as long as one is (such as with water), the "solvent" keywords can be specified. An example would be keeping track of water or ions interacting with a particular donor or acceptor. The maximum number of possible interactions per a given donor or acceptor is specified with the "solventneighbor" keyword. The list of potential "solvent" donors/acceptors is specified with the solventdonor and solventacceptor keywords (with a format the same as the donor/acceptor keywords above).

As an example, if we want to keep track of water interactions with our list of donors/acceptors:

```
hbond distance 3.5 angle 120.0 solventneighbor 6 solventdonor WAT O
  solventacceptor WAT O H1 solventacceptor WAT O H2
```

If you wanted to keep track of interactions with Na+ ions (assuming the atom name was Na+ and residue name was also Na+):

```
hbond distance 3.5 angle 0.0 solventneighbor 6 solventdonor Na+ Na+
  solventacceptor Na+ Na+ Na+
```

To print out information related to the time series, such as maximum occupancy and lifetimes, specify the "series" keyword.

8.8. rdparm

`rdparm` requires an Amber prmtop file for operation and is menu driven. Rudimentary online help is available with the "?" command. Here is a sample run:

```
rdparm prmtop
Opened file prmtop with mode (r)
Read in residue labels...
 C5   C    A    A    C    G    T    T    G    G3
 ...
 CIO  CIO  CIO  CIO  CIO  CIO  CIO  CIO  CIO  CIO
 CIO  CIO  CIO  CIO  CIO  CIO  CIO  CIO  WAT  WAT
 WAT  WAT  WAT  WAT  WAT  WAT  WAT  WAT  WAT  WAT
 ...
 WAT
Scanning Box
Successfully completed readParm.

   MAIN MENU.  Please enter commands.  Use "?" or "help"
   for more info.  "exit" or "quit" to leave program...
```

```
MAIN MENU:  ?

   The following commands are currently available:

   help, ?
   atoms, printAtoms
   bonds, printBonds
   angles, printAngles
   dihedrals, printDihedrals
   pertbonds, perturbedBonds
   pertangles, perturbedAngles
   pertdihedrals, perturbedDihedrals
   printExcluded
   printLennardJones
   printTypes
   parmInfo
   checkcoords
   ddrive       <filename>
   delete       <bond || angle || dihedral> <number>
   delperturbed <bond || angle || dihedral> <number>
   restrain     <bond || angle || dihedral>
   openparm     <filename>
   writeparm    <filename>
   system       <string>
   mardi2sander <constraint file>
   analyze      <Amber trajectory || Amber coordinates>
   rms          <Amber trajectory>
   stripwater
   transform    <Amber trajectory>
   translateBox <Amber coords>
   modifyBoxInfo
   modifyMolInfo
   quit, exit
```

To create an executable, a Makefile is provided. This code was developed on SGI machines and has been tested on DEC Alpha and HPUX platforms. The code is broken up into numerous C files, each of which intends to be fairly independent. The main header file, "ptraj.h" includes all of the other header files, so each file need only include this local header file and other necessary global headers (such as <stdio.h>, <math.h>, etc). In order to hide function prototyping from each individual routine, for a given file, such as rdparm, a local define #define RDPARM_MODULE is defined which hides the prototype. See the header files for more information.

A basic summary of the code is as follows:

```
interface.c    --- code defining the basic user interface.  Note that
                   much of the underlying functionality is implemented
                   in dispatch.c.
utility.c      --- defines error routines, safe memory
                   allocation, etc.
```

```
rdparm.c       --- defines much of the functionality for reading and
                   processing Amber topologies/parameters.  User
                   access to many of these routines is accessed
                   indirectly through the ptraj interface with the
                   command rdparm or directly if the executable is
                   named rdparm.
ptraj.c        --- the main code for trajectory processing.

actions.c      --- the code implementing the ptraj "actions".

dispatch.c     --- defines the token lookup, and string stack processing,
                   and handles the conversion from strings typed by the
                   user to subroutines run.
io.c           --- input/output routines, file handling.  Note:
                   coordinate/trajectory IO is in trajectory.c.
trajectory.c   --- special input/output routines for
                   trajectories/coordinates.
rms.c          --- code to calculate root-mean-squared
                   deviations between conformations.
display.c      --- code for printing 2D RMS plots.
torsion.c      --- code for calculating torsions.
experimental.c --- new stuff.
pdb/<files>    --- code to read/write PDB files from the Computer Graphics
                   lab at UCSF.
main.c         --- the main routine.
```

? Print the help file. If an argument is given, help is printed for this command.

angles <mask>

Print all the angles in the file. If the <mask> is present, only print angles involving these atoms. For example, atoms :CYT@C? will print all angles involving atoms which have 2-letter names beginning with "C" from "CYT" residues.

atoms <mask>

Print all the atoms in the file. If the <mask> is present, only print these atoms.

bonds <mask>

Print all the bonds in the file. If the <mask> is present, only print bonds involving these atoms.

checkcoords <Amber trajectory>

Perform a rudimentary check of the coordinates from the filename specified. This is to look for obvious problems (such as overflow) and to count the number of frames.

dihedrals <mask>

Print all the dihedrals in the file. If the <mask> is present, only print dihedrals

involving one of these atoms.

ddrive <filename>
Create an input file for the SPASMS dihedral driver.

delete <bond || angle || dihedral> <number>
This command will delete a given bond, angle or dihedral angle based on the number specified from the current prmtop. The number specified should match that shown by the corresponding print command. Note that a new prmtop file is not actually saved. To do this, use the writeparm command. For example, "delete bond 5" will delete with 5th bond from the parameter/topology file.

delperturbed <bond || angle || dihedral> <number>
Same as delete above but to delete perturbed bonds, angles or dihedrals.

restrain <bond || angle || dihedral>
This is a means to add restraints as is possible with the "parm" program. Its usage is somewhat obsolete because more flexible restraints can be specified with the NMR functionality of sander. To use this command, specify whether the restraint is to a bond, angle or dihedral and the program will prompt for atom numbers (as specified in the "atom" or "printatom" command). As before, the prmtop is not actually saved until a "writeparm" command is issued.

openparm <filename>
Open up the prmtop file specified.

writeparm <filename>
Write a new prmtop file to "filename" based on the current (and perhaps modified) parameter/topology file.

system <string>
Execute the command "string" on the system.

mardi2sander <constraint file>
A rudimentary conversion of Mardigras style restraints to sander NMR restraint format.

rms <Amber trajectory>
Create a 2D RMSd plot in postscript or PlotMTV format using the trajectory specified. The user will be prompted for information. This command is rather slow and should be integrated into the "ptraj" code, however it hasn’t been yet.

stripwater This command will remove or add three point waters to a prmtop file that already has water. The user will be prompted for information. This is useful to take an existing prmtop and create another with a different amount of water. Of course, corresponding coordinates will also have to be built and this is not done by "rdparm".

To do this, ideally construct a PDB file and convert to Amber coordinate format using "ptraj".

ptraj <script-file>
This command reads a file or from standard input a series of commands to perform processing of trajectory files. See the supplemental documentation.

transform <Amber trajectory>
Perform rudimentary trajectory processing; this command is obsolete.

translateBox <Amber coords>
Translate the coordinates (only if they contain periodic box information) specified to place either at the origin (SPASMS format) or at half the box (Amber format).

modifyBoxInfo
This is a command to modify the box information, such as to change the box size. The changes are not saved until a writeparm command is issued.

modifyMolInfo
This command checks the molecule info (present with periodic box coordinates are specified) and points out problems if they exist. In particular, this is useful to overcome the deficiency in edit which places all the "add" waters into a single molecule.

parmInfo Print out information about the current prmtop file.

pertbonds, perturbedBonds
Print out the perturbed bonds.

pertangles, perturbedAngles
Print out the perturbed angles.

pertdihedrals, perturbedDihedrals
Print out the perturbed dihedrals.

printAngles Same as "angles".

printAtoms Same as "atoms".

printBonds Same as "bonds".

printDihedrals
Same as "dihedrals".

printExcluded
Print the excluded atom list.

printLennardJones
Print out the Lennard-Jones parameters.

printTypes Print out the atom types.

quit Quit the program.

9. MM-PBSA

The MM_PBSA approach represents the postprocessing method to evaluate free energies of binding or to calculate absolute free energies of molecules in solution. The sets of structures are usually collected with molecular dynamics or Monte Carlo methods. However, the collections of structures should be stored in the format of an AMBER trajectory file. The MM_PBSA/GBSA method combines the molecular mechanical energies with the continuum solvent approaches. The molecular mechanical energies are determined with the *sander* program from AMBER and represent the internal energy (bond, angle and dihedral), and van der Waals and electrostatic interactions. An infinite cutoff for all interactions is used. The electrostatic contribution to the solvation free energy is calculated with the Poisson-Boltzmann (PB) method, for example, as implemented in the *DelPhi* program [108], or in the pbsa program of the AMBER suite, or by generalized Born (GB) methods implemented in *sander.* The hydrophobic contribution to the solvation free energy is determined with solvent-accessible-surface-area-dependent terms [62]. The surface area is computed with Paul Beroza's *molsurf* program, which is based on analytical ideas primarily developed by Mike Connolly [109]. Estimates of conformational entropies can be made with the *nmode* module from AMBER.

Although the basic ideas here have many precedents, the first application of this model in its present form was to the A- and B-forms of RNA and DNA, where many details of the basic method are given [110]. You may also wish to refer to a review summarizing many of the initial applications of this model [111], as well as to papers describing more recent applications [112-116].

The initial MM_PBSA scripts were written by Irina Massova. These were later modified and mostly turned into Perl scripts by Holger Gohlke, who also added GB/SA (generalized Born/surface area) options, and techniques to decompose energies into pairwise contributions from groups (where possible).

9.1. General instructions

The general procedure is to edit the *mm_pbsa.in* file (see below), and then to run the code as follows:

```
mm_pbsa.pl  mm_pbsa.in > mm_pbsa.log
```

The *mm_pbsa.in* file refers to "receptor", "ligand" and "complex", but the chemical nature of these is up to the user, and these could equally well be referred to as "A", "B", and "AB". The procedure can also be used to estimate the free energy of a single species, and this is usually considered to be the "receptor".

The user also needs to prepare *prmtop* files for receptor, ligand, and complex using LEaP; if you are just doing "stability" calculations, only one of the *prmtop* files is required.

The output files are labeled ".out", and the most useful summaries are in the "statistics.out" files. These give averages and standard deviations for various quantities, using the following labeling scheme:

```
*** Abbreviations for mm_pbsa output ***
```

```
ELE   - non-bonded electrostatic energy + 1,4-electrostatic energy
VDW   - non-bonded van der Waals energy + 1,4-van der Waals energy
INT   - bond, angle, dihedral energies
GAS   - ELE + VDW + INT

PBSUR - hydrophobic contrib. to solv. free energy for PB calculations
PBCAL - reaction field energy calculated by PB
PBSOL - PBSUR + PBCAL
PBELE - PBCAL + ELE
PBTOT - PBSOL + GAS

GBSUR - hydrophobic contrib. to solv. free energy for GB calculations
GB    - reaction field energy calculated by GB
GBSOL - GBSUR + GB
GBELE - GB + ELE
GBTOT - GBSOL + GAS

TSTRA - translational entropy (as calculated by nmode) times temperature
TSROT - rotational entropy (as calculated by nmode) times temperature
TSVIB - vibrational entropy (as calculated by nmode) times temperature


*** Prefixes in front of abbreviations for energy decomposition ***

"T" - energy part due to _T_otal residue
"S" - energy part due to _S_idechain atoms
"B" - energy part due to _B_ackbone atoms
```

The *$AMBERHOME/src/mm_pbsa/Examples* directory shows examples of running a "Stability" calculation (*i.e.*, estimating the free energy of one species), a "Binding" calculation (estimating ΔG for A + B $\rightarrow$ AB), an "Nmode" calculation (to estimate entropies), and two examples of how total energies can be decomposed (either by residue, or pair-wise by residue). You should study the inputs and outputs in these directories to see how the program is typically used.

9.2. Preparing the input file

Below is a prototype *mm_pbsa.in* file; items in boldface would typically vary from run to run.

```
#
# Input parameters for mm_pbsa.pl
#
# Holger Gohlke
# 08.01.2002
#
###############################################################################
```

```
@GENERAL
#
# General parameters
#   0: means NO; >0: means YES
#
#   mm_pbsa allows to calculate (absolute) free energies for one molecular
#     species or a free energy difference according to:
#
#     Receptor + Ligand = Complex,
#     DeltaG = G(Complex) - G(Receptor) - G(Ligand).
#
#   PREFIX - To the prefix, "{_com, _rec, _lig}.crd.Number" is added during
#            generation of snapshots as well as during mm_pbsa calculations.
#   PATH - Specifies the location where to store or get snapshots.
#
#   COMPLEX - Set to 1 if free energy difference is calculated.
#   RECEPTOR - Set to 1 if either (absolute) free energy or free energy
#              difference are calculated.
#   LIGAND - Set to 1 if free energy difference is calculated.
#
#   COMPT - parmtop file for the complex (not necessary for option GC).
#   RECPT - parmtop file for the receptor (not necessary for option GC).
#   LIGPT - parmtop file for the ligand (not necessary for option GC).
#
#   GC - Snapshots are generated from trajectories (see below).
#   AS - Residues are mutated during generation of snapshots from trajectories.
#   DC - Decompose the free energies into individual contributions
#        (only works with MM and GB).
#
#   MM - Calculation of gas phase energies using sander.
#   GB - Calculation of desolvation free energies using the GB models in sander
#        (see below).
#   PB - Calculation of desolvation free energies by using a PB method (see below).
#   MS - Calculation of nonpolar contributions to desolvation using molsurf
#        (see below).
#        If MS == 0, nonpolar contributions are calculated with the LCPO method
#        in sander.
#   NM - Calculation of entropies with nmode.
#
PREFIX                snapshot
PATH                  ./
#
COMPLEX               1
RECEPTOR              1
LIGAND                1
#
COMPT                 ./parm_com.top
RECPT                 ./parm_rec.top
LIGPT                 ./parm_lig.top
```

```
#
GC                    0
AS                    0
DC                    0
#
MM                    1
GB                    1
PB                    1
MS                    1
#
NM                    0
#
###############################################################################
@DECOMP
#
# Energy decomposition parameters  (this section is only relevant if DC = 1 above)
#
#   Energy decomposition is performed for gasphase energies, desolvation free
#     energies calculated with GB, and nonpolar contributions to desolvation
#     using the LCPO method.
#   For amino acids, decomposition is also performed with respect to backbone
#     and sidechain atoms.
#
#   DCTYPE - Values of 1 or 2 yield a decomposition on a per-residue basis,
#            values of 3 or 4 yield a decomposition on a pairwise per-residue
#               basis. For the latter, so far the number of pairs must not
#               exceed the number of residues in the molecule considered.
#            Values 1 or 3 add 1-4 interactions to bond contributions.
#            Values 2 or 4 add 1-4 interactions to either electrostatic or vdW
#              contributions.
#
#   COMREC - Residues belonging to the receptor molecule IN THE COMPLEX.
#   COMLIG - Residues belonging to the ligand molecule IN THE COMPLEX.
#   RECRES - Residues in the receptor molecule.
#   LIGRES - Residues in the ligand molecule.
#   {COM,REC,LIG}PRI - Residues considered for output.
#   {REC,LIG}MAP - Residues in the complex which are equivalent to the residues
#                  in the receptor molecule or the ligand molecule.
#
DCTYPE                2
#
COMREC                1-166 254-255
COMLIG                167-253
COMPRI                1-255
RECRES                1-168
RECPRI                1-168
RECMAP                1-166 254-255
LIGRES                1-87
LIGPRI                1-87
```

LIGMAP **167-253**

```
################################################################################
@PB
#
# PB parameters (this section is only relevant if PB = 1 above)
#
#   The following parameters are passed to the PB solver.
#   Additional parameters (e.g. SALT in the case of delphi, ISTRNG in the
#     case of pbsa) may be added here.
#   For further details see the delphi and pbsa documentation.
#
#   PROC - Determines which method is used for solving the PB equation:
#          If PROC = 1, the delphi program is applied. If PROC = 2,
#           the pbsa program of the AMBER suite is used.
#   REFE - Determines which reference state is taken for PB calc:
#          If REFE = 0, reaction field energy is calculated with EXDI/INDI.
#            Here, INDI must agree with DIELC from MM part.
#          If REFE > 0 && INDI > 1.0, the difference of total energies for
#            combinations EXDI,INDI and 1.0,INDI is calculated.
#            The electrostatic contribution is NOT taken from sander here.
#   INDI - Dielectric constant for the solute.
#   EXDI - Dielectric constant for the surrounding solvent.
#   SCALE - Lattice spacing in no. of grids per Angstrom.
#   LINIT - No. of iterations with linear PB equation.
#   PRBRAD - Solvent probe radius in A (e.g. use 1.4 with the PARSE parameter
#     set and 1.6 with the radii optimized by R. Luo)
#
#   Parameters for pbsa only
#
#   RADIOPT - Option to set up atomic avity radii for molecular surface
#     calculation and dielectric assignment. A value of 0 uses the cavity
#     radii from the prmtop file. A value of 1 sets up optimized cavity radii
#     at the pbsa initialization phase. The latter radii are optimized for
#     model compounds of proteins only; use cautions when applying these radii
#     to nucleic acids.
#
#   Parameters for delphi only
#
#   FOCUS - If FOCUS > 0, subsequent (multiple) PERFIL and SCALE parameters are
#     used for multiple delphi calculations using the focusing technique.
#     The # of _focusing_ delphi calculations thus equals the value of FOCUS.
#   PERFIL - Percentage of the lattice that the largest linear dimension of the
#            molecule will fill.
#   CHARGE - Name of the charge file.
#   SIZE - Name of the size (radii) file.
#
#   SURFTEN / SURFOFF - Values used to compute the nonpolar contribution Gnp to
#                  the desolvation according to Gnp = SURFTEN * SASA + SURFOFF.
#
```

```
#
PROC                    2
REFE                    0
INDI                    1.0
EXDI                    80.0
SCALE                   2
LINIT                   1000
PRBRAD                  1.6
#
RADIOPT                 1
#
FOCUS                   0
PERFIL                  80.0
CHARGE                  ./my_amber94_delphi.crg
SIZE                    ./my_parse_delphi.siz
#
SURFTEN                 0.005
SURFOFF                 0.0
#
##############################################################################
@MM
#
# MM parameters (this section is only relevant if MM = 1 above)
#
#   The following parameters are passed to sander.
#   For further details see the sander documentation.
#
#   DIELC - Dielectricity constant for electrostatic interactions.
#           Note: This is not related to GB calculations.
#
DIELC                   1.0
#
##############################################################################
@GB
#
# GB parameters (this section is only relevant if GB = 1 above)
#
#   The first group of the following parameters are passed to sander.
#   For further details see the sander documentation.
#
#   IGB - Switches between Tsui's GB (1) and Onufriev's GB (2, 5).
#   GBSA - Switches between LCPO (1) and ICOSA (2) method for SASA calc.
#          Decomposition only works with ICOSA.
#   SALTCON - Concentration (in M) of 1-1 mobile counterions in solution.
#   EXTDIEL - Dielectricity constant for the surrounding solvent.
#   INTDIEL - Dielectricity constant for the solute.
#
#   SURFTEN / SURFOFF - Values used to compute the nonpolar contribution Gnp to
#                   the desolvation according to Gnp = SURFTEN * SASA + SURFOFF.
```

```
#
IGB                     2
GBSA                    1
SALTCON                 0.00
EXTDIEL                 80.0
INTDIEL                 1.0
#
SURFTEN                 0.0072
SURFOFF                 0.00
#
##############################################################################
@MS
#
# Molsurf parameters (this section is only relevant if MS = 1 above)
#
#   PROBE - Radius of the probe sphere used to calculate the SAS.
#           Since Bondi radii are already augmented by 1.4A, PROBE should be 0.0
#
PROBE                   0.0
#
##############################################################################
@NM
#
# Parameters for sander/nmode calculation (this section is only relevant
#                                          if NM = 1 above)
#
#   The following parameters are passed to sander (for minimization) and nmode
#     (for entropy calculation using gasphase statistical mechanics).
#   For further details see documentation.
#
#   DIELC - (Distance-dependent) dielectric constant
#   MAXCYC - Maximum number of cycles of minimization.
#   DRMS - Convergence criterion for the energy gradient.
#
DIELC                   4
MAXCYC                  10000
DRMS                    0.0001
#
##############################################################################
@MAKECRD
#
# The following parameters are passed to make_crd_hg, which extracts snapshots
#   from trajectory files. (This section is only relevant if GC = 1 OR AS = 1 above.)
#
#   BOX - "YES" means that periodic boundary conditions were used during MD
#         simulation and that box information has been printed in the
#         trajectory files; "NO" means opposite.
#   NTOTAL - Total number of atoms per snapshot printed in the trajectory file
#            (including water, ions, ...).
```

```
#   NSTART - Start structure extraction from the NSTART-th snapshot.
#   NSTOP - Stop structure extraction at the NSTOP-th snapshot.
#   NFREQ - Every NFREQ structure will be extracted from the trajectory.
#
#   NUMBER_LIG_GROUPS - Number of subsequent LSTART/LSTOP combinations to
#                       extract atoms belonging to the ligand.
#   LSTART - Number of first ligand atom in the trajectory entry.
#   LSTOP - Number of last ligand atom in the trajectory entry.
#   NUMBER_REC_GROUPS - Number of subsequent RSTART/RSTOP combinations to
#                       extract atoms belonging to the receptor.
#   RSTART - Number of first receptor atom in the trajectory entry.
#   RSTOP - Number of last receptor atom in the trajectory entry.
#   Note: If only one molecular species is extracted, use only the receptor
#         parameters (NUMBER_REC_GROUPS, RSTART, RSTOP).
#
BOX                   YES
NTOTAL                25570
NSTART                1
NSTOP                 5000
NFREQ                 500
#
NUMBER_LIG_GROUPS     0
LSTART                0
LSTOP                 0
NUMBER_REC_GROUPS     1
RSTART                1
RSTOP                 2666
#
###############################################################################
@ALASCAN
#
# The following parameters are additionally passed to make_crd_hg in conjunction
#   with the ones from the @MAKECRD section if "alanine scanning" is requested.
#           (This section is only relevant if AS = 1 above.)
#
# The description of the parameters is taken from Irina Massova.
#
#  NUMBER_MUTANT_GROUPS - Total number of mutated residues. For each mutated
#               residue, the following four parameters must be given
#               subsequently.
#  MUTANT_ATOM1 - If residue is mutated to Ala then this is a pointer on CG
#           atom of the mutated residue for all residues except Thr,
#           Ile and Val.
#           A pointer to CG2 if Thr, Ile or Val residue is mutated to Ala
#           If residue is mutated to Gly then this is a pointer on CB.
#  MUTANT_ATOM2 - If residue is mutated to Ala then this should be zero for
#           all mutated residues except Thr and VAL.
#           A pointer on OG1 if Thr residue is mutated to Ala.
#           A pointer on CG1 if VAL or ILE residue is mutated to Ala.
```

```
#            If residue is mutated to Gly then this should be always zero.
#  MUTANT_KEEP  - A pointer on C atom (carbonyl atom) for the mutated residue.
#  MUTANT_REFERENCE - If residue is mutated to Ala then this is a pointer on
#               CB atom for the mutated residue.
#               If residue is mutated to Gly then this is a pointer on
#               CA atom for the mutated residue.
#  Note: The method will not work for a smaller residue mutation to a bigger
#      for example Gly -> Ala mutation.
#  Note: Maximum number of the simultaneously mutated residues is 40.
#
NUMBER_MUTANT_GROUPS 3
MUTANT_ATOM1          1480
MUTANT_ATOM2          0
MUTANT_KEEP           1486
MUTANT_REFERENCE      1477
MUTANT_ATOM2          1498
MUTANT_ATOM1          1494
MUTANT_KEEP           1500
MUTANT_REFERENCE      1492
MUTANT_ATOM1          1552
MUTANT_ATOM2          0
MUTANT_KEEP           1562
MUTANT_REFERENCE      1549
#
###############################################################################
@TRAJECTORY
#
# Trajectory names
#
#   The following trajectories are used to extract snapshots with "make_crd_hg":
#   Each trajectory name must be preceded by the TRAJECTORY card.
#   Subsequent trajectories are considered together; trajectories may be
#     in ascii as well as in .gz format.
#   To be able to identify the title line, it must be identical in all files.
#
TRAJECTORY            ../prod_II/md_nvt_prod_pme_01.mdcrd.gz
TRAJECTORY            ../prod_II/md_nvt_prod_pme_02.mdcrd.gz
TRAJECTORY            ../prod_II/md_nvt_prod_pme_03.mdcrd.gz
TRAJECTORY            ../prod_II/md_nvt_prod_pme_04.mdcrd.gz
TRAJECTORY            ../prod_II/md_nvt_prod_pme_05.mdcrd.gz
#
###############################################################################
@PROGRAMS
#
# Program executables
#
DELPHI                /home/gohlke/src/delphi.98/exe.R10000/delphi
#
###############################################################################
```

9.3. Auxiliary programs used by MM-PBSA

The DelPhi program is not distributed with Amber. Information about the DelPhi package is available on WWW site:

```
http://honiglab.cpmc.columbia.edu/
```

Other programs for computing numerical Poisson-Boltzmann results are also available, such as MEAD and UHBD. These could be merged into the Perl scripts developed here with a little work. See:

```
http://www.scripps.edu/bashford          (for MEAD)
http://adrik.bchs.uh.edu/uhbd.html       (for UHBD)
```

10. Nmode

Usage:

```
nmode [-O] -i nmdin -o nmdout -c inpcrd -p prmtop -r restrt
       -ref refc -v vecs -l lmode -t tstate -e expfile
```

−O: Overwrite output files if they exist.

10.1. Introduction

This program performs molecular mechanics calculations on proteins and nucleic acids, using first and second derivative information to find local minima, transition states, and to perform vibrational analyses. It is designed to read the *prmtop* and *inpcrd* files from the Amber suite of programs. Both Additive and Non-additive Hamiltonians are available in this version. *Nmode* was originally written at the University of California, Davis, by D.T. Nguyen and D.A. Case, based in part on code in the Amber 2.0 package. Major revisions were made at the Research Institute of Scripps Clinic by J. Kottalam and D.A. Case. M. Pique has provided valuable advice and help in porting it to many different machines. J.W.Caldwell implemented the non-additive capabilities.

The second derivative routines are based on expressions used in the Consistent Force Field programs [117]. The code also contains routines to search for transition state, starting (generally) from a minimum. This procedure uses a modification of the procedure of Cerjan and Miller [118], as described by Nguyen and Case [119]. Langevin modes are analogous to normal modes, but in the presence of a viscous coupling to a continuum solvent. The basic ideas are presented by Lamm and Szabo [120], and were implemented in the Amber environment by Kottalam and Case [121].

10.2. General description This program performs five tasks, depending on the value of the input variable ntrun (see below):

(1) Perform a normal mode analysis from starting coordinates. Requires an input structure that has already been minimized, from process (4), below, or by some other method. In addition to the computation of normal mode frequencies, thermodynamic parameters are calculated.

(2) Search for transition state, starting (generally) from a minimum. See the references above for a detailed description of the method.

(3) Perform a conjugate gradient minimization from the starting coordinates. This routine uses an IMSL library routine for this purpose, which is not supplied with this program. Persons who do not have access to the IMSL library should probably use the AMBER "sander" program to carry out conjugate gradient minimizations. (Use the double precision version for best convergence.)

(4) Does a Newton-Raphson minimization from starting coordinates. A constant (tlamba) is added to the diagonal elements of the Hessian matrix to make it positive definite. Tlamba

is chosen in a manner such that the step is always downhill in all directions. Whenever the change in energy is > emx or the rms of step length is > smx, the step length is scaled back repeatedly until the above two conditions are satisfied. Note that this routine will not converge to a transition state.

(5) Perform a langevin mode calculation, starting from a minimized structure. This option is similar to (1), but includes the viscous effects of a solvent in the calculation.

Input files for this program are the same as for the regular AMBER minimization and molecular dynamics programs, with the exception of *nmdin*, whose parameters are given below. The defaults have been carefully selected, so that for most purposes, few of them need to be changed. See the sample runs for more information.

10.3. Files

```
nmdin  : control input for the run
nmdout : standard output file for print and error messages
prmtop : parameter and topology file
inpcrd : starting coordinates
refc   : input coordinates for constraints
restrt : output coordinates at end of minimization
prlist : file for reading or storing the non-bonded pair list
vecs   : file containing output normal mode frequencies and eigenvectors
tstate : output coordinates at a transition state
expfile: file to read exposed surface area for atoms
lmode  : file to write Langevin modes
```

10.4. Input description

Input found on *nmdin:* You can use as many title cards as you want, followed by the namelist `&data`, which contains the following variables.

General flags describing the calculation

---

ntrun 1: do normal mode analysis *(default)*
2: search for transition states
3: do conjugate gradient minimization (requires IMSL library)
4: do Newton-Raphson minimization
5: do Langevin mode analysis

ibelly 1: some atoms are to be held fixed (default=0)

icons 1: do constrained minimization to initial coordinates specified in *refc*. (default=0)

maxcyc	max. number of cycles for minimization (default=100)
drms	rms gradient to stop minimization (default=1.e-5)
nvect	number of vectors for normal mode analysis (default=0)
ismem	set to 1 for "small memory model" for normal modes, which uses about one-third the memory of the default (ismem=0) model. The tradeoff is that no eigenvectors can be computed, so that nvect is set to zero whenever ismem=1
nsave	for every nsave steps the coordinates are saved. (default= 20)
nprint	every nprint-th step the energy will be printed
ilevel	if .ne. 0, then adjust second derivative matrix to put rotation and translation vectors to a high frequency; this can be useful if you want to perform a normal mode analysis from a not-completely-minimized structure, so that rotations and translations don't mix with the low-lying modes (default=0).
ivform	0 if the normal mode eigenvectors are to be written out in unformatted form; 1 to use the Amber standard formatted option (default); 2 to write out the normal modes in MKL format for the *molekel* program (see *http://www.cscs.ch/molekel*).
ntx	0 if the input coordinates are to be read in unformatted form; 1 to use the formatted option (default).
ntxo	0 if the output (restart) coordinates are to be written out in unformatted form; 1 to use the formatted option (default).
t	Temperature to use in calculating thermodynamic properties from the modes; default is 298.15.

Control of certain force field parameters

---

cut	radius for non-bonded cutoff (default=99.)
scnb	1-4 nonbonded scale factor (default=2.0)
scee	1-4 electrostatic scale factor (default=2.0)
dielc	dielectric constant (default=1.0)
idiel	0 for r**2 dielectric dependence (default); 1 for constant dielectric.
ipol	= 0 no polarization (default) = 1 include polarization modules
i3bod	= 0 no three-body interactions(default) = 1 readin and use specified three body interactions (see the sander input area for details)
iprr	1: read in a non-bonded pair list from *prlist*; (default = 0)
iprw	1: write out non-bonded pair list to *prlist*; (default = 0)

control of Newton-Raphson and transition-state searches

smx maximum rms step length (default = 0.08)

emx maximum energy change per step (default =0.3)

alpha scale factor for step length (default = 0.8) (See Nguyen and Case paper for description of smx, emx, and alpha.)

bdwnhl constant to determine tlamba, the value to be subtracted from the diagonal elements of Hessian matrix for a downhill step. tlamba is chosen as min ((lowest eigenvalue - bdwnhl) , 0). (default bdwnhl = 0.1)

ndiag for every ndiag steps, the matrix is diagonalized to calculate tlamba, when ntrun=4

dfpred a rough estimate of the expected reduction in energy for the initial step (only for ntrun = 3). (default = 0.01 kcal/mol)

parameters for running Langevin modes (set ntrun = 5)

eta viscosity in centipoise

ioseen 0: Stokes Law used for hydrodynamic interaction
1: Oseen interaction included
2: Rotne-Prager correction included

hrmax hydrodynamic radius for the atom with largest area exposed to solvent. If a file named 'expfile' is present, then the relative exposed areas are read from that file as a namelist

```
namelist /exposure/ expr(natom)
```

If 'expfile' does not exist, then all atoms are assigned a hydrodynamic radius of hrmax.

parameters for transition state search (when ntrun = 2)

istart	0: new calc. (default) 1: restart calc.
iflag	0: search for transition state then minimum (default) 1: search for minimum from a transition state -1: search for a transition state, then stop
ivect	no. of eigenvectors wanted (default=2) (ivect has to be >=isdir)
isdir	eigenvector along which search for transition state is to be made. Note that translations and rotations are removed from the Hessian, so this number refers to the ordering of the "true" vibrational normal modes. (default=1)
idir	search direction: = 1 along isdir direction (default); = -1 opposite isdir direction
isw	no. of steps before switching to the lowest mode (default=40)
hnot	initial step length (default=0.1 Ang.)
buphl	switch to Newton-Raphson step when lowest eigenvalue is less than this for uphill walk. (default=-0.1)

Cards 3	group cards for the parts of the molecule that move, if ibelly.ne.0. See group documentation for format.
Cards 4	group cards for the part of the molecule to be constrained, along with the constraint weights, if icons.ne.0. See group documentation for format.

Memory usage: Normal mode analysis can take a lot of memory; users should consult the `alloc.f` file to see all of the details. The biggest memory hog is generally for the second derivative (Hessian) matrix. In the "normal" case, for *ntrun=1*, the program requires $9N(3N-1)/2$ 8-byte words of storage. For a 400 atom system, this is about 2.1 million words, or 17 Mbytes, which is generally no problem. For a 4000 atom system, however, this translates to 216 million words, or 1.7 Gbytes, which may not always be available. For larger systems, normal mode calculations that store everything in memory become increasingly impractical.

By setting *ismem* to 1, you can reduce the memory usage to 1/3 of the above estimate, at the expense of not calculating eigenvectors. This can sometimes make calculations feasible that otherwise would not be, but only for a fairly narrow range of problem sizes. More elaborate schemes, involving sparse matrix storage, are certainly possible, but have not yet been implemented in *nmode*.

11. Miscellaneous

11.1. Resp

Usage:

```
resp [-O] -i input -o output -p punch -q qin -t qout
                 -e espot -w qwts -s esout
```

−O Overwrite output files if they exist.

RESP (Restrained ElectroStatic Potential) fits the quantum mechanically calculated electrostatic potential (esp) at molecular surfaces using an atom-centered point charge model. This method was developed primarily by Christopher Bayly. [122-124] A quantum mechanical program, such as Gaussian, Jaguar, or GAMESS, must be used to generate the ESP input for RESP. See $AMBERHOME/src/resp/0README for tips for interfacing such programs with RESP. Note that *antechamber* automates most of this process: use the *-fo gcrt* option to create a Gaussian input file; then run Gaussian; then use the *-fi gout -c resp* option to automatically create the resp input file and run a two-stage fitting procedure. If you don't use Gaussian, you can still run *respgen* to automatically create the input files needed for resp.

file name	flag	fortran unit		purpose
input	-i	5	required	input options
output	-o	6	a/p	output of results
punch	-p	7	a/p	synopsis of results
qin	-q	3	optional	replacement charges
qout	-t	19	a/p	ouput of current charges
espot	-e	10	required	input of ESP's and coordinates (note: these must in atomic units)
qwts	-w	4	optional	input of new weight factors
esout	-s	20	optional	generated esp values for new charges

a/p = always produced

Input included in the "-i" file

-1st line-

```
TITLE       input: a character string
```

-2nd section-

```
    Begin with namelist " &cntrl"    (see example at end)

       inopt   =  0  normal run
               =  1  cycle through a list of different qwt
                        read from -w unit

      ioutopt  =  0  normal run
               =  1  write restart info of new esp etc to
                     unit -es (esout unit)

        iqopt  =  1  reset all initial charges to zero (default)
               =  2  read in new initial charges from -q (qwt)
               =  3  read in new initial charges from  -q (qwt)
                     and perform averaging of those new
                     initial charges according to ivary values
                     (normally not used)

         nmol  =  n  the number of molecules in a multiple molecule
                     fit (default 1)

       ihfree  =  0  all atoms are restrained
               =  1  hydrogens not restrained (default)

      irstrnt  =  0  harmonic restraints (old style)
               =  1  hyperbolic restraint to charge of zero (default)
               =  2  only analysis of input charges; no
                     charge fitting is carried out

          qwt  =  normally use 0.0005 for Stage 1 (default)
                     "      "  0.001  for Stage 2

    end namelist " &cntrl" with " &end"
```

-3rd "line"-

```
    wtmol .... relative weight for the molecule if
                multiple molecule fit (1.0 otherwise)
```

-4th "line"-

```
    subtitle for molecule (a character string)
```

-5th "line"-

```
    charge, iuniq (charge and number of atoms, in 2I5 format)
```

-6th "area"-

```
 one line for each atom, in 2I5 format:

 Atomic number, ivary

        ivary
              =  0  charge varied independently of previous
                    centers
              =  n  current charge fitted together with
                    center "n"
              = -99 charge frozen at "initial charge" value
                    typically read in unit "qin"
```

-7th- "area"

```
charge constraints...  blank line if no constraints, otherwise
                       in I5,F10.5 format

ngrp = number of centers in the group associated with this
       constraint (i.e. the number of centers to be read in)

grpchg(i) = charge to which the associated group of atoms
           (given on the next card) is to be constrained
```

-7.1th- "area"

```
imol, iatom  (in 16I5 format)

  the list (ngrp long) of the atom indices of those atoms to be
  constrained to the charge specified on the previous line.

 *blank to end
```

-8th "area"-

```
intermolecular charge constraints
same format as indvidual molecule constraints

 *blank to end
```

-9th "area"-

```
Multiple molecule atom equivalencing
format is analagous to 7th area and 7.1
ngrp(I5) and then, on separate lines: imol,iatoms(16I5)

 *blank to end
```

```
Other file formats

   -q    input of replacement charges if requested, 8F10.6 format
         (note: same format as produced by -t)
```

```
   -w  input of new weight factors if requested

     input: i5     nqwt  number of new weights to cycle thru

     input: f10.5  new weights (nqwt lines)
```

```
  -e   input of ESP's and coordinates

    -1st line-

       n_atoms n_esp_points (2I5 format)

    -2nd- 2->natoms+1 lines-

   atom coordinates
   x,y,z (in Bohrs)   (format is 17x,3e16.7)

  -3rd natoms+2->natoms+2+nesp lines-

   potential and coordinate
   qpot,x,y,z  (in a.u.,bohrs)   (format is 1X,4E16.7)
```

Several examples of input and output files are in `$AMBERHOME/examples/resp_charge_fit`; these should be consulted by those interested in running the program.

11.2. nucgen

Usage: `nucgen [-O] -i ngin -o ngout -d ngdat -p pdbout`

−O Overwrite output files.

This program generates cartesian coordinate models for either double helical DNA or RNA with a number of possible conformations. The conformations are taken from fibre-diffraction studies [125]. The helical topology of the double helix is stored in a file for individual types in terms of cylindrical coordinates. The program loads the required topoplogy and applies two fold symmetry with necessary helical repeat and height values. The cartesian coordinates are output in PDB format. The residue information is read as in the link module either for DNA or RNA. The input is described below.

NUCGEN requires specification of two strands: if only one is given, it will wrap it into two with highly stretched base-phosphate bonds across the end, so for single strands, specify a dummy strand and edit it out of the resulting PDB file. NUCGEN only generates reasonable geometries for complementary base pairs.

NUCGEN can generate PDB files using the 1994 Amber force field convention, which does not have explicit terminal hydrogen or phosphate residues. For the new residue names, only the bases need to be specified, while for the old convention, terminal hydrogen residues (HB and HE) and phosphates (POM) must also be specified. In the 1994 convention, residues are indicated by the first letter (A, G, C, T) and terminal residues have an additional 5 or 3 appended (*e.g.* A5, A3). See the LEaP chapter for a table of these names.

```
file    unit        purpose
------------------------------------------------------------------
ngin      5   Input:   Control and sequence data for the run
ngout     6   Output:  Diagnostics
ngdat     7   Input:   Monomer geometry file, found in amber41/dat
pdbout   10   Output:  PDB output coordinates
------------------------------------------------------------------
```

Nucleic Acid sequence information is given as described here for each strand. Both strands are entered in the 5’ to 3’ direction.

```
-------------------------------------------------------------

  - 1A -     A TITLE FOR EACH STRAND (20A4)

  TITLE      A title for the molecule.

-------------------------------------------------------------

  - 1B -     ILBMOL (A4)

  ILBMOL     Label for the type of molecule.

        'D'  DNA
        'R'  RNA

-------------------------------------------------------------

  - 1C -     RESIDUE INFORMATION FOR EACH STRAND
             it is read in the following format until a blank
             card is encounterd (card 1D).

             LBRES(I) , I = 1,NRESM     (16(A4,1X))

  LBRES(I)   Residue name.
```

```
------------------------------------------------------------
  - 1D -     Blank Card to terminate residue input
------------------------------------------------------------
  NOTE:  Cards 1A-1D are repeated for the second strand.
------------------------------------------------------------

  - 2 -      KEND    (A4)

  KEND       Control to stop reading the nucleotide strands.

     'END '  end of reading the sequence information

------------------------------------------------------------

  - 3 -      CONTROL FOR THE TYPE OF DNA OR RNA CONFORMATION

             TYPM     (A8)

  TYPM       Name of the type of conformation to be generated.

   '$ARNA'     right handed a-rna (arnott)
   '$APRNA'    right handed a-prime rna (arnott)
   '$LBDNA'    right handed bdna (langridge)
   '$ABDNA'    right handed bdna (arnott)
   '$SBDNA'    left handed bdna (sasisekharan)
   '$ADNA'     right handed a-dna (arnott)
   '$SPECIAL'  special type by the user

------------------------------------------------------------

  - 4 -       special helical parameter

              ***** only if typm .eq. '$SPECIAL' *****

              hxrep , hxht    (2f10.5)

  hxrep       Helical repeat angle in degrees for the special
              type of conformation.

  hxht        Helical height.

              NOTE: If you use '$SPECIAL', you will have to
                    add the appropriate data to file ngdat (found
                    in the database directory).  Consult subroutine
                    gennuc for details.
------------------------------------------------------------
```

11.3. ambpdb

NAME

ambpdb − convert amber-format coordinate files to pdb format

SYNOPSIS

ambpdb [-p prmtop-file][-tit title] [-pqr|-bnd|-atm] [-aatm] [-bres] [-noter] [-offset #] [-bin] [-first]

DESCRIPTION

ambpdb is a filter to take a coordinate "restart" file from an AMBER dynamics or minimization run (on STDIN) and prepare a pdb-format file (on STDOUT). The program assumes that a *prmtop* file is available, from which it gets atom and residue names.

OPTIONS

title The title, if given, will be output as a REMARK at the top of the file. It should be protected by quotes or double quotes if it contains spaces or special characters.

-pqr If *-pqr* is set, output will be in the format needed for the MEAD suite of programs created by Don Bashford. The *-atm* option creates files used by Mike Connolly's surface area/volume programs. The *-bnd* option creates a file that lists the bonds in the molecule, one per line.

-aatm This switch controls whether the output atom names follow Amber or Brookhaven (PDB) formats. With the default (when this switch is not set), atom names will be placed into four columns in an approximation to the rules used by the Protein Data Base. This gives files that look very much like PDB files, EXCEPT that PDB uses "1" and "2" for amino-acid beta-protons (for example) whereas the standard Amber database (along with many in the NMR field) use "2" and "3", i.e. we have 2HB and 3HB, whereas Brookhaven files use 1HB and 2HB. The *protonate* program can be used to check and re-name proton names to various conventions.

If *-aatm* is set, Amber atom names will be left-justified in the output file, starting in column 13.

Generally speaking, Amber programs that read PDB files (like *protonate* and *LEaP*, work with either style of atom names. Programs like RASMOL, that expect more strict conformance to Brookhaven standards, require the default behavior; some other programs may work better with *-aatm* set, so that (for example) all hydrogen atoms begin with "H", etc.

-bin If *-bin* is set, an unformatted (binary) "restart" file is read instead of a formatted one (default). Please note that no detection of the byte ordering happens, so binary files should be read on the machine they were created on.

-bres If *-bres* (Brookhaven-residue-names) is not set (the default), Amber-specific atom names (like CYX, HIE, DG5, etc.) will be kept in the pdb file; otherwise, these

will be converted to PDB-standard names (CYS, HIS, G, in the above example). Note that setting -bres creates a naming ambiguity between protonated and uprotonated forms of amino acids, and between DNA and RNA.

If you plan to re-read the pdb file back into Amber programs, you should use the default behavior; for programs that demand stricter conformance to Brookhaven standards, set *-bres*.

-first If *-first* is set, a pdb file augmented by additional information about hydrogen bonds, salt bridges, and hydrophobic tethers is generated, which can serve as input to the standalone version of the FIRST software by D. J. Jacobs, L. A. Kuhn, and M. F. Thorpe.

-noter If *-noter* is set, the output PDB file not include TER cards between molecules. Otherwise, TER cards will be added whenever there is not bond between adjacent residues. Note that this means there will be a TER card between each water molecule, for example, unless *-noter is set*. The PDB is idiosyncratic about TER cards: they are generally present between separate protein chains, but generally not present between cofactors or solvent molecules. This behavior is not mimicked by *ambpdb*.

-offset If a number is given here, it will be added to all residue numbers in the output pdb file. This is useful if you want the first residue (which is always "1" in an Amber prmtop file, to be a larger number, (say to more closely match a file from Brookhaven, where initial residues may be missing). Note that the number you provide is one less than what you want the first residue to have.

Residue numbers greater than 9999 will not "fit" into the Brookhaven format; ambpdb actually prints mod(resno,10000); that is, after 9999, the residue number re-cycles to 0.

FILES

Assumes that a *prmtop* file (with that name, or the one given in the *-p* option) exists in the current directory; reads AMBER coordinates from STDIN, and writes pdb-file to STDOUT.

BUGS

Inevitably, various niceties of the Brookhaven format are not as well supported as they should be. The *protonate* program can be used to fix up hydrogen atom names, but that functionality should really be integrated here. There is no good solution to the PDB problem of using the same residue name for different chemcial species; depending on how the output file is to be used, the two options supported (setting or not setting *-bres*) may or may not suffice. Radii used for the *-pqr* option are hard-wired into the code, requiring a re-compilation if they are to be changed. Atom name output may be incorrect for atoms with two-character atomic symbols, like calcium or iron. The *-offset* flag is a very limited start toward more flexible handling of residue numbers; in the future (we hope!) Amber *prmtop* files will keep track of the "original" residue identifiers from input pdb files, so that this information would be available on output.

11.4. protonate

NAME

protonate − add protons to a heavy-atom protein or DNA PDB file; convert proton names between various conventions; check (pro)-chirality.

SYNOPSIS

```
Usage: protonate [-bcfhkmp] [-d datafile]
 [-i input-pdb-file] [-o output-pdb-file] [-l logfile]
 [-al link-file] [-ae edit-file] [-ap parm-file]
   -b to write Brookhaven-like atom names
   -c to write chains as separate molecules
   -f to force write of atoms found (debugging)
   -h to write ONLY hydrogens to output file
   -k to keep original coordinates of matched protons
   -m to list mismatched protons
   -p to print proton substitutions
   -d to specify datafile (default is PROTON_INFO)
   -i to specify input file (default is stdin)
   -o to specify output file (default is stdout)
   -l to specify logfile (default is stderr)
```

DESCRIPTION

Protonate combines a program originally written by K. Cross to add protons to a heavy-atom pdb file, with many extensions by J. Holland, G.P. Gippert & D.A. Case. Names and descriptions of the output protons are contained in the info-file (see below.) *Protonate* can be used to add protons that don't exist, to change the names of existing protons to some new convention, and to check pro-chirality of protons in an input pdb file. The source code is in the `src/protonate/` directory. Protonate generally will not do a careful job of orienting polar hydrogens, particularly for hydroxyls of serine, threonine and tyrosine; you can use the *pol_h* program (described below) for this purpose.

OPTIONS

−k The output pdb file will keep the proton coordinates of the input file, to the extent consistent with how well it can identify what names they should really have. Otherwise it will replace input protons with ones it builds.

−b The program will insert a space before the name of each heavy atom in the output file. This is most often used to convert input files whose atom names begin in column 13 to the Brookhaven format where most heavy atom names begin in column 14. NOTE: two-letter heavy atom names (like FE or CA [calcium]) will not be correct; the resulting output file must be hand-edited to check for this.

−d info_file Specifies the file containing information on how to build and name protons. The default name is PROTON_INFO. This information used to determine where on the amino acids the protons should be placed. The file provided handles funny Amber residue names like HIE, HIP and HID and HEM. Other files provided

include PROTON_INFO.Brook, which uses Brookhaven proton naming convention (such as 1HB, etc.), and PROTON_INFO.oldnames, which uses old amber names. For example, to take an Amber pdb file and convert to the Brookhaven naming convention, set -d PROTON_INFO.Brook.

Output to LOGFILE includes matches of protons the program builds with any found in the input file, plus a list of any input protons that could not be matched. Questionable matches are flagged and should be checked manually.

BUGS

Format of the PROTON_INFO file is not obvious unless you have read the code.

Methyl protons are built in a staggered conformation; hydroxyl protons in a arbitrary (and generally sub-optimal) conformation. A program like *pol_h* or its equivalent should be used (if desired) to place polar hydrogens on LYS, SER, THR, and SER residues.

HIS in the input file is assumed to be HID. Users should generally explicitly figure out the desired protonation state for histidines.

No attempt is made to identify heavy atoms in the input file that have two-letter element names; this means that Brookhaven-style output may require some hand-editing if atoms like calcium or iron are present.

It is assumed that the alternate conformer flag in column 17 of the PDB file is either blank, or A. The program needs to be recompiled to change this; perhaps this should become an input option.

11.5. ambmask

NAME

ambmask − test group input FIND mask (or mask string given in the &cntrl section) and dump the resulting atom selection in a given format

SYNOPSIS

```
ambmask -p prmtop -c inpcrd -prnlev [0-3] -out [short|
pdb| amber] -find [maskstr]
```

DESCRIPTION

ambmask acts as a filter which takes amber topology and coordinate "restart" file and applies the "maskstr" selection string (similar syntactically to UCSF Chimera/Midas) to select specific atoms or residues. Residues can be selected by their numbers or names. Atoms can be selected by numbers, names, or amber (forcefield) type. Selections are case insensitive. The selected atoms are printed to **stdout** (by default, in amber−style pdb format). Atom and residue names and numbers are taken from amber topology. Beware that selection string works on those names and not the ones from the original pdb file. If you are not sure how atoms or residues are named or numbered in the amber topology, use **ambmask** with a selection string ":*" (which is the default) to dump the whole pdb file with corresponding amber atom/residue names and numbers.

The "maskstr" 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 "%". ":*" means all residues and "@*" means all atoms. The following examples show the usage of this syntax. Square brackets should not be used in actual expressions, they are only used for clarity here:

```
:{residue numlist}          [:1-10] [:1,3,5] [:1-3,5,7-9]
:{residue namelist}         [:LYS] [:ARG,ALA,GLY]
@{atom numlist}             [@12,17] [@54-85] [@12,54-85,90]
@{atom namelist}            [@CA] [@CA,C,O,N,H]
@%{atom typelist}           [@%CT] [@%N*,N3]
```

These "elementary selections" can be combined into more complex selections using binary operators "&" (and) and "|" (or), unary operator "!" (negation), distance binary operators "<:", ">:", "<@", ">@", and parentheses. Spaces around operators are irrelevant. Parentheses have the highest priority, followed by distance operators ("<:", ">:", "<@", ">@"), "!" (negation), "&" (and) and "|" (or) in order of descending priority. A wildcard "=" in an atom or residue name matches any name starting with a given character (or characters). For example, `[:AS=]` would match all aspartic acid residues (ASP), and asparagines (ASN); `[@H=]` would match all atom names starting with H (which are effectively all hydrogens). It cannot be used to match the end part of names (such as `[:=A]`). Some examples of more complex selections follow:

`[@C= & !@CA,C]`
.. all carbons except backbone alpha and carbonyl carbons
`[(:1-3@CA | :5-7@CB)]`
.. alpha carbons in residues 1-3 and beta carbons in residues 5-7
`[:CYS,ARG & !(:1-10 | @CA,CB)]`
.. all CYS and ARG atoms except those which are in residues 1-10 and which are CA or CB
`[:* & !@H=]` or `[!@H=]`
.. all heavy atoms (i.e. except hydrogens)
`[:5 <@4.5]`
.. all atoms within 4.5A from residue 5
`[(:1-55 <:3.0) & :WAT]`
.. all water molecules within 3A from residues 1-55

Compound expressions of the following type are also allowed:

`:{residue numlist|namelist}@{atom numlist|namelist|typelist}`
`[:1-10@CA]` is equivalent to `[:1-10 & @CA]`
`[:LYS@H=]` is equivalent to `[:LYS & @H=]`

OPTIONS

The program needs an amber topology file and coordinates (restrt format). The filename specified with the *−p* option is amber topology, while the filename given with the *−c* option is a coordinate file. If *−p* or *−c* options are not given, the program expects that files "prmtop" and/or "inpcrd" exist in the current directory, which will be taken as topology and coordinate files correspondingly. If no command line options are given, the program prints the usage statement.

The option *−prnlev* specifies how much (debugging) information is printed to **stdout**. If it is 0, only selected atoms are printed. More verbose output (which might be useful for debugging purposes) is achieved with higher values: 1 prints original "maskstr" in its tokenized (with operands enclosed in square brackets) and postfix (or Reverse Polish Notation) forms; number of atoms and residues in the topology file and number of selected atoms are also printed to **stdout**. 2 prints the resulting mask array, which is an array of integer values, with ’1’ representing a selected atom, and ’0’ an unselected one. Value of 3, in addition, prints mask arrays as they are pushed or popped from the stack (this is really only useful for tracing the problems occurring during stack operations). The *−prnlev* values of 0 or 1 should suffice for most uses.

The option *−out* specifies the format of printed atoms. "short" means a condensed output using residue (:) and atom (@) designators followed by residue ranges and atom names. "pdb" (default) prints atoms in amber-like pdb format with the original "maskstr" printed as a REMARK at the top of the pdb file, and "amber" prints atom/residue ranges in the format suitable for copying into group input section of amber input file.

The option *−find* is followed by "maskstr" expression. This is a string where some characters have a special meaning and thus express what parts (atoms/residues) of the molecule will get selected. The syntax of this string is explained in the section above (DESCRIPTION). If this option is left out, it defaults to ":*", which selects all atoms in the given topology file. The length of "maskstr" is limited to 80 characters. If the "maskstr" contains spaces or some special characters (which would be expanded by the shell), it should be protected by single or double quotes (depending on the shell).

FILES

Assumes that a *prmtop* and *inpcrd* files exists in the current directory if they are not specified with −*p* and −*c* options. Resulting (i.e. selected) atoms are written to **stdout**.

BUGS

Because all atom names are left justified in amber topology and the selections are case insensitive, there is no way to distinguish some atom names: alpha carbon CA and a calcium ion Ca are a notorious example of that.

11.6. pol_h and gwh

NAME

pol_h − set positions of polar hydrogens in proteins
gwh − set positions of polar hydrogens onto water oxygen positions

SYNOPSIS

```
pol_h < input-pqr-file > output-pdb-file

gwh [-p <prmtop>] [-w <water.pdb>] [-c] [-e] < input_pdb_file
     > output_pdb_file
```

DESCRIPTION

The program *pol_h* resets positions of polar hydrogens of protein residues (Lys, Ser, Tyr and Thr), by optimizing simple electrostatic interactions. The input *pqr* file can be created by *ambpdb*.

The program *gwh* sets positions of water hydrogens onto water oxygen positions that may be present in PDB files, by optimizing simple electrostatic interactions. If the -w flag is set, the program reads water oxygen positions from the file *water-position-file*, rather than the default name *watpdb*. If *-c* is set, a constant dielectric will be used to construct potentials, otherwise the (default) distant-dependent dielectric will be used. If *-e* is set, the electrostatic potential will be used to determine which hydrogens are placed first; otherwise, a distance criterion will be used.

Accuracy of pol_h & gwh:

```
* In the following the results for BPTI and RSA(ribosuclease A) are
   given together with those of Karplus(1) and Ornstein(2) groups.
   In the case of Ornstein's method, it handles only some of hydrogens
   in question and therfore I normalized(scaled) their results using
   expected values for random generation.  The rms deviation from the
   experimental positions (neutron difraction) and the number of
   hydrogens are shown below.

      BPTI        Lys    Ser    Tyr    Thr       Wat
-------------------------------------------------------------
      # of H     12      1       4       3      112  (4~)
    Pol_H       0.39    0.36    1.08    0.20    0.98(0.38)
    Karplus     0.25    0.71    0.81    0.19     -  (0.35)
    Ornstein    0.22    0.96    0.00    0.07     -
 Ornstn(scaled) 0.51    0.96    1.28    0.07        (1.17)^
-------------------------------------------------------------
      ~internal waters.  ^by random generation


 RSA        Lys    Ser    Tyr    Thr    Wat
```

```
-----------------------------------------------------------
    # of H     30     15     6     10    256
GuesWatH       0.61    0.96    1.22    0.96    0.98
Karplus        0.60    0.98    0.60    1.12    1.20
Ornstein       0.20    0.61    0.60    0.30     -
Ornstn(scaled) 0.49    0.89    0.76    0.93   (1.14)^
-----------------------------------------------------------
      ^by random generation

   1) A. T. Brunger and M. Karplus, Proteins, 4, 148 (1988).
   2) M. B. Bass,,, R. L. Ornstein, Proteins, 12, 266 (1992).

* The accuracies seem to be similar among three approaches
  if scaled values of Ornstein's data are considered.
```

FILES Default for <prmtop-file> is "prmtop". The input-pdb-file must have been generated by LEaP or ambpdb, *i.e.* it must have exactly the same atoms (in the same order) as the prmtop file.

11.7. intense

NAME

intense − compute NOESY intensities from a structure

SYNOPSIS

intense -tauc *rot._corr._time,ns* -taum *mixing_time,sec.*
[-taumet *value,MHz* -omega *value,ns*
-leak *value,sec-1* -cutoff *cutoff*
-p *pdb_file* -i *output_intensity_file*
-c *output_shift_skeleton* -s *smatrix_file*]

DESCRIPTION

Intense takes a structure from a pdb file as input, and outputs a list of NOESY intensities, suitable for input to other programs such as *spectrum*. The source code is in the `src/nmr_aux/` directory.

The input pdb file must include all hydrogens that you want to include in the spin system. If a phe or tyr residue exists, the order of the hydrogen names must be HD1, HE1, HZ (or HOH), HE2, HD2. IUPAC-IUB names for methyls are required for them to be properly identified. The command line also must include rotational correlation time and a mixing time.

All intensities greater the *cutoff* (default 0.0005) will be printed in the output intensity file. OMEGA (spectrometer frequency, default 500 MHz) and TAUMET (correlation time for methyl jump motion, default 0.001 ns) are discussed in the Sander documentation.

A "leakage rate" can be added to diagonal elements of the rate matrix to simulate relaxation caused by mechanisms other than dipolar relaxation with other protons. This is accomplished via the "-leak" flag, followed by the leakage rate in sec**-1.

If present, the S-matrix file will be read and used instead of computing distances from the pdb file; any matrix elements not present in the *smatfile* will be estimated from the distances in the pdb file. The format of the *smatfile* is a namelist "smat", containing the two-dimensional variable "s". Indices of s are the absolute proton numbers, i.e. those in the pdb file.

The output *cshfile* contains the atom names in the proper order for providing chemical shift information to the next program, *spectrum.* Edit this file to put the chemical shifts in the first 15 columns. If you have already put your chemical shift information into a database of the format support by Garry Gippert, then the script */case/nmr/spectrum/shiftconv* will take the skeleton file that *intense* makes and insert the proper shifts for you. See that shell script for instructions on using it.

Default file names are *pdbfile*, *intfile*, *smatfile*, and *cshfile*.

SEE ALSO

These programs are based on the "remarc" codes in Amber 4.0.

DIAGNOSTICS

File names, correlation and mixing times and number of protons are output to

stderr. An error message is generated if the input pdb file has more than 750 protons or more than 1500 total atoms; the program needs to be recompiled after changing the appropriate variables in the "nmr.h" file.

BUGS

Format of the output file should be expanded to allow the parameters used to be embedded as comments.

If a tyrosine is present, the proton HOH must be present, even if this is a D20 simulation in which that proton has been exchanged away. The work-around is to include HOH, but with very large coordinates (e.g. 999.,999.,999) so that it won’t contribute to the spin systems. Other exchanged protons can be left out, or entered as "D...".

11.8. spectrum

NAME

spectrum − compute smx format file from the output of *intense*

SYNOPSIS

spectrum [-c *chemical-shift-file* -i *intense-file* -s *smx-file* -s1min *omega1-min* -s1max *omega1-max* -s2min *omega2-min* -s2max *omega2-max* -hwidth *half-width*]

DESCRIPTION

Spectrum takes the output of the *intense* program (*q.v.*) plus information on chemical shifts and produces an output "smx" format spectrum that ranges from s1min to s1max and from s2min to s2max. The source code is in the `src/nmr_aux/` directory.

Peaks are assigned a half-width given by hwidth. The defaults are 0 to 10 ppm in each direction, with a half width of 0.05 ppm. Default filenames are *cshfile*, *int-file* and *smxfile.smx*. The output file is a 512 x 512 real smx file that should be acceptable for viewing or processing by *ftnmr*. Since a square matrix is first set up, and then converted to the funny block smx format, it should be relatively easy to modify this program to accommodate other nmr analysis packages, or other plotting programs, etc. To accommodate the default contour levels in *ftnmr*, the peaks are multiplied by 10**7.

The program is currently configured for a maximum of 900 protons. This can easily be changed by modifying parameter statements at the beginning of the program.

SEE ALSO

intense
Sample calculation is in */case/nmr/spectrum/example*.

BUGS

Only 512 x 512 spectra can be output. This should not be too hard to fix with a code hack.

The width of the peaks must be the same in each direction, and the same for all peaks.

The header information that *ftnmr* uses is not fully documented, but spectrum will set up some of the important ones: it assumes a spectral frequency of 500.00 MHz in each dimension, sets up the proper spectral width and reference points (referenced at the edge of the spectrum) and sets the axis type to "ppm". A simple revision could make the spectrometer frequency an input variable.

11.9. fantasian

```
A program to evaluate magnetic anisotropy tensor parameters
                      Ivano Bertini
    Depart. of Chemistry, Univ. of Florence, Florence, Italy
              e-mail: bertini@risc1.lrm.fi.cnr.it
```

INPUT FILES:

Observed shifts file (pcshifts.in):

```
1st    column   -->   residue number
2nd    column   -->   residue name
3rd    column   -->   proton name
4th    column   -->   observed pseudocontact shift value
5th    column   -->   multiplicity of the NMR signal (for example
                      it is 3 for of a methyl group)
6th    column   -->   relative tolerance
7th    column   -->   relative weight
```

Amber pdb file (parm.pdb): coordinates file in PDB format. If you need to use a solution NMR family of structures you have to superimpose the structures before to use them.

OUTPUT FILES:

Observed out file (obs.out): This file is built and read by the program itself, it reports the data read from the input files.

output file (res.out): The main output file. In this file the result of the fitting is reported. Using fantasian it is possible to define an internal reference system to visualize the orientation of the tensor axes. Then in this file you can find PDB format lines (ATOM) which can be included in a PDB file to visualize the internal reference system and the tensor axes. In the main output file all the three equivalent permutations of the tensor parameters with respect to the reference system are reported. The summary of the minimum and maximum errors and that of errors^2 are also reported.

Example files: in the directory example there are all the files necessary to run a fantasian calculation:

```
fantasian.com  --> run file
pcshifts.in    --> observed shifts file
parm.pdb       --> coordinate file in PDB format
obs.out        --> data read from input files
res.out        --> main output file
```

11.10. pbsa

This is a stand-alone program that is much like sander with the *igb=10* option, except that the dielectric boundary between the low and high dielectric regions is the molecular surface of the solute, rather than the boundary of the spherical cap of water. The PB solvent can only be used when *imin* = 1 and *maxcyc* = 1, i.e. it is only used for single-point energy calculation. The *surften* variable can be used to assign surface tension for the nonpolar SA energy as in the GBSA method. It should be pointed out that MPI has not been implemented in PBSA because it is primarily used as a post- processing tool (invoked by the *mm_pbsa* script) in this release.

All PB options described below can be defined in the *&pb* namelist, which is read immediately after the *&cntrl* namelist. Note that it is not necessary to use the *&pb* namelist at all to turn on PB as long as *igb* = 10. Of course, this means that you only want to use default options for PB. The *&pb* namelist has the following variables:

EPSIN — Sets the dielectric constant of the solute region, default to 1.0.

EPSOUT — Sets the implicit solvent dielectric constant, default to 80.

ISTRNG — Sets the ionic strength (in mM) for the Poisson-Boltzmann solvent, default to 0 mM.

PBTEMP — Temperature used for the PB equation, needed to compute the Boltzmann factor for salt effect, default to 300 K.

RADIOPT — The option to set up atomic cavity radii for molecular surface calculation and dielectric assignment. A value of 0 uses the cavity radii from the prmtop file. A value of 1 (the default) sets up optmized cavity radii at the PB initialization phase. These default atomic cavity radii are optimized for model compounds of proteins only. Use cautions when applying these radii to nucleic acids. For optimal results, it is recommended that a user study the performance of PBSA reproducing solvation free energies of model compounds for molecules other than proteins

SPROB — Solvent probe radius, default to 1.6 Å, the sigma value of TIP3P water. The radii set up by *radiopt* = 1 are optimized with a *sprob* of 1.6 Å, a grid spacing of 0.5 Å, and *surften* = 5 cal/mol-Å^2.

NSAS — PBSA uses a numerical method to compute solvent accessible surface area. The *nsas* variable gives the approximate number of dots to represent the maximum atomic solvent accessible surface, default to 400. These dots are first checked against covalently bonded atoms to see whether any of the dots are buried. The exposed dots from the first step are then checked against a nonbonded pair list for van der Waals interactions (see below) to see whether any of the exposed dots from the first step are buried. The exposed dots of each atom after the second step then represent the solvent accessible portion of the atom and are used to compute the SASA of the atom. The molecular SASA is simply a summation of the atomic SASA's. Molecular SASA is used for both PB dielectric map assignment and nonpolar SA energy calculation.

SPACE — Sets the grid spacing for the finite-difference solver, default to 0.5 Å.

ACCEPT — Sets the convergence criterion (relative) for the finite-difference solver, default to 0.001. Higher accuracy, i.e. more significant digits in the electrostatic energy, requires tighter convergence criterion. However, it is not necessary to

use very tight convergence as long as the accuracy in the PB electrostatics energy is significantly smaller than the accuracy (standard deviation) in the MM/PBSA energy averaging.

MAXITN — Sets the maximum number of iterations for the finite-difference solver, default to 100. Larger systems and systems with many highly charged atoms usually require longer iterations.

CUTFD — Atom-based cutoff distance to remove short-range finite-difference Coulombic interactions, default to 10 grid units. See discussion in 5.15.1 and Eqn (20) in Lu and Luo [10].

CUTNB — Atom-based cutoff distance for van der Waals interactions, default to 9 Å.

NSNBA — Sets how often the atom-based pairlist is generated, default to 5 steps. Note that in the atom-based pairlist, all finite-difference Coulombic pairs are included in the van der Waals pairs because the *cutnb* is longer than *cutfd*, and both *cutnb* and *cutfd* are less than cutres.

CUTRES — Residue-based cutoff distance, default to 12 Å. The residue-based nonbonded list is used to make the generation of the atom-based cutoff list efficient because it is updated very frequently, typically every 5 steps.

NSNBR — Sets how often residue-based pairlist is generated, default to 25 steps.

OUTPHI — When set to 1, PBSA outputs electrostatic potential (kcal/mol-e) in ascii format for visualization. When this option is turned on, the PBSA program will stop after printing out a file named *phi.dat*, default to 0. The information on the data format and grid positions and dimensions are all included in the header section of the file.

NPBVERB — When set to 1, turns on verbal mode for PB calculations, default to 0. Atomic cavity radii, charges, and convergence data of the FDPB solver will be printed out when it is turned on.

12. Science background

This chapter gives some brief introductions to background information that may be useful to Amber users. These sections offer pointers to original literature, but are not substitutes for textbooks or other reviews.

12.1. Parameter Development

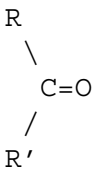
Allison Howard and Bill Ross

How should one proceed to develop parameters for new molecules or fragments? The general principle is to use analogy as much as possible. The amount of effort that should be expended is related to the scientific question being asked. To accurately calculate thermodynamic interactions with water or a macromolecule, one needs the best parameters that can be obtained. If only qualitatively reasonable geometries are needed, less work may be required. Van der Waals, bond, angle, torsion and improper torsion parameters are discussed; the philosophy of derivation of specific atomic charges for a new residue is given in Chapter 13.2.

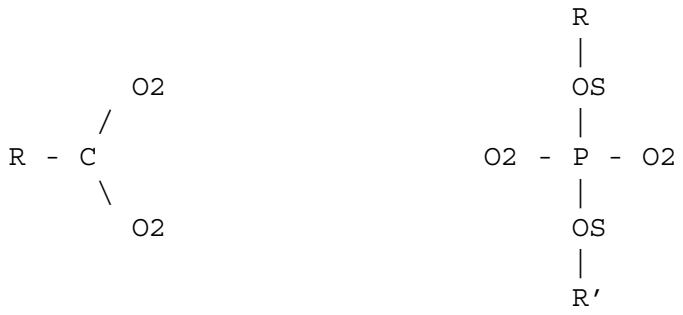
Atom types. The first step in parameter development is to make a two-dimensional sketch of the fragment for which parameters are needed, and then to assign atom types to the atoms. The comments in the first section of the `parm94.dat` file describe the hybridization and other attributes of the atom types for the 1994 force field [26]. This approach may be augmented by looking at the atom types in the existing residues in the files `all_*94.in`. For example, in pyridine the nitrogen would be assigned the same type (NC) as N1 and N3 in adenine. Note that an atom type is intrinsic to an array of distance, angle, and dihedral parameters involving the types of the neighboring atoms, as well as having its own van der Waals (VDW) parameters, and, of course, atomic mass (charge is not fixed per atom type). Therefore, if a new atom type is required, the first step is to attempt to reason by analogy and clone as many of the pre-existing parameters as possible to account for the environment of the new atom. Here it is instructive to consider the variability of the existing parameters, which tend to be duplicated over various combinations of atoms. This step may also be required if old atom types are used in a new topological relation.

For example, consider the oxygen of a sulfoxide or a sulfone:

We would expect the van der Waals parameters of this oxygen to be similar to those of a carbonyl oxygen of the force field (type "O"):



or to those of carboxyl or phosphate oxygens (type "O2" in both examples)



because VDW radii are dominated by the number of electrons in an atom and are not very sensitive to chemical environment. In fact, the VDW parameters for types "O" and "O2" are identical. Can one of these types be used for the oxygen in sulfoxide/sulfone? The environment must now be considered. If no suitable analogy can be found, a new atom type must be created and a complete set of bond, angle, and dihedral parameters for its neighbors added to the force field. In this case, both sulfoxide and sulfone oxygens are substituents of tetrahedral sulfurs rather than trigonal planar carbons, so the phosphate oxygen case makes an appealing analogy. We then check whether any existing bond, angle or dihedral parameters involve a S="O2" bond, and if they do, we check that those parameters are appropriate for *this* case of an S=O bond. But there are no such parameters − it is therefore reasonable to extend the use of type "O2" for this case. (Had there been inappropriate S="O2" parameters, we might want to either make a new sulfur type, or make a new oxygen type starting with the VDW parameters of "O" and "O2".) We continue discussion of bond and angle parameters for these example fragments below.

van der Waals parameters. What if no atom type lends itself to adaptation? When creating a new type, the first thing one must consider is VDW parameters. As illustrated above, for organic compounds these parameters may be straightforward to find by analogy based on element and bond order alone. Monoatomic ions, however, do not present such analogies in the AMBER force field and are discussed in more detail as an example.

The shape of the VDW potential for a given atom type is specified in terms of the distance between two atoms of the same type at the minimum energy point. Half the interatomic distance at that point is treated as the basic radius, or R*, parameter for that type. The form for the radial potential for two atoms is the sum of the R* values of their types. The potential well depth ("e") of the minimum energy point between two atoms of the same type is combined with the potential of another atom type by taking the root of the product. (Other parametric forms can be used which tend to have different type-type "combining rules".)

The simplest approach to deriving VDW parameters is to match a relevant experimental determination of the size of the atom in question. One source of such measurements is diffraction data. The sum of metal and oxygen Pauling radii tends to be 3% smaller than indicated by water-ion neutron and X-ray diffraction data for Li+ and Na+ ions, 2% smaller than for K+, and 1% smaller than for Rb+ and Cs+ [126]. Another source of ion "size" information is crystallographic studies of ion complexes [127]. Since van der Waals parameters consist of two terms, the parameters that *e.g.* yield a given first peak of the radial distribution of the distance between two types of atoms are not unique. Another variety of experimental data that can contribute to parameterization is the free energy of solvation in water or another relevant solvent. However, it is still not clear whether the combination of experimental size and solvation free energy is sufficient to determine unique R* and "e" parameters for an atom in relation to an existing type. A further

complication arises because an atom type may come into contact with more than one other type, and nothing in principle guarantees that VDW parameters for a group of types can be fitted to yield uniformly correct pairwise potentials. Therefore it is important to choose parameters consistent with the most significant atom types that the new type will come in contact with. To a first approximation, atom types that tend to be oppositely charged, if present, are of most interest. In a particularly important case for ions, the TIP water models (as well as some other waters) have a spherical van der Waals potential centered on the water oxygen (type "OW"), which is somewhat inflated to enclose the hydrogen atoms in the molecule [37]. Thus a cation that has been parameterized to give a correct ion-"OW" radial distance distribution function in such a water model will be "smaller" and come in closer contact with neighboring atoms if it is bound in a molecule consisting of AMBER atoms.

Moreover, remembering that different pairwise combining rules are in use in the modeling community, parameters from one convention must be adapted to yield the same results for a given pair of types in another scheme. Thus it was necessary to adapt the monovalent cation parameters of Åqvist [35] (found in `parm94.dat` and `parm91.dat`) for AMBER so that the ion-water *combined* potential gave the same optimal distance as with the combining rules used by Åqvist. Matching the ion size in the environment seems to be sufficient in the case with small monoatomic *monocations*; the default has traditionally been to use a somewhat arbitrary well depth (epsilon) of 0.1 kcal/mol, characteristic of a rather nonpolarizable atom, and fit an R* parameter (see the Ross and Hardin reference). For the multivalent ions, different further approaches may be considered to capture the quasi-bonding electron mobility, including the use of explicit bonds or hydrogen bonding terms (see the Vedani and Huhta reference).

We have also discussed the derivation of van der Waals parameters for hydrogen in different bonding environments [128]. Using ab-initio calculations to study the interaction between water and various hydrogens, we found that a reduction in R* was required for hydrogens attached to carbons with adjacent electronegative atoms. This trend is nicely paralleled in the progressively smaller R* for hydrogens attached to carbon (HC), nitrogen (H), and water oxygen (HW).

Bond and angle parameters. Having chosen or created one or more atom types and sets of van der Waals parameters, the bond, angle and dihedral parameters must be created. Equilibrium bond lengths and angles may be obtained from tabulations of experimental data in the literature [129,130]. Initial bond and angle force constants may be chosen based upon analogy to similar parameters in the force field or using the method of Hopfinger and Pearlstein [131]. Cannon gives an example of extending the Weiner *et al.* force field to guanosine triphosphate and analogs [132]. The AMBER-1994 force field contains a limited number of unique bond and angle force constants and therefore selection by analogy is a feasible starting point. Returning to our sulfoxide/sulfone example, we find that the only existing bonds involving O2 are:

```
        Kbond    Rbond
C -O2   656.0    1.250       JCC,7,(1986),230; GLU,ASP
O2-P    525.0    1.480       JCC,7,(1986),230; NA PHOSPHATES
```

and it would be reasonable to use the O2-P force constant with a bond distance from the literature. Similarly, it would be reasonable to use the same angle bending parameters as for phosphates:

```
Ktheta(O2-P-O2) = Ktheta(O-S-O)
Ktheta(O2-P-OS) = Ktheta(R-S=O)
Ktheta(OS-P-OS) = Ktheta(R-S-R)
```

Unless only crude parameters are desired, one should check the force constants by means of normal mode calculations if spectroscopic measurements are available for comparison; such calculations on N-methylacetamide are described in the Weiner *et al.* JACS 1984 paper. The bond and angle parameters are the primary determinants of the high and middle frequency vibrational modes of a molecule. For applications which require the highly accurate reproduction of vibrational frequencies, it is necessary to use a force field which includes higher order terms (anharmonicity) and cross-terms. For modeling the structures and interactions of molecules which are not highly strained, however, the simple harmonic approximation used in AMBER appears to be quite adequate.

Dihedral parameters. The dihedral parameters, in conjunction with the atomic charges and van der Waals parameters, are the primary determinants of the relative conformational energies of a molecule. The AMBER parameters IDIVF, PK, PN, and PHASE are used to define the torsional potential energy function. Each bonded series of atoms I-J-K-L must have at least one set of these dihedral parameters in the force field (just as every bonded pair I-J or triplet I-J-K must have bond or angle parameters, except that for dihedrals multiple terms may be used). The torsional energy function formula is:

$$Etors = (PK / IDIVF) * (1 + \cos(PN * phi - PHASE))$$

Let us look at a few examples in order to illustrate the nature of the dihedral parameters. For our first example (Figure 1), if atoms J and K are sp3 carbons (type CT) as in the molecule ethane (H3C-CH3), then the intrinsic barrier to rotation about the J-K bond is on the order of 3 kcal/mol. We must decide whether to make this a generic potential for torsions about CT-CT

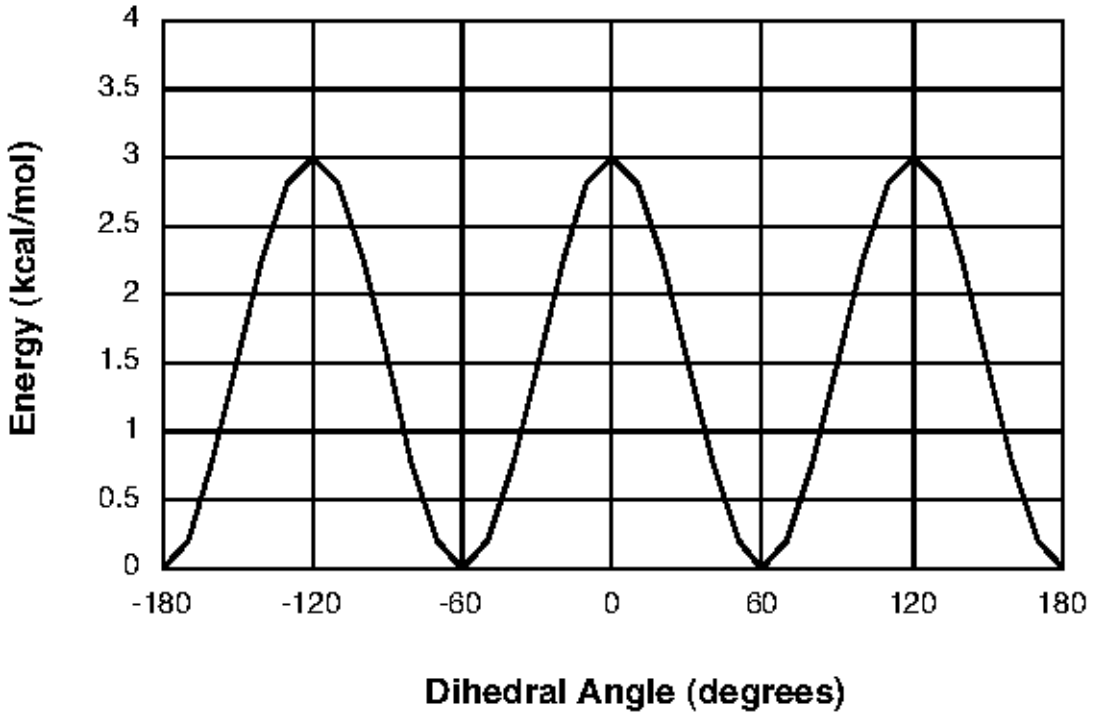


Figure 1: Ethane Rotational Barrier

bonds (X-CT-CT-X), or to make it explicit torsion restricted to HC substituents (HC-CT-CT-HC). This choice determines IDIVF, which is the total number of torsions about a single bond that the potential applies to. If all atoms are explicit, then IDIVF=1 and we divide the total potential for the bond (3.0 in this case) by the number of torsions involved; since each substituent 'sees' the opposite 3 substituents, there are 3x3=9 torsions around the bond, as would be the case whenever the central bond is between two sp3 atoms. If the generic representation is chosen, then the entire potential is used and IDIVF=9. PK is equal to one-half of the barrier magnitude and would therefore be equal to 3.0 / 2.0 = 1.5 kcal/mol for the generic case, or 3.0 / 9 / 2.0 = 0.1667 for the specific case. The topology about the dihedral of interest has a three-fold periodicity (PN); that is, there are three potential barriers as the C-C bond is rotated -180 to 180 degrees. These barriers occur when the methyl hydrogens eclipse each other: at 0, -120, and 120 degrees. Since the dihedral formula is a Fourier series truncated to a single cosine term, no phase shift would be needed to reproduce the potential energy barriers and PHASE = 0 degrees. (PHASE = 0 degrees if an energy *maximum* is at 0 degrees; PHASE = 180 degrees if an energy *minimum* is at 0 degrees.) So we have:

```
        PN    = 3
        PHASE = 0.0 degrees

    for HC-CT-CT-HC:

        IDIVF = 1
        PK    = 3.0 kcal/mol / 9 / 2.0 = 0.1667 kcal/mol

    for X -CT-CT-X :

        IDIVF = 9
        PK    = 3.0 kcal/mol / 2.0 = 1.5 kcal/mol
```

These same torsional parameters can be used for n-butane, and the results are in good agreement with experiment and higher-level calculations for the relative energy of *trans* and *gauche* minima and *cis* and *skew* energy barriers.

Consider now the molecule ethylene, H2C=CH2, whose dihedral potential energy is shown in Figure 2.

The lowest-energy conformation of this molecule is planar with a two-fold (PN = 2), 60 kcal/mol barrier to rotation about the C=C bond. The barriers are found at dihedral angles of -90 and 90 degrees (energy minimum at 0 degrees), and can be reproduced by the truncated Fourier series only if a phase shift of 180 degrees (PHASE = 180.0 degrees) is used.

```
        PN    = 2
        PHASE = 180.0

    specific:

        IDIVF = 1
        PK    = 60.0 kcal/mol / 4 / 2.0 = 7.5 kcal/mol

    generic:

        IDIVF = 4
        PK    = 60.0 kcal/mol / 2.0 = 30.0 kcal/mol
```

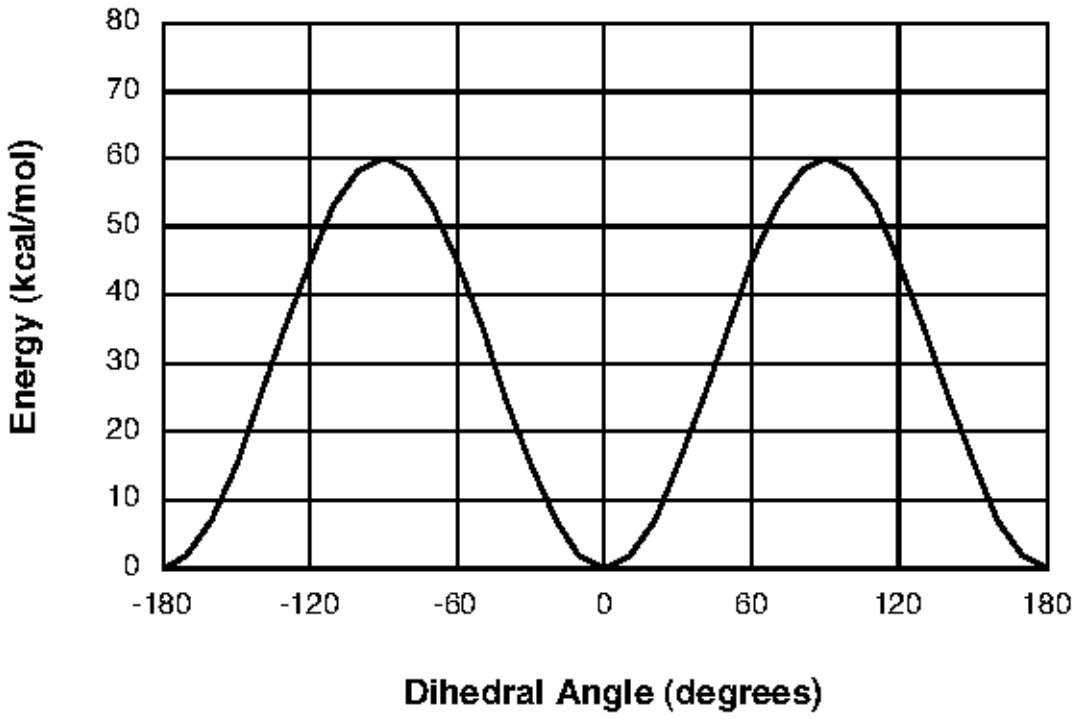


Figure 2: Ethylene Rotational Barrier

Finally, we examine a hypothetical molecule ZH2C-CH2Z, where Z represents an electronegative functional group. Let us imagine that we either have experimental data on the relative conformational energies or we have simulated the rotational potential of this molecule with a series of quantum mechanical calculations. In practice, this is only done for minimum and maximum energy conformations − *trans*, *gauche+*, *gauche-*, *eclipsed*, *skew*, etc. In our example, the energy profile shows that the *trans* conformation (Z-C-C-Z = 180 degrees) is about 0.5 kcal/mol less stable than the *gauche*.

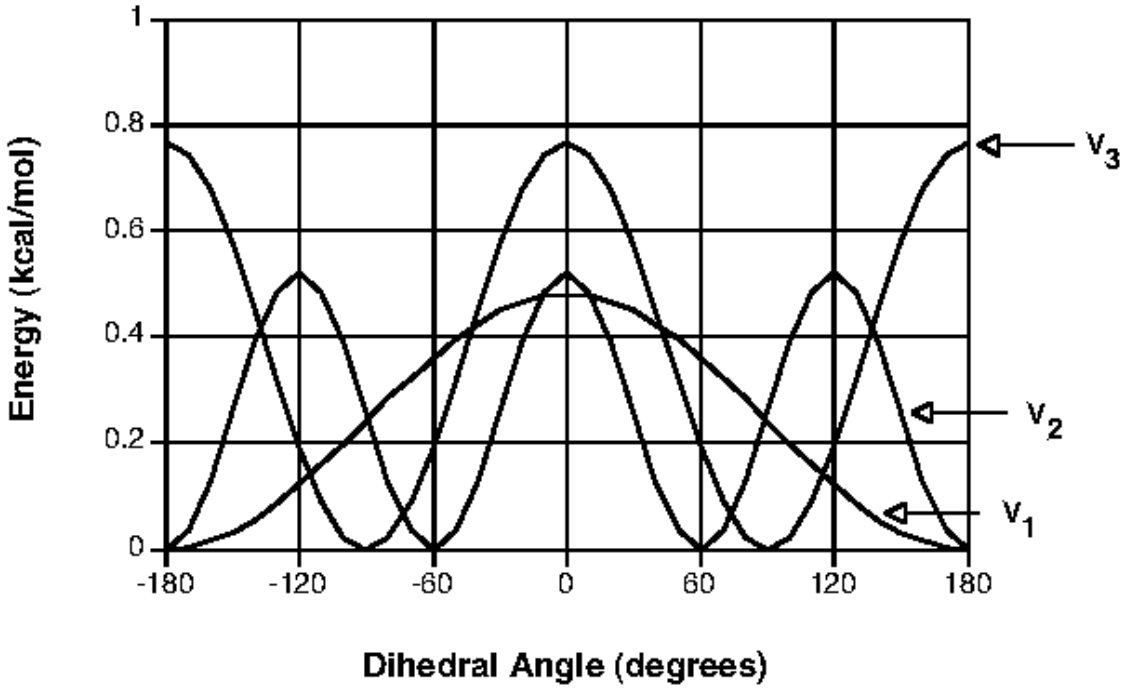
Before fitting the torsional parameters, we must generate the energy profile for the molecular mechanical nonbonded potential as was done for the quantum potential, subtract this curve from the quantum curve, and fit the torsional potential to the difference potential.

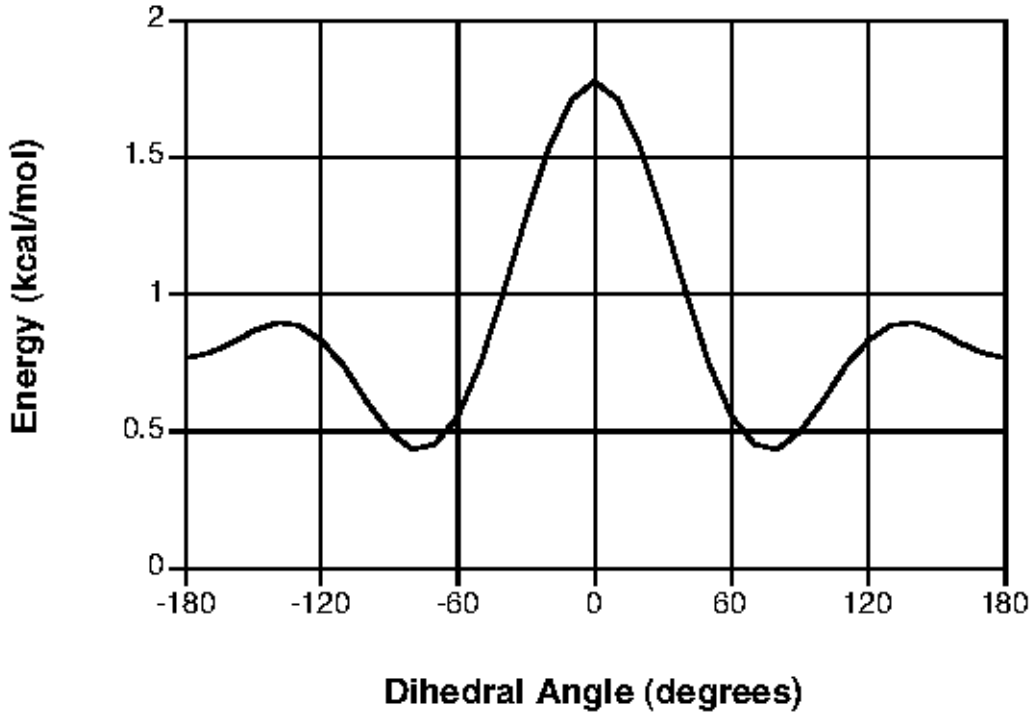
Before these calculations can be done, atomic charges need to be calculated, also by fitting to quantum mechanical results. The difference potential is then deconvoluted into Fourier series terms (Figure 3) which give the force field parameters:

	IDIVF	PK	PHASE	PN
Z-CT-CT-Z	1	0.260	0	-3
Z-CT-CT-Z	1	0.384	0	-2
Z-CT-CT-Z	1	0.241	0	1

which result in the total torsional potential shown in Figure 4. (In AMBER, PN is set to less than zero when additional terms remain to be read.)

Care must be taken when deconvoluting the torsional potential not to introduce spurious minima or maxima into the rotational energy profile. The combined potential of the deconvoluted





Figures 3 and 4

parameters can be plotted directly by a graphing program, or the torsional energy profile can be

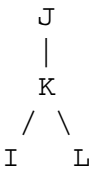
"empirically" generated at 20-30 degree intervals in AMBER.

In practice, such an elaborate Fourier series treatment may not be appropriate because (a) the quantum mechanical treatment may not be accurate enough to warrant it, (b) one would rather have a simpler torsional potential that is more consistent with the existing force field and (c) the electrostatic potential fitting procedure may capture the torsional energy profile well enough so that many terms are not needed. For example, in the case of 1,2-difluoroethane, the known gauche tendency of the fluorines can be simulated by adding a twofold torsion

	IDIVF	PK	PHASE	PN
F-CT-CT-F	1	X	0	2

with "X" adjusted to make the total molecular mechanical energy of the *gauche* conformation 1 kcal/mol lower than the *trans* conformation.

Improper torsions. Improper torsions are so named because the atoms involved are not serially bonded; rather they are branched:



Improper I-J-**K**-L

The convention is that the central atom is listed in the third position of the dihedral ("K" in the figure). Improper dihedral potentials are sometimes necessary to reproduce out-of-plane bending frequencies, *i.e.* they keep four atoms properly trigonal planar for a two-fold torsional potential (PN=2). They are additionally used in the united-atom force field model when a carbon with an implicit hydrogen is a chiral center; in effect they keep the position from inverting (PN=3).

The PHASE for improper torsions is always 180 degrees. Improper torsional parameters listed in the force field file can use wild-card specifications ("X") for the non-central atoms (note that wild-card impropers must follow the explicit ones in the parm.dat force field file). In LEaP, every atom with three substituents is matched against the impropers in the force field file, and all matches are applied (discarding any wild-card terms if an explicit match is found). Thus care must be taken that a new improper does not inadvertently match other cases. In both PLEP and LEaP, an improper with no wild cards causes all wild-card-containing impropers to be ignored. Except for not mixing wild-card with explicit cases, all improper terms that match a given central atom are applied. In LEaP, if no match is found, no improper term is applied (unlike bonds, angles and "proper" torsions, for which parameters *must* exist).

Hydrogen bonding parameters. Unlike the previous AMBER force fields, the 1994 force field does not include a 10-12 hydrogen bonding function. This function, however, is still supported by the software. When using the hydrogen bonding function, all relevant pairs of atom types need to have parameters. Note that if a pair of atom types has H-bond (10-12) parameters, these will override any van der Waals (6-12) parameters for that pair.

12.2. Charge Fitting Philosophy

Wendy Cornell

The philosophy of the Kollman group (AMBER) has been that the accurate representation of electrostatic interactions is crucial for a force field intended for application to biological molecules [124]. We note that the choice of a particular force field should depend on the system properties one is interested in. Some applications require more refined force fields than others. Moreover, there should be a balance between the levels of accuracy or refinement of different parts of a molecular model. Otherwise the computing effort put into a very detailed and accurate part of the calculations may easily be wasted due to the distorting effect of the cruder parts of the model.

The new charges which were developed for the 1994 force field are called RESP charges, for Restrained ElectroStatic Potential fit. This modification of the original ESP method was developed by Christopher Bayly [122,123]. The basic idea with electrostatic potential fit charges is that a least squares fitting algorithm is used to derive a set of atom-centered point charges which best reproduce the electrostatic potential of the molecule. In the AMBER charge fitting programs, the potential is evaluated at a large number of points defined by 4 shells of surfaces at 1.4, 1.6, 1.8, and 2.0 times the VDW radii. These distances have been shown to be appropriate for deriving charges which reproduce typical intermolecular interactions (energies and distances). The dipole moment of the molecule is well reproduced.

Other programs have embedded the molecule in a cubic grid of points to evaluate the potential. We believe that assigning the points along the contours of the molecule provides a reasonable sampling of the ESP around each atom.

The value of the electrostatic potential at each grid point is calculated from the quantum mechanical wavefunction. The charges derived using this procedure are basis set dependent. For example, the Weiner *et al.* force field employs STO-3G based charges, whereas the new Cornell *et al.* 1994 force field uses charges derived using the 6-31G* basis set. The 6-31G* basis set is bigger and, for the most part, "better." Because quantum mechanics calculations scale as the number of basis functions to about the 2.7 power (HF as implemented in Gaussian92), the bigger 6-31G* basis set was prohibitively large for use in developing the earlier 1984/1986 force field.

The 6-31G* basis set tends to result in dipole moments which are 10-20% larger than gas phase. This behavior is desirable for deriving charges to be used for condensed phase simulations within an effective two-body additive model, where polarization is being represented implicitly. In other words a molecule is expected to be more polarized in condensed phase vs. gas phase due to many body interactions, so we "pre-polarize" the charges.

A study by St-Amant *et al.* calculated DFT charges for a number of small molecules and found them to be smaller than HF/6-31G* derived ones [133]. DFT charges for methanol did not reproduce the relative free energy of solvation of methanol. Such charges may be more appropriate for use with a non-additive model, since the DFT model reproduced the gas phase dipole moments very well.

ESP fit charges have many advantages. They reproduce interaction energies well. They can be calculated in a straightforward fashion. They have been shown to perform well at reproducing

conformational energies when used with an appropriate 1-4 electrostatic scale factor. The Cornell *et al.* JACS paper provides much of the validation of our new charge model. A study by Howard, Cieplak, and Kollman [134] showed how ESP and RESP charges performed quite well at modeling the conformational energies of a series of 1,3-dioxanes.

It should be noted that Mulliken charges do NOT reproduce the electrostatic potential of a molecule very well. Mulliken charges are calculated by determining the electron population of each atom as defined by the basis functions. When the density is associated with the square of a single basis function, that density is assigned to the atom associated with that basis function. Similarly, if the density is associated with 2 basis functions which are on a common atom, the density is assigned to that atom. The ambiguity arises when the density is associated with 2 basis functions lying on different atoms. In that case the density is partitioned equally onto each atom.

12.3. A brief introduction into implicit solvation methodology

Alexey Onufriev

An accurate description of the aqueous environment is essential for realistic biomolecular simulations, but may become very expensive computationally. For example, an adequate representation of the solvation of a medium-size protein typically requires thousands of discrete water molecules to be placed around it. An alternative which is becoming more and more popular is based on replacing a real water environment consisting of discrete molecules by "virtual water" -- an infinite continuum medium with the dielectric and "hydrophobic" properties of water.

These continuum "virtual water", or "implicit solvent" models have several advantages over the explicit water representation, especially in molecular dynamics simulations:

(1) The computational cost associated with the use of such models in simulations is generally considerably smaller than the cost of representing water explicitly.

(2) The models describe instantaneous solvent dielectric response which eliminates the need for the lengthy equilibration of water that is typically necessary in explicit water simulations.

(3) Due to the absence of viscosity associated with the explicit water environment, the molecule can more quickly explore the available conformational space.

(4) The continuum model corresponds to solvation in an infinite volume of solvent, thereby avoiding possible artifacts of the replica interactions that occur in the periodic systems typically used for explicit water calculations.

(5) Since solvent degrees of freedom are taken into account implicitly, estimating energies of solvated structures is much more straightforward than with explicit water models.

A neophyte to the field of molecular modeling should be properly warned at this point: all of the attractive features of the "virtual water"/continuum solvent methodology listed above come at a price of making a number of approximations whose effects are often hard, if not impossible, to estimate. Note that the *discrete* $\rightarrow$ *continuum* step is not the only deviation from reality: for example, the present day implicit solvent models also suffer from the generic inadequacies of "mean-field" type theories, as correlations in the solvent are ignored. Also note that some familiar descriptors of molecular interaction, such as solute--solvent hydrogen bonds, are no longer explicitly present in the model -- instead, they come in implicitly, in the mean-field sense via a linear dielectric response, and contribute to the overall solvation energy. However, despite the fact that the methodology represents an approximation at a fundamental level, it has in many cases been successful in calculating various macromolecular properties [9,108,135-139].

In many molecular modeling applications, and especially in molecular dynamics (MD), the key quantity that needs to be computed is the total energy of the molecule in the presence of solvent. This energy is a function of molecular configuration, its gradients with respect to atomic positions determine the forces on the atoms. The total energy of a solvated molecule can be written as $E_{tot} = E_{vac} + \Delta G_{solv}$, where E_{vac} represents a molecule's energy in vacuum (gas-phase), and ΔG_{solv} is the free energy of transferring the molecule from vacuum into solvent, *i.e.* solvation free energy. In what follows, we assume that E_{vac} is given by a classical potential function, or force-field, that breaks the interaction down into various physical components, such as bond and angle stretching, torsional twist, and VDW and Coulomb interactions between its atoms; details of the AMBER force-field are discussed elsewhere in this manual.

To estimate the total solvation free energy of a molecule, ΔG_{solv}, one typically assumes that it can be decomposed into the "electrostatic" and "non-electrostatic" parts:

$$\Delta G_{solv} = \Delta G_{el} + \Delta G_{nonel},$$

where ΔG_{nonel} is the free energy of solvating a molecule from which all charges have been removed (i.e. partial charges of every atom are set to zero), and ΔG_{el} is the free energy of first removing all charges in the vacuum, and then adding them back in the presence of a continuum solvent environment. The above decomposition, which is yet another approximation, is the basis of the widely used MMPB/SA scheme [111]. Generally speaking, ΔG_{nonel} comes from the combined effect of two types of interaction: the favorable van der Waals attraction between the solute and solvent molecules, and the unfavorable cost of breaking the structure of the solvent (water) around the solute. Within the PB/SA, ΔG_{nonel} is taken to be proportional to the total solvent accessible surface area (SA) of the molecule, with a proportionality constant derived from experimental solvation energies of small non-polar molecules. Note that $\Delta G_{nonel} \approx \sigma SA$ is an approximation, but arguably not the most critical one in the hierarchy of assumptions that form the foundation of the implicit solvent methodology [140]. Currently, AMBER follows this simple way to compute ΔG_{nonel}, and uses a fast LCPO algorithm to compute an analytical approximation to the surface accessible area of the molecule. This part is relatively straightforward, and is not the bottleneck of a typical MD simulation. The most time consuming part is the computation of the electrostatic contribution to the total solvation free energy, mainly because the forces involved are long-ranged (the Coulomb potential of a point charge scales as r^{-1} with distance). A numerically exact, within the framework of the continuum model, way to compute the electrostatic potential $\phi(\mathbf{r})$ produced by molecular charge distribution $\rho_m(\mathbf{r})$, is based on the PB approach in which the following equation (or its equivalent) must be solved; for simplicity we give its linearized form:

$$\nabla\varepsilon(\mathbf{r})\nabla\phi(\mathbf{r}) = -4\pi\rho_m(\mathbf{r}) + \kappa^2\varepsilon(\mathbf{r})\phi(\mathbf{r}).$$

Here, $\varepsilon(\mathbf{r})$ represents the position-dependent dielectric constant which equals that of bulk water far away from the molecule, and is expected to decrease fairly rapidly across the solute/solvent boundary. The electrostatic screening effects of (monovalent) salt enter via the second term on the right-hand side, where the Debye-Huckel screening parameter $\kappa \approx 0.1\text{Å}^{-1}$ at physiological conditions. Once the potential $\phi(\mathbf{r})$ is computed, the electrostatic part of the solvation free energy is given by $\sum_i q_i[\phi(\mathbf{r}_i) - \phi(\mathbf{r}_i)|_{vac}]$, where q_i are the partial atomic charges at positions $\mathbf{r}_i$ that make up the molecular charge density $\rho_{m(}\mathbf{r}) = \sum_i \delta(\mathbf{r} - \mathbf{r}_i)$, and $\phi(\mathbf{r}_i)|_{vac}$ is the electrostatic potential computed for the same charge distribution in the absence of the dielectric boundary, *e.g.* in vacuum. In this very brief discussion we omit the details of the numerical procedures for solving the PB equation along with all of the related technical issues, such as handling of the Coulombic singularities at $\mathbf{r} = \mathbf{r}_i$; the reader is encouraged to consult recent literature on the subject, *e.g.* Ref. [140] and references therein.

The Poisson-Boltzmann approach as described above has traditionally been used in calculating electrostatic properties of "static" configurations (snapshots), as in estimates of pK_a of titratable groups in proteins [141]. However, in molecular dynamics applications, the associated computational costs are often very high, as the Poisson-Boltzmann equation needs to be solved every time the conformation of the molecule changes. AMBER developers have pursued an alternative approach to obtain a a reasonable, computationally efficient estimate of ΔG_{el} to be used in molecular dynamics simulations. The analytic generalized Born (GB) method is an approximate, relative to the PB treatment, way to calculate the electrostatic part of the solvation free energy, ΔG_{el}. The methodology has become popular [46,56,60,135,142-145], especially in molecular dynamics

applications [8,43,146,147] due to its relative simplicity and computational efficiency, compared to the more standard numerical solution of the Poisson-Boltzmann equation. Within AMBER GB models, each atom in a molecule is represented as a sphere of radius ρ_i with a charge q_i at its center; the interior of the atom is assumed to be filled uniformly with a material of dielectric constant of 1. The molecule is surrounded by a solvent of a high dielectric ε_w (80 for water at 300 K). Note that 1 -- the default in sander -- is the value of the dielectric constant of the molecular interior that should be used in molecular dynamics simulations based on current AMBER forcefields. However, if the GB or PB model is employed for analysis of "static" snapshots, as in *pK* calculations, a different value for the dielectric constant may be appropriate [141], such as 4. This higher value takes into account, albeit very approximately, the polarizability of the molecular interior; in an MD simulation, it comes in naturally through dynamic fluctuations. The GB model approximates ΔG_{el} by an analytical formula [46,58],

$$\Delta G_{el} \approx \Delta G_{gb} = -\tfrac{1}{2} \sum_{ij} \frac{q_i q_j}{f_{GB}(r_{ij}, R_i, R_j)} \left(1 - \frac{e^{-\kappa f_{gb_{ij}}}}{\varepsilon_w}\right)$$

where r_{ij} is the distance between atoms i and j, the R_i are the so-called *effective Born radii* of atoms i and j, and f_{GB} is a certain smooth function of its arguments. A common choice [46] of f_{GB} is

$$f_{GB} = \left[r_{ij}^2 + R_i R_j \exp(-r_{ij}^2 / 4 R_i R_j) \right]^{\frac{1}{2}}$$

although other other expressions have been tried [144,148]. The effective Born radius of an atom reflects the degree of its burial inside the molecule: for an isolated ion, R_i is equal to its van der Waals (VDW) radius ρ_i. Then one obtains the particularly simple form:

$$\Delta G_{el} \approx -\tfrac{1}{2} (1 - \frac{1}{\varepsilon_w}) \frac{q^2}{\rho} \approx 166 \frac{q^2}{\rho} [kcal/mol]$$

where we assumed $\varepsilon_w = 80$ and $\kappa = 0$ (pure water). The reader can recognize the famous expression due to Born for the solvation energy of a single ion. For simple monovalent ions, substituting $q = 1$ and $\rho \approx$ 1.5 Å, yields $\Delta G_{el} \approx -100$ *kcal/mol*, in reasonable agreement with experiment. This type of calculation was probably the first success of the implicit solvent model based on continuum electrostatics. The function f_{GB} is designed to interpolate, in a clever manner, between the limit $r_{ij} \to 0$ when atomic spheres merge into one, and the opposite extreme $r_{ij} \to \infty$ when the ions can be treated as point charges obeying the Coulomb's law [60] For deeply buried atoms, the effective radii are large, $R_i \gg \rho_i$, and for such atoms one can use a rough estimate $R_i \approx L$, where L is the distance from the atom to the molecular surface. Closer to the surface, the effective radii become smaller, and for a completely solvent exposed side-chain one can expect R_i to approach ρ_i. The principle way to compute an effective radius is to invert the Born formula:

$$R_i^{-1} = 2\Delta G_{el} q_i^{-2} (1 - \varepsilon_w^{-1})^{-1}$$

which shifts the computational burden to finding a good estimate of ΔG_{el} for each atom. Classical electrostatics [149] relates ΔG_{el} to a volume integral of electric field density $(1/8\pi)\mathbf{D}^2$. Note that the effective radii depend on the molecule's conformation, and so have to be re-computed every time the conformation changes. This makes the computational efficiency a critical issue, and various approximations are normally made that facilitate an effective estimate of ΔG_{el}. In particular, the so-called *Coulomb field approximation*, or *CFA*, is often used, which replaces the true electric displacement, $\mathbf{D}^{true}$, around the atom by the Coulomb field $\mathbf{D}_i^0(\mathbf{r}) \equiv (q_i / r^3)\mathbf{r}$. Within this assumption, the following expression for R_i can be derived [60]:

$$R_i^{-1} = \rho_i^{-1} - \frac{1}{4\pi} \int\limits_{solute} \theta(|\mathbf{r}| - \rho_i) \frac{1}{r^4} d^3\mathbf{r}.$$

where the integral is over the solute volume surrounding atom i. It was also shown [46] that using slightly reduced values of $\tilde{\rho}_i = \rho_i - 0.09$ Å, instead of ρ_i gives, for small molecules, a better agreement with the corresponding PB calculations based on ρ_i. For a realistic molecule, the solute boundary (molecular surface) is anything but trivial, and so further approximations are made to obtain a closed-form analytical expression for the above equation, *e.g.* the so-called pairwise de-screening approach of Hawkins, Cramer and Truhlar [53], which leads to a GB model termed GB^{HCT}. The electrostatic screening effects of (monovalent) salt are incorporated [58] via the Debye-Huckel screening parameter $\kappa \text{Å}^{-1} \approx 0.316\sqrt{[salt][mol/L]}$. Note, that within the implicit solvent framework, one does not use explicit counter-ions (*e.g.* Na^+ and Cl^-) unless these are known to remain "fixed" in specific positions around the solute, as in the case of Mg^{++} and the DNA [43]. To set-up an MD simulation based on an implicit solvation model one requires a set of atomic radii ρ_i, which is an extra set of input parameters compared to the explicit solvent case. Over the years, a number of slightly different radii sets have been proposed, each being optimal for some class of problems, *e.g.* for computing solvation free energies of molecules. A good set is expected to perform reasonably well in different types of problems, that is to be transferable. One example is the Bondi radii set originally proposed in the context of geometrical analysis of macromolecular structures, but later found to be useful in continuum electrostatics models as well.

Since the GB model shares the same underlying physical approximation -- continuum electrostatics -- with the Poisson-Boltzmann approach, it is natural, in optimizing the GB performance, to use the PB model as a reference. In particular, it was recently shown [148] that a very good agreement between the GB and PB models can be achieved if the effective Born radii match those computed exactly using the PB approach. Therefore, by improving the way the effective radii are computed within the analytic generalized Born, one can improve accuracy of the GB model. However, agreement with PB calculations is not the only criterion of optimal GB performance; other tests include comparisons to explicit solvent simulations results, and to experiment. Also, to ensure computational efficiency in molecular dynamics simulations, the analytical expressions used to compute the effective Born radii must be simple enough, and the resulting electrostatic energy component ΔG_{el} "well-behaved", so as not to cause any instabilities in numerical integration of Newton's equations of motion. The versions of the GB model currently available in AMBER reflect various stages of the evolution of the approach guided by the principles outlined above; a relatively thorough accuracy analysis of these and other popular GB models and their performance comparison can be found in Ref [61]. Historically, the first GB model to appear in AMBER was GB^{HCT}, it corresponds to {igb=1} input parameter. It was later found that the model worked well on small molecules, but not so well on macromolecules, relative to the PB treatment. Note that in the GB^{HCT}, the 3D integral used in the estimation of the effective radii, is performed over the van der Waals (VDW) spheres of solute atoms, which implies a definition of the solute volume in terms of a set of spheres, rather than the complex molecular surface [150], commonly used in the PB calculations. For macromolecules, this approach tends to underestimate the effective radii for buried atoms [60], arguably because the standard integration procedure treats the small vacuum--filled crevices between the van der Waals (VDW) spheres of protein atoms as being filled with water, even for structures with large interior [148]. This error is expected to be greatest for deeply buried atoms characterized by large effective radii, while for the surface atoms it is largely canceled by the opposing error arising from the Coulomb approximation, which tends [54,56,142] to overestimate R_i.

The deficiency of the GB^{HCT} model described above can, to some extend, be corrected by noticing that even the optimal packing of hard spheres, which is a reasonable assumption for biomolecules, still occupies only about 3/4 of the space, and so "scaling-up" of the integral by a factor of $\approx 4/3$ should effectively increase the underestimated radii by about the right amount, without any loss of computational efficiency. This idea was developed and applied in the context of pH titration [60], where it was shown to improve the performance of the GB approximation in calculating pK_a values of protein sidechains. This simple correction did not, however, perform as well in molecular dynamics where it led to numerical instabilities. The main difficulty was that small changes in conformation could sometimes lead to large alterations of effective radii, requiring very careful MD simulations with extremely short time-steps. The problem also exists, but is less pronounced, with the GB^{HCT} model where the R_i were routinely underestimated, and as a consequence, were less sensitive to structural variations. Furthermore, the one-parameter correction introduced in Ref. [60] was not optimal in keeping the GB^{HCT} model's established performance on small molecules. It was therefore proposed [8] to re-scale the effective radii with the re-scaling parameters being proportional to the degree of the atom's burial, as quantified by the value I of the 3D integral. The latter is large for the deeply buried atoms and small for exposed ones. Consequently, one seeks a well-behaved re-scaling function, such that $R_i \approx (\tilde{\rho}_i^{-1} - I)^{-1}$ for small I, and $R_i > (\tilde{\rho}_i^{-1} - I)^{-1}$ when I becomes large. One would also want to have a "smooth" upper bound on R_i vs. I to ensure numerical stability, see below. While there is certainly more than one way to satisfy these constraints, the following simple, infinitely differentiable re-scaling function was chosen to replace the GB^{HCT} original expression for the effective radii:

$$R_i^{-1} = \tilde{\rho}_i^{-1} - \rho_i^{-1} \tanh\left(\alpha\Psi - \beta\Psi^2 + \gamma\Psi^3\right)$$

where $\Psi = I\tilde{\rho}_i$, and α, β, γ are treated as adjustable dimensionless parameters which were optimized using the guidelines mentioned earlier (primarily agreement with the PB). Currently, AMBER supports two GB models (termed GB^{OBC}) based on this idea. These differ by the values of α, β, γ, and are invoked by setting igb to either igb=2 or igb=5. The details of the optimization procedure and the performance of the GB^{OBC} model relative to the PB treatment and in MD simulations on proteins is described in Ref. [8]; an independent comparison to the PB in calculating the electrostatic part of solvation free energy on a large data set of proteins can be found in [61].

12.4. Locally enchanced sampling

Carlos Simmerling

At room temperatures, normal nanosecond length molecular dynamics simulations have difficulty overcoming barriers to conformational transitions and may only sample conformations in the neighborhood of the initial structure. Among the various techniques to enhance sampling during a simulation, Locally Enhanced Sampling (hereafter called LES) stands out as a promising strategy [151]. This mean-field technique allows the selective application of additional computational effort to a portion of the system, increasing the sampling of the region of interest. The enhanced sampling is achieved by replacing the region(s) of interest with multiple copies. These copies are constructed in a special way- they do not interact with each other, and interact with other LES regions and the rest of the system in an average way. This average is an average force or energy from all of the individual copy contributions, not one force or energy from an average conformation of the copies. A key feature is that the energy function is modified such that the energy is identical to that of the original system when all LES copies have the same coordinates.

During the simulation, the copies are free to move apart and explore different regions of conformational space, thereby increasing the statistical sampling. The amount of copy independence is an issue that will be discussed in further detail below. In practical terms, this means that one can obtain multiple trajectories for the region of interest while carrying out only a single simulation. If the LES region is a small part of the system (such as a peptide in solution, or a loop in a protein), then the additional computational effort from the added LES particles will be a small percentage of the total number of atoms, and the multiple trajectories will be obtained with a small additional computational effort. Perhaps the most useful feature of the LES method is that it has been shown [152] that the barriers to conformational transitions in a LES system are reduced as compared to the original system, resulting in more frequent conformational changes. This can be rationalized with a simple model: imagine a protein side-chain that has been replaced with 2 copies. At finite temperatures, these copies will have different conformations. Now consider the interaction of another part of the system with this region. Previously, steric conflicts or other unfavorable interactions may have created high barriers. Now, however, the rest of the system sees each of these 2 copies with a scaling factor of ½. If one copy is in an unfavorable conformation, the other may not be, and the effective barriers with a distribution of copies is less than with the single copy. Another way to consider the LES copies is that they represent an intermediate state between a *normal* simulation where each point in time represents a single structure, and a purely continuum model where the probability distribution of regions of interest are represented by a continuous function. The atoms outside a LES region interact with that region as if it were (in the limit of many copies) a continuum, with a probability scaling given to all interactions. Therefore, the most unfavorable interactions are reduced in magnitude as compared to the original system.

Another key feature of the LES system is that the global energy minimum occurs when all copies occupy the position of the global energy minimum in the original system [152]. This means that optimization of the LES system directly provides information about the original system without complicated mapping procedures. This can also be imagined with a simple model: imagine a system where one region is replaced with two copies. When the copies are together, the energy is identical to the corresponding non-LES structure. Now move one copy, and the energy may go up or down. The copies are independent and have no direct interaction (the energy

function looks like a sum of terms for each LES copy, with no terms involving both). Therefore, if the energy after moving one copy went down, it will go down even more if the second copy is moved in the same way. In other words, the energy sum related to one of the copies must always be less than or equal to the other copies. Any configuration with copies separated must be higher in energy than a single copy system corresponding to just one of the copies. This means that the global energy minimum of the LES system must have all of the copies in the same location, and each of these LES configurations has energy identical to the corresponding non-LES system, so the lowest of these must be the original global energy minimum. This argument can be extended to any number of copies and any number of LES regions.

Another way to envision the equivalence of global energy minima is through the following *experiment*. Imagine we have a molecule in the global energy minimum conformation. We now replace one region with multiple copies in the same conformation, which does not change the energy. Now, we move one copy, leaving the rest of the system unchanged, and propose that the energy might go down. We could then move the other copy to the same location and gain even further reduction in energy. Next, we remove the extra copy (go to a non- LES system) leaving the energy unchanged. We are therefore in a position other than the global energy minimum with energy lower than the global minimum. This contradiction shows that our assumption that it was possible to move a LES copy out of the non-LES global energy minimum conformation and obtain a lower energy was incorrect. This equivalence of global energy minima not only makes the mapping of results simple, but provides a critical self-consistency check for optimization using LES. For example, LES can be combined with optimization techniques such as simulated annealing (SA). At the end of a typical non-LES SA run, a single result is obtained. In order to evaluate the convergence of the SA run, one must repeat the simulation with slower cooling, an alternate initial structure, or another way to assess the result. With LES, however, one can simply look at the final distribution of copies. If the copies are not in the same locations, then either degenerate solutions exist (which can be tested by evaluating non-LES versions of each of the copy conformations), or convergence was not achieved (since this type of configuration cannot be the global minimum). Of course, convergence of copies does not guarantee that global minimum was found, but this is still an extremely valuable property of optimization using LES. Other methods exist that also provide a smoothing of the potential energy surface. However, the property of direct mapping of global energy minimum is an attractive feature of LES. Another major advantage of LES over alternate methods to reduce barriers or improve sampling is that it is compatible with current state-of-the-art simulation techniques such as molecular dynamics in explicit aqueous solvation (problems for techniques such as Monte Carlo or Genetic Algorithms) and the Particle Mesh Ewald technique for accurate treatment of long-range electrostatic interactions [103,153-155]. Higher temperatures can increase rates of barrier crossing, but one is then faced with issues related to solvent behavior at higher temperatures, maintaining proper densities and pressures, stability of the molecule of interest at the elevated temperature, and so on. LES gives more direct control over which regions should be enhanced, and also provides other benefits such as improvement in statistical sampling discussed above.

12.5. Poisson-Boltzmann dynamics

Ray Luo

Solvent interactions, especially solvent-mediated dielectric screening and Debye-Huckel screening, are thought to be one of the essential determinants of the structure and function of proteins and nucleic acids [108]. Ideally, one would like to provide a detailed description of solvent through explicit simulation of a large number of solvent molecules. This approach is frequently used in molecular dynamics simulations of solution systems. In many applications, however, the solute is the focus of interest, and the detailed properties of the solvent are not of central importance. In such cases, a simplified representation of the solvent, based on an approximation of the mean-force potential for the solvation interactions, can be employed to accelerate the computation. The mean-force potential averages out the degrees of freedom of solvent molecules, so that they are often called implicit or continuum solvents.

The Poisson-Boltzmann (PB) solvents are a class of widely used implicit solvents [156,157]. In these models, a solute is represented by an atomic-detail model as in a molecular mechanics force field, while the solvent molecules and any dissolved electrolyte are treated as a structureless continuum. The solute intramolecular interactions are computed by the usual molecular mechanics force field terms, while the solute-solvent and solvent-solvent interactions are computed by a mean-field approximation through the use of the PB electrostatic theory. The electrostatic model represents the solute as a dielectric body whose shape is defined by atomic coordinates and radii [158]. The solute contains a set of charges that produce an electrostatic field in the solute region and the solvent region. The electrostatic fields in such a system, including the solvent reaction field and the Coulombic field, may be computed by solving the PB equation [159,160]. The nonpolar solvation interactions are typically modeled with a term proportional to the solvent accessible surface area. This class of solvent models has been demonstrated to be reliable in reproducing the energetics and conformations as compared with explicit solvent simulations and experimental measurements for a wide range of systems.

The formalism with which a PB solvent can be applied in molecular mechanics simulations is based on rigorous foundation in statistical mechanics, at least for additive molecular mechanics force field. PB solvents approximate the solvent-induced electrostatic mean-force potential by computing the reversible work done in the process of charging the atomic charges in a solute molecule as $1/2 \sum_j Q_j \phi_j$ with j running over all charged atoms. The electrostatic potential ϕ_j at atomic charge site is computed by solving the PB equation:

$$\nabla \varepsilon(\mathbf{r}) \nabla \phi(\mathbf{r}) \; = \; -4\pi \rho(\mathbf{r}) - 4\pi \sum_i z_i \exp(-z_i \phi(\mathbf{r})/k_B T)$$

where $\varepsilon(\mathbf{r})$ is the dielectric constant, $\phi(\mathbf{r})$ is the electrostatic potential, $\rho(\mathbf{r})$ is the solute charge, z_i is the charge of ion type i, c_i is the number density of ion type i far from the solute, k_B is the Boltzmann constant, and T is temperature; the summation is over all different ion types. In this release, only the linearized form of the PB equation is used.

Many numerical methods may be used to solve the linearized PB equation. The finite-difference Poisson-Boltzmann (FDPB) method is one of the most popular methods in computational studies of biomolecules [161-163]. It involves the following steps: mapping atomic charges to the finite difference grid points (termed grid charges below); assigning non-periodic boundary conditions, i.e. electrostatic potentials on the boundary surfaces of the finite difference grid; and

applying a dielectric model to define the boundary between high-dielectric (i.e. water) and low-dielectric (i.e. solute interior) regions [164].

These steps allow the partial differential equation to be converted into a linear system A x = b with the electrostatic potential on grid points, x as unknowns, the charge distribution on the grid points as the source b, and the dielectric constant on the grid edges and salt-related terms wrapped into the coefficient matrix A, which is a seven-banded symmetric matrix. Once the linear system is solved, the solution is used to compute the electrostatic energies and forces.

It has been shown that the total electrostatic energy of a solute molecule can be approximated through the FDPB approach by subtracting the self FD Coulombic energy ($G_{FD,coul,self}$) and the short-range FD Coulombic energy ($G_{FD,coul,short}$) from the total FD electrostatic energy, and adding back the analytical short-range Coulombic energy ($G_{ana,coul,short}$) (see for example [10]). The self FD Coulombic energy is due to interactions of grid charges within one single atom. The self energy exists even when the atomic charge is exactly positioned on one grid point. It also exists in the absence of solvent and any other charges. It apparently is a pure artifact of the FD approach and must be removed. The short-range FD Coulombic energy is due to interactions between grid charges in two different atoms that are separated by a short distance, usually less than 14 grid units. The short-range Coulombic energy is inaccurate because the atomic charges are mapped onto their eight nearest finite-difference grids, thus causing deviation from the analytical Coulomb energy. The correction of $G_{FD,coul,self}$ and $G_{FD,coul,short}$ is made possible by the work of Luty and McCammon's analytical approach to compute FD Coulombic interactions [165]. Therefore, the PB electrostatic interactions include both Coulombic interactions and reaction field interactions for all atoms of the solute. The total electrostatic energy is included in the energy component EELE in the output file. If you want to know how much of EELE is reaction field, you can run PB twice, once with *epsout* = 80, and once with *epsout* = 1. ESURF term will return the SA nonpolar solvation free energy if it is requested.

Note that the accuracy of PB is related to the finite-difference grid spacing, the convergence criterion for the PB solver, and the cutoff distance used for correction of finite difference Coulombic interactions. PB calculations are memory intensive for macromolecules. Thus, the efficiency of PB depends on how much memory is allocated for it and the performance of the memory system of the computer. The option directly related its memory allocation is the finite-difference grid spacing. To make PB run faster, it is possible to compile the PB code in single precision by removing the line "#define PBDPREC" in file "pb_def.h" under the "src/sander" or "src/pbsa" directory and recompile. Make sure you have successfully installed SANDER and PBSA by running all related test cases before you do this.

12.6. Coupled Potential (QM/MM) Method

B. Wang, R. S. Stanton, A. van der Vaart, D. S. Hartsough, G. Monard, M. Garcia, and K. M. Merz, Jr.

In the following, we have provided a brief overview of the major new theoretical tools that have been included in QM/MM module. The interested reader is encouraged to examine the literature references for further details.

12.6.1. Basic Methodology

The basic strategy for this approach was laid out in a seminal paper by Levitt and Warshel [166] where a classical potential was combined with an early semiempirical quantum mechanical method (MINDO/2). Formally the method can be described as follows (independent of the QM or MM potential function used): The Hamiltonian used within the QM/MM formulation is taken to be an effective Hamiltonian (H_{eff}) which operates on the wavefunction (Ψ) of the system. The wavefunction is dependent on the position of the quantum mechanical nuclei, R_{QM}, the molecular mechanical nuclei, R_M, as well as the positions of the electrons, r.

$$H_{eff}\Psi(r, R_{QM}, R_{M)} = E(r, R_{QM}, R_{M)}\Psi(r, R_{QM}, R_M) \quad (1)$$

The effective Hamiltonian can be divided into three terms (equation 2), which are generated from the interactions which occur within and between the components of the system (see Scheme 1). The contributions considered here include the completely quantum mechanical, H_{QM} (QM in Scheme 1), the purely molecular mechanical interactions, H_{MM} (MM in Scheme 1) and the interactions between the QM and MM portions of the system, $H_{QM/MM}$ (indicated by the arrow in Scheme 1).

$$H_{eff} = H_{QM} + H_{MM} + H_{QM/MM} \quad (2)$$

The total energy of the system can likewise be divided into three component parts.

$$E_{eff} = E_{QM} + E_{MM} + E_{QM/MM} \quad (3)$$

These component energies can be obtained by solving either the Roothaan-Hall equations (associated with Hartree-Fock based methodologies) or the Kohn-Sham equations (associated with density functional theory based methodologies) for H_{eff} (equation 2). Another way to express the total energy of the system is as the expectation value of H_{eff}. The purely MM term can be removed from the integral because it is independent of the electronic positions. Thus, the total energy of the system can be given as,

$$E_{eff} = <\Psi|H_{QM} + H_{QM/MM}|\Psi> + E_{MM} \quad (4)$$

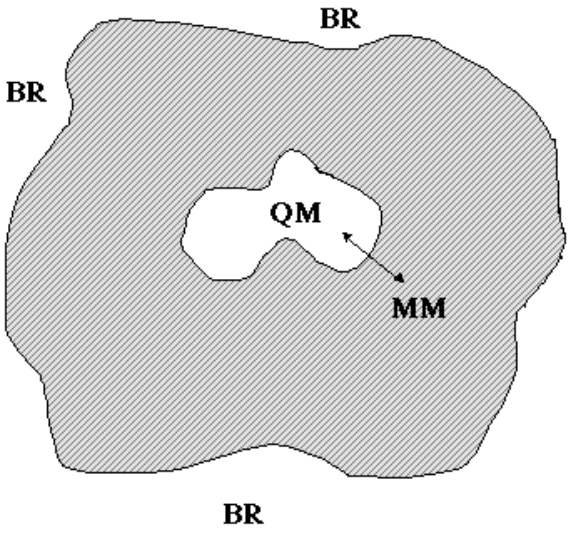
In equation 4, H_{QM} is the Hamiltonian given by either semiempirical, Hartree-Fock or density functional theory, while E_{MM} is an energy obtained using a classical force field of some type. The key remaining term is $H_{QM/MM}$. This term represents the interaction of the MM atom "cores" with the electron cloud of the QM atoms when interacting with MM atoms, as well as the repulsion between the MM and QM atomic cores. Finally, it was found to be necessary to add a Lennard-Jones term to the QM atoms to obtain good interaction energies as well as good geometries for intermolecular interactions [167]. The form of this term is:

$$H_{QM/MM} = -\sum_{iM}\frac{q_M}{|r_{iM}|} + \sum_{A}\frac{q_M Z_A}{|R_{AM}|} + \sum_{AM}\left(\frac{A_{AM}}{R_{AM}^{12}} - \frac{B_{AM}}{R_{AM}^{6}}\right) \quad (5)$$

Where, q_M is the atomic point charge on the MM atom, r_{iM} is the QM electron to MM atom distance, Z_A is the core charge of QM atom A, R_{AM} is the QM atom A to MM atom M distance and A_{AM} and B_{AM} are the Lennard-Jones parameters for QM atom A interacting with MM atom M. The critical term that allows the QM region to "see" the MM environment is the first term in equation 5 where the summation is over all interactions between MM atoms and QM electrons. This represent the core-electron interaction between MM and QM atoms and is incorporated into the QM Hamiltonian explicitly. Thus, the QM electronic structure can respond to its environment through the interaction of its electrons with the surrounding solvent/protein. The last two terms of equation 5 are added on to the total energy once the electronic energy has been determined by a self-consistent field (SCF) procedure and does not affect the electronic distribution of the system directly, but does affect the geometry of the system through the computed gradients and, hence, the resulting electronic energy on subsequent SCF cycles. The original 1976 paper describing this method by Warshel and Levitt [166] was clearly ahead of its time as this approach was not widely used again until the late 1980's and early 1990's when it was re-examined by several research groups using a number of different quantum mechanical methods.

12.6.2. Practical Aspects

There are a number of decisions required when designing and carrying out a QM/MM study. The first is the way in which conformational sampling is to be done. In many ways the choice of sampling technique and choice of model Hamiltonian are interdependent. For example, using a purely *ab initio* approach on a large biomolecular system is too computationally expensive to effectively use a sampling technique like MD, but is more compatible with an energy minimization approach. However, it must be kept in mind that the energy minimization techniques, while less computationally demanding since only a few thousand energy and gradient evaluations



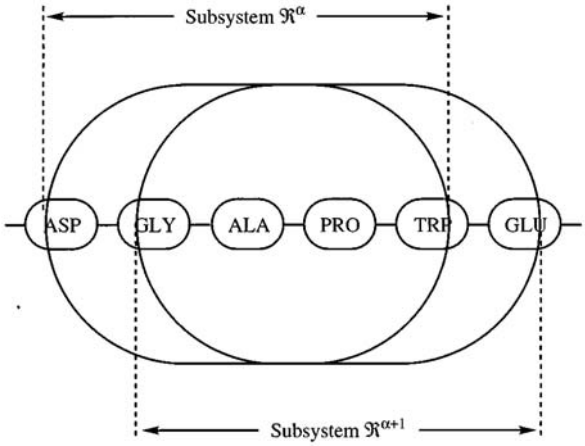
Scheme 1.

are involved, are prone to being trapped in local minima that are near the initial starting configuration. Thus, care must be exercised when using energy minimization techniques when one is interested in sampling the very rich conformational space of a protein. In order to get effective sampling of a protein system a Monte Carlo (MC) or MD approach are preferred to minimization, and because of the generality of MD techniques they are more typically used in QM/MM studies. MD and MC methods are much more computationally demanding than energy minimization because by their very nature they require the evaluation of the energy and forces (MD only) of a system many thousands of times. For example, in a protein system to sample 100 ps using an MD simulation requires at least the evaluation of the energy and forces 100,000 times with a typical time step of 1 fs. Indeed, we have found that in some cases a timestep of 0.5 fs or less may be more appropriate in some QM/MM MD studies of protein systems which further increases the number of times the energy and gradient must be evaluated. In most studies to date, the QM part of the system utilized a semiempirical method because of the expense associated with a fully *ab initio* or DFT method. Indeed, at the present time it is unlikely that a "long" MD simulation using an *ab initio* or DFT Hamiltonian will be carried out on a protein system given the inherent computational expense of this type of calculation. Thus, we are at present limited to MD simulations of proteins using semiempirical models and using energy minimization (or even so-called single point calculations where the energy is evaluated for a single geometric configuration) with *ab initio* or DFT Hamiltonians. Energy minimization, MD and single point calculations are all possible within QM/MM module, but only for a semiempirical Hamiltonian. Future releases will include DFT and *ab initio* capabilities.

The next necessary choice is the selection of the quantum mechanical level to be employed. The choice of MM model will be discussed briefly below, but it should be pointed out the QM portion of a QM/MM calculation so dominates the MM portion, in computational expense and energy, that the critical computational choice which will impact the speed of the algorithm has to do with selecting the QM potential and not the MM model. However it must be noted that recent efforts to incorporate a polarized MM model into a QM/MM calculation increases the expense of the MM portion of the calculation significantly. Semiempirical QM models (e.g., MNDO, AM1 and PM3) formally scale as $O(N^{3)}$ (where due to semiempirical approximations N is typically no greater than 4 for a "heavy" atom like carbon), while local density approximation DFT methods also scale as $O(N^{3)}$, but N is typically much greater than that found in semiempirical theory.

Recently, linear-scaling semiempirical molecular orbital methods have been developed based on the density matrix divide and conquer (D&C) technique which decomposes a large system into a set of small, overlapping subsystems (see Scheme 2) [168-170]. For each subsystem, a comparatively small Fock matrix is diagonalized, and an accurate global description of the system is obtained by assembling information from all the subsystem density matrices. Therefore, the expensive global matrix diagonalization is avoided. When the overall system is large enough, the so-called crossover point is reached and a D&C calculation becomes faster than the standard molecular orbital method.

Typically a simple MM potential is used in QM/MM studies. These potentials contain harmonic bonds and angles along with a truncated Fourier series expansion to represent the torsion potential. Non-bonded interactions (*i.e.*, those interactions beyond the 1 and 4 positions along a polymer chain) are typically represented with a Lennard-Jones "6-12" potential function, while electrostatic interactions are handled using atom centered charges. The major deficiency in these models is the lack of explicit polarization. This leads to an unbalanced model in QM/MM studies typically because the QM region is polarized while the MM region is not. However, some work has gone into incorporating polarization effects into the MM region of a QM/MM study, but this type of effect is not yet incorporated into Amber.



Scheme 2.

12.6.3. vdW Parameters for QM Atoms

One of the more difficult decisions which needs to be made when setting up a QM/MM simulation is the proper value for the van der Waals parameters to be used on the QM atoms for their interaction with MM atoms (term 3 of equation 5). While one of the strengths of QM/MM methods is their ability to accurately model a system without extensive reparameterization of either the QM or MM methodology, it has been noted empirically that a 5-10% scaling of the vdW parameters normally used within a classical force field can greatly improve the accuracy of the free energies of solvation calculated using QM/MM methods [167,171-174]. Some researchers will also scale the MM core charges seen by the QM partition of the system (terms 1 and 2 of equation 5) to improve calculated radial distribution functions and solvation free energies calculated for a solute. While researchersworked to establish definitively the vdW parameters to be used in QM/MM calculations through in depth parameterization on test systems it must be noted that the necessary scaling factor is highly method dependent (*i.e.* S, DF, or HF) and even basis set dependent. This is unfortunate as it requires individual parameterization for any particular combination of MM and QM levels of theory or acceptance of the 5-10% scaling factor approximation.

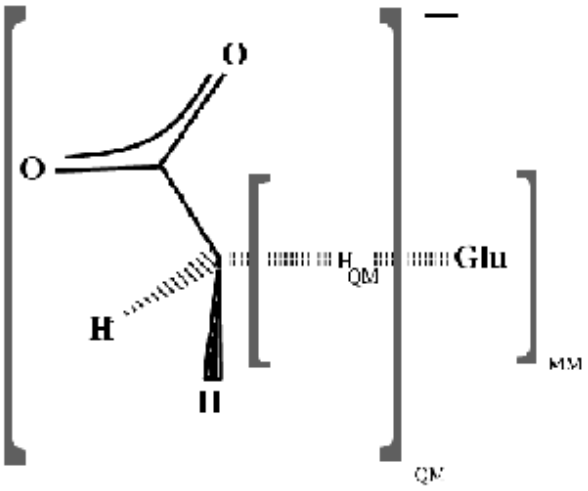
12.6.4. Link Atoms in QM/MM Studies of Enzymes

In solution phase reactions the boundary between the solute (usually QM) and the solvent (usually MM) is very clear and typically it is possible to avoid introducing the QM/MM interface between atoms that are covalently linked. However, in the case of enzymes this is not the case and we have to introduce the concept of link atoms that covalently connect the QM and MM regions used in the representation of the protein. This interfacial region can be quite arbitrary and should be chosen with care. For example, consider modeling a glutamic acid residue within the QM/MM framework (see Scheme 3). The first consideration is where to make the QM and MM "cut" such that it does not adversely affect the electronic structure that we associate with a glutamate ion. If we treat the carboxylate as the formate anion this will alter the pKa relative to a typical carboxylate anion (4.0 versus 4.5). Thus, while formate is computationally convenient it is electronically the incorrect choice. A better choice is to add an extra carbon atom to generate the acetate anion, which has a better pKa match with the glutamate anion.

The next step necessary when the QM/MM interface falls at a covalent bond is the introduction of an extra QM atom, for example H_{QM} in Scheme 3. This is necessary to cap the exposed

valence at the carbon atom such that a closed shell calculation is done. There are several reasons for doing this not the least of which being that the free radical of the carboxylate anion is not what we are attempting to model. Another import consideration is the fact that closed-shell calculations are from a technical aspect far easier to carry out than are open-shell calculations. This so-called link atom does not see any atoms within the MM region because it is an extra QM atom solely included to satisfy valence considerations that normally would not be present in the real system. The final MM carbon atom of the Glu residue is then attached to the final QM carbon atom by a MM-type harmonic bond. This bond keeps the MM and QM regions at the appropriate distance apart. The remaining angles and dihedrals present across the QM/MM boundary are typically represented using MM terms (*i.e.*, an harmonic bond angle and a Fourier torsion potential) and care should be taken in selecting these such that the torsional properties around the bond at the QM/MM boundary are as accurate as possible (e.g., based on the appropriate QM or MM calculations).

The concept of a link atom is an approximation inherent in the basic QM/MM approach that ones needs to be aware of when treating protein systems or any system that requires the use of a covalent QM/MM boundary region. However, through careful selection of the location of this region gross errors can be avoided. New and better ways need to be formulated to treat the link atom region, but such ideas are not yet incorporated into sander.QMMM.



Scheme 3.

13. Appendices

13.1. Appendix A: Namelist Input Syntax

Namelist provides list-directed input, and convenient specification of default values. It dates back to the early 1960's on the IBM 709, but was regrettably not part of Fortran 77. It is a part of the Fortran 90 standard, and is supported as well by most Fortran 77 compilers (including g77).

Namelist input groups take the form:

```
&name
 var1=value, var2=value, var3(sub)=value,
 var4(sub,sub,sub)=value,value,
 var5=repeat*value,value,
/
```

The variables must be names in the Namelist variable list. The order of the variables in the input list is of no significance, except that if a variable is specified more than once, later assignments may overwrite earlier ones. Blanks may occur anywhere in the input, except embedded in constants (other than string constants, where they count as ordinary characters).

It is common in older inputs for the ending "/" to be replaced by "&end"; this is non-standard-conforming.

Letter case is ignored in all character comparisons, but case is preserved in string constants. String constants must be enclosed by single quotes ('). If the text string itself contains single quotes, indicate them by two consecutive single quotes, e.g. C1' becomes 'C1'' as a character string constant.

Array variables may be subscripted or unsubscripted. An unsubscripted array variable is the same as if the subscript (1) had been specified. If a subscript list is given, it must have either one constant, or exactly as many as the number in the declared dimension of the array. Bounds checking is performed for ALL subscript positions, although if only one is given for a multi-dimension array, the check is against the entire array size, not against the first dimension. If more than one constant appears after an array assignment, the values go into successive locations of the array. It is NOT necessary to input all elements of an array.

Any constant may optionally be preceded by a positive (1,2,3,..) integer repeat factor, so that, for example, 25*3.1415 is equivalent to twenty-five successive values 3.1415. The repeat count separator, *, may be preceded and followed by 0 or more blanks. Valid LOGICAL constants are 0, F, .F., .FALSE., 1, T, .T., and .TRUE.; lower case versions of these also work.

13.2. Appendix B: GROUP Specification

Entering Group Information

This section describes the format used to define groups of atoms in various AMBER programs. In *sander*, a group can be specified as a movable "belly" while the other atoms are fixed absolutely in space (aside from scaling caused by constant pressure simulation), and/or a group of movable atoms can independently restrained (held by a potential) at their positions. In *anal*, groups can be defined for energy analysis.

Except in the analysis module where different groups of atoms are considered with different group numbers for energy decomposition, in all other places the groups of atoms defined are considered as marked atoms to be included for certain types of calculations. In the case of constrained minimization or dynamics, the atoms to be constrained are read as groups with a different weight for each group.

Reading of groups is performed by the routine RGROUP and you are advised to consult it if there is still some ambiguity in the documentation.

Input description:

```
   - 1 -       Title

               format(20a4)

   ITITL       Group title for identification.

        Setting ITITL = 'END' ends group input.

-------------------------------------------------------------------------

   - 1A -      Weight

        This line is only provided/read when using GROUP input to
        define restrained atoms.

             format(f)

   WT   The harmonic force constants in kcal/mol-A**2 for the group
        of atoms for restraining to a reference position.

-------------------------------------------------------------------------

   - 1B -     Control to define the group

        KTYPG , (IGRP(I) , JGRP(I) , I = 1,7)

             format(a,14i)

   KTYPG      Type of atom selection performed.  A molecule can be
```

defined by using only 'ATOM' or 'RES', or part of the molecule can be defined by 'ATOM' and part by 'RES'.

'ATOM' The group is defined in terms of atom numbers. The atom number list is given in igrp and jgrp.

'RES' The group is defined in terms of residue numbers. The residue number list is given in igrp and jgrp.

'FIND' This control is used to make additional conditions (apart from the 'ATOM' and 'RES' controls) which a given atom must satisfy to be included in the current group. The conditions are read in the next section (1C) and are terminated by a SEARCH card.

Note that the conditions defined by FIND filter any set(s) of atoms defined by the following ATOM/RES instructions. For example,

```
-- group input: select main chain atoms --
FIND
* * M *
SEARCH
RES 1 999
END
END
```

'END' End input for the current group. Followed by either another group definition (starting again with line 1 above), or by a second 'END' "card", which terminates all group input.

IGRP(I) , JGRP(I)

The atom or residue pointers. If ktypg .eq. 'ATOM' all atoms numbered from igrp(i) to jgrp(i) will be put into the current group. If ktypg .eq. 'RES' all atoms in the residues numbered from igrp(i) to jgrp(i) will be put into the current group. If igrp(i) = 0 the next control card is read.

It is not necessary to fill groups according to the numerical order of the residues. In other words, Group 1 could contain residues 40-95 of a protein, Group 2 could contain residues 1-40 and Group 3 could contain residues 96-105.

If ktypg .eq. 'RES', then associating a minus sign with igrp(i) will cause all residues igrp(i) through jgrp(i) to be placed in separate groups.

In the analysis modules, all atoms not explicitly defined as members of a group will be combined as a unit in the (n + 1) group, where the (n) group in the last defined group.

- 1C - Section to read atom characteristics

***** Read only if KTYPG = 'FIND' *****

JGRAPH(I) , JSYMBL(I) , JTREE(I) , JRESNM(I)

format(4a)

A series of filter specifications are read. Each filter consists of four fields (JGRAPH,JSYMBL,JTREE,JRESNM), and each filter is placed on a separate line. Filter specification is terminated by a line with JGRAPH = 'SEARCH'. A maximum of 10 filters may be specified for a single 'FIND' command.

The union of the filter specifications is applied to the atoms defined by the following ATOM/RES cards. I.e. if an atom satisfies any of the filters, it will be included in the current group. Otherwise, it is not included. For example, to select all non main chain atoms from residues 1 through 999:

```
-- group input: select non main chain atoms --
FIND
* * S *
* * B *
* * 3 *
* * E *
SEARCH
RES 1 999
END
END
```

'END' End input for the current group. Followed by either another
The four fields for each filter line are:

JGRAPH(I) The atom name of atom to be included. If this and the following three characteristics are satisfied the atom is included in the group. The wild card '*' may be used to to indicate that any atom name will satisfy the search.

JSYMBL(I) Amber atom type of atom to be included. The wild card '*' may be used to indicate that any atom type will

```
          satisfy the search.

     JTREE(I)   The tree name (M, S, B, 3, E) of the atom to be included.
          The wild card '*' may be used to indicate that any tree
          name will satisfy the search.

     JRESNM(I)  The residue name to which the atom has to belong to be
          included in the group.  The wild card '*' may be used to
          indicate that any residue name will satisfy the search.
```

Examples:

The molecule 18-crown-6 will be used to illustrate the group options. This molecule is composed of six repeating (-CH2-O-CH2-) units. Let us suppose that one created three residues in the PREP unit: CRA, CRB, CRC. Each of these is a (-CH2-O-CH2-) moiety and they differ by their dihedral angles. In order to construct 18-crown-6, the residues CRA, CRB, CRC, CRB, CRC, CRB are linked together during the LINK module with the ring closure being between CRA(residue 1) and CRB(residue 6).

Input 1:

```
Title one
RES   1  5
END
Title two
RES   6
END
END
```

Output 1: Group 1 will contain residues 1 through 5 (CRA, CRB, CRC, CRB, CRC) and Group 2 will contain residue 6 (CRB).

Input 2:

```
Title one
RES   1  5
END
Title two
ATOM 36 42
END
END
```

Output 2: Group 1 will contain residues 1 through 5 (CRA, CRB, CRC, CRB, CRC) and Group 2 will contain atoms 36 through 42. Coincidentally, atoms 36 through 42 are also all the atoms in residue 6.

Input 3:

```
Title one
RES  -1  6
END
END
```

Output 3: Six groups will be created; Group 1: CRA, Group 2: CRB,..., Group 6: CRB.

Input 4:

```
Title one
FIND
O2 OS M CRA
SEARCH
RES   1  6
END
END
```

Output 4: Group 1 will contain those atoms with the atom name ’O2’, atom type ’OS’, tree name ’M’ and residue name ’CRA’.

Input 5:

```
Title one
FIND
O2 OS * *
SEARCH
RES   1  6
END
END
```

Output 5: Group 1 will contain those atoms with the atom name ’O2’, atom type ’OS’, any tree name and any residue name.

13.3. Appendix C: Parameter File Format

Listed below is the "old" prmtop format. It can still be used in all Amber routines, and can even still be generated in LEaP by the command:

```
set default oldPrmtopFormat on
```

The "new" format (introduced in Amber 7) is pretty much compatible with what is listed below. However, each section now has two extra lines at the beginning: the first line begins with "%FLAG ", followed by an identifier; the second line begins with "%FORMAT", followed by a Fortran format string that is used to parse the following lines. The format given must provide spaces between all entries, so that the file can be more easily read with a "C" program using `fscanf()`. In addition, the very first line of the new format file begins with "%VERSION ", followed by version and date information.

Although this sounds complicated, looking at an output from LEaP is probably the easiest way to understand what is going on.

```
FORMAT(20a4)  (ITITL(i), i=1,20)

  ITITL  : title

FORMAT(12i6)  NATOM,  NTYPES, NBONH,  MBONA,  NTHETH, MTHETA,
              NPHIH,  MPHIA,  NHPARM, NPARM,  NNB,    NRES,
              NBONA,  NTHETA, NPHIA,  NUMBND, NUMANG, NPTRA,
              NATYP,  NPHB,   IFPERT, NBPER,  NGPER,  NDPER,
              MBPER,  MGPER,  MDPER,  IFBOX,  NMXRS,  IFCAP,
              NEXTRA

  NATOM  : total number of atoms
  NTYPES : total number of distinct atom types
  NBONH  : number of bonds containing hydrogen
  MBONA  : number of bonds not containing hydrogen
  NTHETH : number of angles containing hydrogen
  MTHETA : number of angles not containing hydrogen
  NPHIH  : number of dihedrals containing hydrogen
  MPHIA  : number of dihedrals not containing hydrogen
  NHPARM : currently not used
  NPARM  : set to 1 if LES is used
  NNB    : number of excluded atoms (=NEXT)
  NRES   : number of residues
  NBONA  : MBONA + number of constraint bonds
  NTHETA : MTHETA + number of constraint angles
  NPHIA  : MPHIA + number of constraint dihedrals
  NUMBND : number of unique bond types
  NUMANG : number of unique angle types
  NPTRA  : number of unique dihedral types
  NATYP  : number of atom types in parameter file, see SOLTY below
  NPHB   : number of distinct 10-12 hydrogen bond pair types
  IFPERT : set to 1 if perturbation info is to be read in
  NBPER  : number of bonds to be perturbed
  NGPER  : number of angles to be perturbed
  NDPER  : number of dihedrals to be perturbed
  MBPER  : number of bonds with atoms completely in perturbed group
  MGPER  : number of angles with atoms completely in perturbed group
  MDPER  : number of dihedrals with atoms completely in perturbed groups
  IFBOX  : set to 1 if standard periodic box, 2 when truncated octahedral
  NMXRS  : number of atoms in the largest residue
  IFCAP  : set to 1 if the CAP option from edit was specified
  NEXTRA : number of "extra points" (atom type of EP)

FORMAT(20a4)  (IGRAPH(i), i=1,NATOM)
```

```
  IGRAPH : the user atoms names

FORMAT(5E16.8)  (CHRG(i), i=1,NATOM)

  CHRG   : the atom charges.  (Divide by 18.2223 to convert to units of
           electron charge)

FORMAT(5E16.8)  (AMASS(i), i=1,NATOM)

  AMASS  : the atom masses

FORMAT(12I6)  (IAC(i), i=1,NATOM)

  IAC    : index for the atom types involved in Lennard Jones (6-12)
           interactions.  See ICO below.

FORMAT(12I6)  (NUMEX(i), i=1,NATOM)

  NUMEX  : total number of excluded atoms for atom "i".  See
           NATEX below.

FORMAT(12I6)  (ICO(i), i=1,NTYPES*NTYPES)

  ICO    : provides the index to the nonbon parameter
           arrays CN1, CN2 and ASOL, BSOL.  All possible 6-12
           or 10-12 atoms type interactions are represented.
           NOTE: A particular atom type can have either a 10-12
           or a 6-12 interaction, but not both.  The index is
           calculated as follows:

             index = ICO(NTYPES*(IAC(i)-1) + IAC(j))

           If index is positive, this is an index into the
           6-12 parameter arrays (CN1 and CN2) otherwise it
           is an index into the 10-12 parameter arrays (ASOL
           and BSOL).

FORMAT(20A4)  (LABRES(i), i=1,NRES)

  LABRES : the residue labels

FORMAT(12I6)  (IPRES(i), i=1,NRES)

  IPRES  : the atom number of the first atom in residue "i"

FORMAT(5E16.8)  (RK(i), i=1,NUMBND)

  RK     : force constant for the bonds of each type, kcal/mol
```

```
FORMAT(5E16.8)  (REQ(i), i=1,NUMBND)

  REQ    : equilibrium bond length for the bonds of each type, angstroms

FORMAT(5E16.8)  (TK(i), i=1,NUMANG)

  TK     : force constant for the angles of each type, kcal/mol A**2

FORMAT(5E16.8)  (TEQ(i), i=1,NUMANG)

  TEQ    : the equilibrium angle for the angles of each type, degrees

FORMAT(5E16.8)  (PK(i), i=1,NPTRA)

  PK     : force constant for the dihedrals of each type, kcal/mol

FORMAT(5E16.8)  (PN(i), i=1,NPTRA)

  PN     : periodicity of the dihedral of a given type

FORMAT(5E16.8)  (PHASE(i), i=1,NPTRA)

  PHASE  : phase of the dihedral of a given type

FORMAT(5E16.8)  (SOLTY(i), i=1,NATYP)

  SOLTY  : currently unused (reserved for future use)

FORMAT(5E16.8)  (CN1(i), i=1,NTYPES*(NTYPES+1)/2)

  CN1    : Lennard Jones r**12 terms for all possible atom type
           interactions, indexed by ICO and IAC; for atom i and j
           where i < j, the index into this array is as follows
           (assuming the value of ICO(INDEX) is positive):
           CN1(ICO(NTYPES*(IAC(i)-1)+IAC(j))).

FORMAT(5E16.8)  (CN2(i), i=1,NTYPES*(NTYPES+1)/2)

  CN2    : Lennard Jones r**6 terms for all possible atom type
           interactions.  Indexed like CN1 above.

NOTE: the atom numbers in the arrays which follow that describe bonds,
angles, and dihedrals are obfuscated by the following formula (for
runtime speed in indexing arrays).  The true atom number equals the
absolute value of the number divided by three, plus one.  In the case
of the dihedrals, if the fourth atom is negative, this implies an
improper torsion and if the third atom is negative, this implies that
end group interactions are to be ignored.  End group interactions are
ignored, for example, in dihedrals of various ring systems (to prevent
```

```
double counting) and in multiterm dihedrals.

FORMAT(12I6)  (IBH(i),JBH(i),ICBH(i), i=1,NBONH)

  IBH    : atom involved in bond "i", bond contains hydrogen
  JBH    : atom involved in bond "i", bond contains hydrogen
  ICBH   : index into parameter arrays RK and REQ

FORMAT(12I6)  (IB(i),JB(i),ICB(i), i=1,NBONA)

  IB     : atom involved in bond "i", bond does not contain hydrogen
  JB     : atom involved in bond "i", bond does not contain hydrogen
  ICB    : index into parameter arrays RK and REQ

FORMAT(12I6)  (ITH(i),JTH(i),KTH(i),ICTH(i), i=1,NTHETH)

  ITH    : atom involved in angle "i", angle contains hydrogen
  JTH    : atom involved in angle "i", angle contains hydrogen
  KTH    : atom involved in angle "i", angle contains hydrogen
  ICTH   : index into parameter arrays TK and TEQ for angle
           ITH(i)-JTH(i)-KTH(i)

FORMAT(12I6)  (IT(i),JT(i),KT(i),ICT(i), i=1,NTHETA)

  IT     : atom involved in angle "i", angle does not contain hydrogen
  JT     : atom involved in angle "i", angle does not contain hydrogen
  KT     : atom involved in angle "i", angle does not contain hydrogen
  ICT    : index into parameter arrays TK and TEQ for angle
           IT(i)-JT(i)-KT(i)

FORMAT(12I6)  (IPH(i),JPH(i),KPH(i),LPH(i),ICPH(i), i=1,NPHIH)

  IPH    : atom involved in dihedral "i", dihedral contains hydrogen
  JPH    : atom involved in dihedral "i", dihedral contains hydrogen
  KPH    : atom involved in dihedral "i", dihedral contains hydrogen
  LPH    : atom involved in dihedral "i", dihedral contains hydrogen
  ICPH   : index into parameter arrays PK, PN, and PHASE for
           dihedral IPH(i)-JPH(i)-KPH(i)-LPH(i)

FORMAT(12I6)  (IP(i),JP(i),KP(i),LP(i),ICP(i), i=1,NPHIA)

  IP     : atom involved in dihedral "i", dihedral does not contain hydrogen
  JP     : atom involved in dihedral "i", dihedral does not contain hydrogen
  KP     : atom involved in dihedral "i", dihedral does not contain hydrogen
  LP     : atom involved in dihedral "i", dihedral does not contain hydrogen
  ICP    : index into parameter arrays PK, PN, and PHASE for
           dihedral IPH(i)-JPH(i)-KPH(i)-LPH(i).

FORMAT(12I6)  (NATEX(i), i=1,NEXT)
```

```
  NATEX  : the excluded atom list.  To get the excluded list for atom
           "i" you need to traverse the NUMEX list, adding up all
           the previous NUMEX values, since NUMEX(i) holds the number
           of excluded atoms for atom "i", not the index into the
           NATEX list.  Let IEXCL = SUM(NUMEX(j), j=1,i-1), then
           excluded atoms are NATEX(IEXCL) to NATEX(IEXCL+NUMEX(i)).
           The excluded atoms for each atom "i" must be in ascending
           numerical order.

FORMAT(5E16.8)  (ASOL(i), i=1,NPHB)

  ASOL   : the value for the r**12 term for hydrogen bonds of all
           possible types.  Index into these arrays is equivalent
           to the CN1 and CN2 arrays, however the index is negative.
           For example, for atoms i and j, with i < j, the index is
           -(NTYPES*(IAC(i)-1)+IAC(j)).

FORMAT(5E16.8)  (BSOL(i), i=1,NPHB)

  BSOL   : the value for the r**10 term for hydrogen bonds of all
           possible types.  Indexed like ASOL.

FORMAT(5E16.8)  (HBCUT(i), i=1,NPHB)

  HBCUT  : no longer in use

FORMAT(20A4)  (ISYMBL(i), i=1,NATOM)

  ISYMBL : the AMBER atom types for each atom

FORMAT(20A4)  (ITREE(i), i=1,NATOM)

  ITREE  : the list of tree joining information, classified into five
           types.  M -- main chain, S -- side chain, B -- branch point,
           3 -- branch into three chains, E -- end of the chain

FORMAT(12I6)  (JOIN(i), i=1,NATOM)

  JOIN   : tree joining information, potentially used in ancient
           analysis programs.  Currently unused in sander or gibbs.

FORMAT(12I6)  (IROTAT(i), i = 1, NATOM)

  IROTAT : apparently the last atom that would move if atom i was
           rotated, however the meaning has been lost over time.
           Currently unused in sander or gibbs.

================================================================================
```

```
**** The following are only present if IFBOX .gt. 0 ****

FORMAT(12I6)  IPTRES, NSPN, NSPSOL

  IPTRES : final residue that is considered part of the solute,
           reset in sander and gibbs
  NSPM   : total number of molecules
  NSPSOL : the first solvent "molecule"

FORMAT(12I6)  (NSP(i), i=1,NSPM)

  NSP    : the total number of atoms in each molecule,
           necessary to correctly determine the pressure scaling

FORMAT(5E16.8)  BETA, BOX(1), BOX(2), BOX(3)

  BETA   : periodic box, angle between the XY and YZ planes in
           degrees.
  BOX    : the periodic box lengths in the X, Y, and Z directions

================================================================================

**** The following are only present if IFCAP .gt. 0 ****

FORMAT(12I6)  NATCAP

  NATCAP : last atom before the start of the cap of waters
           placed by edit

FORMAT(5E16.8)  CUTCAP, XCAP, YCAP, ZCAP

  CUTCAP : the distance from the center of the cap to the outside
  XCAP   : X coordinate for the center of the cap
  YCAP   : Y coordinate for the center of the cap
  ZCAP   : Z coordinate for the center of the cap

================================================================================

**** The following are only present if IFPERT .gt. 0 ****

Note that the initial state, or equivalently the prep/link/edit state,
is represented by lambda=1 and the perturbed state, or final
state specified in parm, is the lambda=0 state.

FORMAT(12I6)  (IBPER(i), JBPER(i), i=1,NBPER)

  IBPER  : atoms involved in perturbed bonds
  JBPER  : atoms involved in perturbed bonds
```

```
FORMAT(12I6)  (ICBPER(i), i=1,2*NBPER)

  ICBPER : pointer into the bond parameter arrays RK and REQ for the
           perturbed bonds.  ICBPER(i) represents lambda=1 and
           ICBPER(i+NBPER) represents lambda=0.

FORMAT(12I6)  (ITPER(i), JTPER(i), KTPER(i), i=1,NGPER)

  IPTER  : atoms involved in perturbed angles
  JTPER  : atoms involved in perturbed angles
  KTPER  : atoms involved in perturbed angles

FORMAT(12I6)  (ICTPER(i), i=1,2*NGPER)

  ICTPER : pointer into the angle parameter arrays TK and TEQ for
           the perturbed angles.  ICTPER(i) represents lambda=0 and
           ICTPER(i+NGPER) represents lambda=1.

FORMAT(12I6)  (IPPER(i), JPPER(i), KPPER(i), LPPER(i), i=1,NDPER)

  IPTER  : atoms involved in perturbed dihedrals
  JPPER  : atoms involved in perturbed dihedrals
  KPPER  : atoms involved in perturbed dihedrals
  LPPER  : atoms involved in perturbed dihedrals

FORMAT(12I6)  (ICPPER(i), i=1,2*NDPER)

  ICPPER : pointer into the dihedral parameter arrays PK, PN and
           PHASE for the perturbed dihedrals.  ICPPER(i) represents
           lambda=1 and ICPPER(i+NGPER) represents lambda=0.

FORMAT(20A4)  (LABRES(i), i=1,NRES)

  LABRES : residue names at lambda=0

FORMAT(20A4)  (IGRPER(i), i=1,NATOM)

  IGRPER : atomic names at lambda=0

FORMAT(20A4)  (ISMPER(i), i=1,NATOM)

  ISMPER : atomic symbols at lambda=0

FORMAT(5E16.8)  (ALMPER(i), i=1,NATOM)

  ALMPER : unused currently in gibbs

FORMAT(12I6)  (IAPER(i), i=1,NATOM)
```

```
  IAPER  : IAPER(i) = 1 if the atom is being perturbed

FORMAT(12I6)  (IACPER(i), i=1,NATOM)

  IACPER : index for the atom types involved in Lennard Jones
           interactions at lambda=0.  Similar to IAC above.
           See ICO above.

FORMAT(5E16.8)  (CGPER(i), i=1,NATOM)

  CGPER  : atomic charges at lambda=0

================================================================================

**** The following is only present if IPOL .eq. 1 ***

FORMAT(5E18.8) (ATPOL(i), i=1,NATOM)

  ATPOL  : atomic polarizabilities

**** The following is only present if IPOL .eq. 1 .and. IFPERT .eq. 1 ****

FORMAT(5E18.8) (ATPOL1(i), i=1,NATOM)

  ATPOL1 : atomic polarizabilities at lambda = 1 (above is at lambda = 0)

================================================================================

**** The following is only present if NPARM .eq. 1 ***

FORMAT(I8) LES_NTYP
  LES_NTYP  : number of LES types

FORMAT(10I8) (LES_TYPE(i), i=1,NATOM)

  LES_TYPE  : LES atom type; each new set of copies is a new type

FORMAT(5E16.8) (LES_FAC(i), i=1,LES_NTYP*LES_NTYP)

  LES_FAC   : LES scaling factors

FORMAT(10I8) (LES_CNUM(i), i=1,NATOM)

  LES_CNUM  : LES copy number - which LES copy an atom belongs to

FORMAT(10I8) (LES_ID(i), i=1,NATOM)

  LES_ID    : LES subspace number to which each atom belongs
```

13.4. Appendix D: Restart File Format

```
FORMAT(20A4) ITITL

  ITITL  : the title of the current run, from the AMBER
           parameter/topology file

FORMAT(I5,E15.7) NATOM,TIME

  NATOM  : total number of atoms in coordinate file
  TIME   : option, current time in the simulation (picoseconds)

           Note: Amber programs check the NATOM representation: if it
           contains 6 digits rather than 5, it is read in an I6 format,
           instead of I5.  This allows for simulations with more than
           100,000 atoms, while preserving backwards compatibility with
           older files.

FORMAT(6F12.7) (X(i), Y(i), Z(i), i = 1,NATOM)

  X,Y,Z  : coordinates

IF dynamics:

FORMAT(6F12.7) (VX(i), VY(i), VZ(i), i = 1,NATOM)

  VX,VY,VZ : velocities

IF periodic box [4.0 and previous: only if constant pressure]:

FORMAT(6F12.7) BOX(1), BOX(2), BOX(3)

  BOX    : size of the periodic box
```

13.5. Appendix E: Trajectory (Coordinates or Velocities) File Format

```
FORMAT(20A4) ITITL

  ITITL  : the title of the current run, from the AMBER
           parameter/topology file

The following is sequentially dropped for each snapshot of the
trajectory:
```

```
FORMAT(10F8.3)  (X(i), Y(i), Z(i), i=1,NATOM)

  X,Y,Z  : coordinates or velocities

IF periodic box [4.0 and previous: only if constant pressure]:

FORMAT(10F8.3)  BOX(1), BOX(2), BOX(3)

  BOX    : size of periodic box
```

13.6. Appendix F: Retired Namelist Variables

Unfortunately, many implementations of Fortran namelist reading do not emit specific errors for invalid variables. To remedy this situation retired input variables are kept in their namelist, and a warning is produced in the output file. Listed below are input variables that have been eliminated in the development period from version 7 to version 8. For each variable the relevant program and namelist are specified. A terse description of the original meaning is followed by the new behavior.

DOBS — Sander `&align` namelist. The observed dipolar splitting, in Hz. Now lower and upper bounds are specified with DOBSL and DOBSU.

DTEMP — Sander `&cntrl` namelist. A control of temperature for NTT = 4 via the reassignment of velocities. Now NTT = 4 is included only for historical reasons and with reduced functionality.

DXM — Sander `&cntrl` namelist. The maximum step length in an energy minimization. Now no limit on the step length exists. In fact, this behavior is identical to that in Amber 7 since DXM was unused.

FRC_INT — Sander `&ewald` namelist. A control of the computation of forces in PME. Now forces cannot be obtained by interpolation; the forces are always calculated via differentiation of the energy.

HEAT — Sander `&cntrl` namelist. The initial velocities scaling factor for temperature regulation. Now the initial velocities are controlled solely by TEMPI.

ISCHRGD — Sander `&ewald` namelist. The control of charge neutralization in a unit cell. Now an insignificant nonzero net charge, one less than or equal to 0.01, is assumed to be due to roundoff error and is always neutralized. In fact, this behavior is identical to that in Amber 7 since ISCHRGD was silently ignored.

MATCAP Sander `&cntrl` namelist. The modified cap atom pointer. Now the cap atom pointer in the *prmtop* cannot be modified.

NTU Sander `&cntrl` namelist. The behavior is identical to that in Amber 7 since NTU was reset to 1 even if it appeared in the namelist.

PLEVEL Sander `&cntrl` namelist. The parallelization level for constant pressure simulations. Now all the atoms of a small molecule are assigned to a single processor, and, consequently, the velocities do not need to be distributed to compute the correct virial.

14. References

1. D.A. Pearlman, D.A. Case, J.W. Caldwell, W.S. Ross, T.E. Cheatham, III, S. DeBolt, D. Ferguson, G. Seibel & P. Kollman. AMBER, a package of computer programs for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to simulate the structural and energetic properties of molecules. *Comp. Phys. Commun.* **91,** 1-41 (1995).
2. J.W. Ponder & D.A. Case. Force fields for protein simulations. *Adv. Prot. Chem.* **66,** 27-85 (2003).
3. Y. Duan, C. Wu, S. Chowdhury, M.C. Lee, G. Xiong, W. Zhang, R. Yang, P. Cieplak, R. Luo & T. Lee. A Point-Charge Force Field for Molecular Mechanics Simulations of Proteins. *J. Comput. Chem.* **24,** 1999-2012 (2003).
4. J. Wang, R.M. Wolf, J.W. Caldwell, P.A. Kollamn & D.A. Case. Development and testing of a general Amber force field. *J. Comput. Chem.* (2004). (in press).
5. K.N. Kirschner & R.J. Woods. Solvent interactions determine carbohydrate conformation. *Proc. Natl. Acad. Sci. USA* **98,** 10541-10545 (2001).
6. M. Basma, S. Sundara, D. Calgan, T. Venali & R.J. Woods. Solvated ensemble averaging in the calculation of partial atomic charges. *J. Comput. Chem.* **22,** 1125-1137 (2001).
7. K.N. Kirschner, R.J. Woods & Quantum mechanical study of the nonbonded forces in water-methanol complexes. *J. Phys. Chem. A* **105,** 4150-4155 (2001).
8. A. Onufriev, D. Bashford & D.A. Case. Exporling protein native states and large-scale conformational changes with a modified generalized Born model. *Proteins* (2004). in press
9. R. Luo, L. David & M.K. Gilson. Accelerated Poisson-Boltzmann Calculations for Static and Dynamic Systems. *J. Comput. Chem.* **23,** 1244-1253 (2002).
10. Q. Lu & R. Luo. A Poisson-Boltzmann dynamics method with nonperiodic boundary condition. *J. Chem. Phys.* **119,** 11035-11047 (2003).
11. S. Harvey & J.A. McCammon. *Dynamics of Proteins and Nucleic Acids.* Cambridge: Cambridge University Press, (1987).
12. A.R. Leach. *Molecular Modelling. Principles and Applications, Second Edition.* Harlow, England: Prentice-Hall, (2001).
13. M.P. Allen & D.J. Tildesley. *Computer Simulation of Liquids.* Oxford: Clarendon Press, (1987).
14. D. Frenkel & B. Smit. *Understanding Molecular Simulation: From Algorithms to Applications.* San Diego: Academic Press, (1996).
15. W.F. van Gunsteren, P.K. Weiner & A.J. Wilkinson, eds.. *Computer Simulations of Biomolecular Systems, Vol. 3.*. Leiden: ESCOM Science Publishers, (1997).
16. L.R. Pratt & G. Hummer, eds.. *Simulation and Theory of Electrostatic Interactions in Solution.* Melville, NY: American Institute of Physics, (1999).
17. O. Becker, A.D. MacKerell, B. Roux & M. Watanabe, eds.. *Computational Biochemistry and Biophysics.* New York: Marcel Dekker, (2001).
18. J.J. Vincent & K.M. Merz, Jr.. A highly portable parallel implementation of AMBER4 using the message passing interface standard. *J. Comput. Chem.* **16,** 1420-1427 (1995).

19. T. E. Cheatham, III, B. R. Brooks & P. A. Kollman. Molecular modeling of nucleic acid structure. In *Current Protocols in Nucleic Acid Chemistry*, New York: Wiley, (1999). pp. Sections 7.5, 7.8, 7.9, 7.10.

20. J. Wang, P. Cieplak & P.A. Kollman. How well does a restrained electrostatic potential (RESP) model perform in calculating conformational energies of organic and biological molecules?. *J. Comput. Chem.* **21,** 1049-1074 (2000).

21. C. Simmerling, B. Strockbine & A.E. Roitberg. All-Atom Structure Prediction and Folding Simulations of a Stable Protein. *J. Am. Chem. Soc.* **124,** 11258-11259 (2002).

22. A.E. García & K.Y. Sanbonmatsu. α-helical stabilization by side chain shielding of backbone hydrogen bonds. *Proc. Natl. Acad. Sci. USA* **99,** 2782-2787 (2002).

23. P. Cieplak, J. Caldwell & P. Kollman. Molecular Mechanical Models for Organic and Biological Systems Going Beyond the Atom Centered Two Body Additive Approximation: Aqueous Solution Free Energies of Methanol and N-Methyl Acetamide, Nucleic Acid Base, and Amide Hydrogen Bonding and Chloroform/Water Partition Coefficients of the Nucleic Acid Bases. *J. Comput. Chem.* **22,** 1048-1057 (2001).

24. R.W. Dixon & P.A. Kollman. Advancing Beyond the Atom-Centered Model in Additive and Nonadditive Molecular Mechanics.. *J. Comput. Chem.* **18,** 1632-1646 (1997).

25. E. Meng, P. Cieplak, J.W. Caldwell & P.A. Kollman. Accurate solvation free energies of acetate and methylammonium ions calculated with a polarizable water model. *J. Am. Chem. Soc.* **116,** 12061-12062 (1994).

26. W.D. Cornell, P. Cieplak, C.I. Bayly, I.R. Gould, K.M. Merz, Jr., D.M. Ferguson, D.C. Spellmeyer, T. Fox, J.W. Caldwell & P.A. Kollman. A second generation force field for the simulation of proteins, nucleic acids, and organic molecules. *J. Am. Chem. Soc.* **117,** 5179-5197 (1995).

27. P.A. Kollman, R. Dixon, W. Cornell, T. Fox, C. Chipot & A. Pohorille. The development/application of a 'minimalist' organic/biochemical molecular mechanic force field using a combination of *ab initio* calculations and experimental data. In *Computer Simulation of Biomolecular Systems, Vol. 3*, A. Wilkinson, P. Weiner & W.F. van Gunsteren, Ed. Elsevier, (1997). pp. 83-96.

28. M.D. Beachy & R.A. Friesner. Accurate ab intio quantum chemical determination of the relative energies of peptide conformations and assessment of empirical force fields. *J. Am. Chem. Soc.* **119,** 5908-5920 (1997).

29. L. Wang, Y. Duan, R. Shortle, B. Imperiali & P.A. Kollman. Study of the stability and unfolding mechanism of BBA1 by molecular dynamics simulations at different temperatures. *Prot. Sci.* **8,** 1292-1304 (1999).

30. J. Higo, N. Ito, M. Kuroda, S. Ono, N. Nakajima & H. Nakamura. Energy landscape of a peptide consisting of α-helix, 3_{10} helix, β-turn, β-hairpin and other disordered conformations. *Prot. Sci.* **10,** 1160-1171 (2001).

31. T.E. Cheatham, III, P. Cieplak & P.A. Kollman. A modified version of the Cornell et al. force field with improved sugar pucker phases and helical repeat. *J. Biomol. Struct. Dyn.* **16,** 845-862 (1999).

32. S.J. Weiner, P.A. Kollman, D.A. Case, U.C. Singh, C. Ghio, G. Alagona, S. Profeta, Jr. & P. Weiner. A new force field for molecular mechanical simulation of nucleic acids and proteins. *J. Am. Chem. Soc.* **106,** 765-784 (1984).

33. S.J. Weiner, P.A. Kollman, D.T. Nguyen & D.A. Case. An all-atom force field for simulations of proteins and nucleic acids. *J. Comput. Chem.* **7,** 230-252 (1986).

34. U.C. Singh, S.J. Weiner & P.A. Kollman. Molecular dynamics simulations of d(C-G-C-G-A).d(T-C-G-C-G) with and without "hydrated" counterions. *Proc. Nat. Acad. Sci.* **82,** 755-759 (1985).

35. J. Åqvist. Ion-water interaction potentials derived from free energy perturbation simulations. *J. Phys. Chem.* **94,** 8021-8024 (1990).

36. T. Darden, D. Pearlman & L.G. Pedersen. Ionic charging free energies: Spherical versus periodic boundary conditions. *J. Chem. Phys.* **109,** 10921-10935 (1998).

37. W.L. Jorgensen, J. Chandrasekhar, J. Madura & M.L. Klein. Comparison of Simple Potential Functions for Simulating Liquid Water. *J. Chem. Phys.* **79,** 926-935 (1983).

38. W.L. Jorgensen & J.D. Madura. Temperature and size dependence for Monte Carlo simulations of TIP4P water. *Mol. Phys.* **56,** 1381-1392 (1985).

39. M.W. Mahoney & W.L. Jorgensen. A five-site model for liquid water and the reproduction of the density anomaly by rigid, nonpolarizable potential functions. *J. Chem. Phys.* **112,** 8910-8922 (2000).

40. J.W. Caldwell & P.A. Kollman. Structure and properties of neat liquids using nonadditive molecular dynamics: Water, methanol and N-methylacetamide. *J. Phys. Chem.* **99,** 6208-6219 (1995).

41. H.J.C. Berendsen, J.R. Grigera & T.P. Straatsma. The missing term in effective pair potentials. *J. Phys. Chem.* **91,** 6269-6271 (1987).

42. V. Tsui & D.A. Case. Theory and applications of the generalized Born solvation model in macromolecular simulations. *Biopolymers (Nucl. Acid. Sci.)* **56,** 275-291 (2001).

43. V. Tsui & D.A. Case. Molecular dynamics simulations of nucleic acids using a generalized Born solvation model. *J. Am. Chem. Soc.* **122,** 2489-2498 (2000).

44. A. Jakalian, B.L. Bush, D.B. Jack & C.I. Bayly. Fast, Efficient Generation of High-Quality Atomic Charges. AM1-BCC Model: I. Method. *J. Comput. Chem.* **21,** 132-146 (2000).

45. J. Wang & P.A. Kollman. Automatic Parameterization of Force Field by Systematic Search and Genetic Algorithms. *J. Comput. Chem.* **22,** 1219-1228 (2001).

46. W.C. Still, A. Tempczyk, R.C. Hawley & T. Hendrickson. Semianalytical treatment of solvation for molecular mechanics and dynamics. *J. Am. Chem. Soc.* **112,** 6127-6129 (1990).

47. T. Darden, D. York & L. Pedersen. Particle mesh Ewald--an Nlog(N) method for Ewald sums in large systems. *J. Chem. Phys.* **98,** 10089-10092 (1993).

48. U. Essmann, L. Perera, M.L. Berkowitz, T. Darden, H. Lee & L.G. Pedersen. A smooth particle mesh Ewald method. *J. Chem. Phys.* **103,** 8577-8593 (1995).

49. M.F. Crowley, T.A. Darden, T.E. Cheatham, III & D.W. Deerfield, II. Adventures in improving the scaling and accuracy of a parallel molecular dynamics program. *J. Supercomput.* **11,** 255-278 (1997).

50. C. Sagui & T.A. Darden. P3M and PME: a comparison of the two methods. In *Simulation and Theory of Electrostatic Interactions in Solution*, L.R. Pratt & G. Hummer, Ed. Melville, NY: American Institute of Physics, (1999). pp. 104-113.

51. A. Toukmaji, C. Sagui, J. Board & T. Darden. Efficient particle-mesh Ewald based approach to fixed and induced dipolar interactions. *J. Chem. Phys.* **113,** 10913-10927 (2000).

52. G.D. Hawkins, C.J. Cramer & D.G. Truhlar. Pairwise solute descreening of solute charges from a dielectric medium. *Chem. Phys. Lett.* **246,** 122-129 (1995).

53. G.D. Hawkins, C.J. Cramer & D.G. Truhlar. Parametrized models of aqueous free energies of solvation based on pairwise descreening of solute atomic charges from a dielectric medium. *J. Phys. Chem.* **100,** 19824-19839 (1996).

54. M. Schaefer & C. Froemmel. A precise analytical method for calculating the electrostatic energy of macromolecules in aqueous solution. *J. Mol. Biol.* **216,** 1045-1066 (1990).

55. M. Schaefer, H.W.T. Van Vlijmen & M. Karplus. Electrostatic contributions to molecular free energies in solution.. *Adv. Protein Chem.* **51,** 1-57 (1998).

56. D. Bashford & D.A. Case. Generalized Born Models of Macromolecular Solvation Effects. *Annu. Rev. Phys. Chem.* **51,** 129-152 (2000).

57. A. Bondi. *J. Chem. Phys.* **64,** 441 (1964).

58. J. Srinivasan, M.W. Trevathan, P. Beroza & D.A. Case. Application of a pairwise generalized Born model to proteins and nucleic acids: inclusion of salt effects. *Theor. Chem. Acc.* **101,** 426-434 (1999).

59. J. Weiser, P.S. Shenkin & W.C. Still. Approximate Atomic Surfaces from Linear Combinations of Pairwise Overlaps (LCPO). *J. Comput. Chem.* **20,** 217-230 (1999).

60. A. Onufriev, D. Bashford & D.A. Case. Modification of the Generalized Born Model Suitable for Macromolecules. *J. Phys. Chem. B* **104,** 3712-3720 (2000).

61. M. Feig, A. Onufriev, M. Lee, W. Im, D. A. Case & C. L. Brooks, III. Performance Comparison of the Generalized Born and Poisson Methods in the Calculation of the Electrostatic Solvation Energies for Protein Structures. *J. Comput. Chem.* **25,** 265-284 (2004).

62. D. Sitkoff, K.A. Sharp & B. Honig. Accurate calculation of hydration free energies using macroscopic solvent models. *J. Phys. Chem.* **98,** 1978-1988 (1994).

63. T. Morishita. Fluctuation formulas in molecular-dynamics simulations with the weak coupling heat bath. *J. Chem. Phys.* **113,** 2976 (2000).

64. H.J.C. Berendsen, J.P.M. Postma, W.F. van Gunsteren, A. DiNola & J.R. Haak. Molecular dynamics with coupling to an external bath. *J. Chem. Phys.* **81,** 3684-3690 (1984).

65. T.A. Andrea, W.C. Swope & H.C. Andersen. The role of long ranged forces in determining the structure and properties of liquid water. *J. Chem. Phys.* **79,** 4576-4584 (1983).

66. H.C. Andersen. Molecular dynamics simulations at constant pressure and/or temperature. *J. Chem. Phys.* **72,** 2384-2393 (1980).

67. R.W. Pastor, B.R. Brooks & A. Szabo. An analysis of the accuracy of Langevin and molecular dynamics algorithms. *Mol. Phys.* **65,** 1409-1419 (1988).

68. R.J. Loncharich, B.R. Brooks & R.W. Pastor. Langevin dynamics of peptides: The frictional dependence of isomerization rates of N-actylananyl-N'-methylamide. *Biopolymers* **32,** 523-535 (1992).

69. J.A. Izaguirre, D.P. Catarello, J.M. Wozniak & R.D. Skeel. Langevin stabilization of molecular dynamics. *J. Chem. Phys.* **114,** 2090-2098 (2001).

70. J.-P. Ryckaert, G. Ciccotti & H.J.C. Berendsen. Numerical integration of the cartesian equations of motion of a system with constraints: Molecular dynamics of n-alkanes. *J. Comput. Phys.* **23,** 327-341 (1977).

71. S. Miyamoto & P.A. Kollman. SETTLE: An analytical version of the SHAKE and RATTLE algorithm for rigid water models. *J. Comput. Chem.* **13,** 952-962 (1992).

72. A. Kalk & H.J.C. Berendsen. Proton magnetic relaxation and spin diffusion in proteins. *J. Magn. Reson.* **24,** 343-366 (1976).

73. E.T. Olejniczak & M.A. Weiss. Are methyl groups relaxation sinks in small proteins?. *J. Magn. Reson.* **86,** 148-155 (1990).

74. K.J. Cross & P.E. Wright. Calibration of ring-current models for the heme ring. *J. Magn. Reson.* **64,** 220-231 (1985).

75. K. Ösapay & D.A. Case. A new analysis of proton chemical shifts in proteins. *J. Am. Chem. Soc.* **113,** 9436-9444 (1991).

76. D.A. Case. Calibration of ring-current effects in proteins and nucleic acids. *J. Biomol. NMR* **6,** 341-346 (1995).

77. C.R. Sanders, II, B.J. Hare, K.P. Howard & J.H. Prestegard. Magnetically-oriented phospholipid micelles as a tool for the study of membrane-associated molecules. *Prog. NMR Spectr.* **26,** 421-444 (1994).

78. V. Tsui, L. Zhu, T.H. Huang, P.E. Wright & D.A. Case. Assessment of zinc finger orientations by residual dipolar coupling constants. *J. Biomol. NMR* **16,** 9-21 (2000).

79. D.A. Case. Calculations of NMR dipolar coupling strengths in model peptides. *J. Biomol. NMR* **15,** 95-102 (1999).

80. G. Hummer & A. Szabo. Calculation of free-energy differences from computer simulations of initial and final states. *J. Chem. Phys.* **105,** 2004-2010 (1996).

81. T. Simonson. Free energy calculations. In *Computational Biochemistry and Biophysics*, O. Becker, A.D. MacKerell, B. Roux & M. Watanabe, Ed. New York: Marcel Dekker, (2001).

82. J.P. Valleau & G.M. Torrie. In *Modern Theoretical Chemistry, Vol. 5: Statistical Mechanics, Part A, Equilibrium Techniques*, B.J. Berne, Ed. New York: Plenum Press, (1977).

83. J. Kottalam & D.A. Case. Dynamics of ligand escape from the heme pocket of myoglobin. *J. Am. Chem. Soc.* **110,** 7690-7697 (1988).

84. S. Kumar, D. Bouzida, R.H. Swendsen, P.A. Kollman & J.M. Rosenberg. The weighted histogram analysis method for free-energy calculations on biomolecules. I. The method. *J. Comput. Chem.* **13,** 1011-1021 (1992).

85. S. Kumar, J.M. Rosenberg, D. Bouzida, R.H. Swendsen & P.A. Kollman. Multidimensional free-energy calculations using the weighted histogram analysis method. *J. Comput. Chem.* **16,** 1339-1350 (1995).

86. B. Roux. The calculation of the potential of mean force using computer simulations. *Comput. Phys. Comm.* **91,** 275-282 (1995).

87. A. Mitsutake, Y. Sugita & Y. Okamoto. Generalized-ensemble algorithms for molecular simulations of biopolymers. *Biopolymers* **60,** 96-123 (2001).

88. M. Feig, J. Karanicolas & C.L. Brooks, III. MMTSB Tool Set: Enhanced sampling and multiscale modeling methods for application in structural biology. *J. Mol. Graphics Mod.*

(2004). (in press).

89. B.M. Duggan, G.B. Legge, H.J. Dyson & P.E. Wright. SANE (Structure Assisted NOE Evaluation): An automated model-based approach for NOE assignment. *J. Biomol. NMR* **19,** 321-329 (2001).

90. G.P. Gippert, P.F. Yip, P.E. Wright & D.A. Case. Computational methods for determining protein structures from NMR data. *Biochem. Pharm.* **40,** 15-22 (1990).

91. D.A. Case & P.E. Wright. Determination of high resolution NMR structures of proteins. In *NMR in Proteins*, G.M. Clore & A. Gronenborn, Ed. New York: MacMillan, (1993). pp. 53-91.

92. D.A. Case, H.J. Dyson & P.E. Wright. Use of chemical shifts and coupling constants in nuclear magnetic resonance structural studies on peptides and proteins. *Meth. Enzymol.* **239,** 392-416 (1994).

93. R. Brüschweiler & D.A. Case. Characterization of biomolecular structure and dynamics by NMR cross-relaxation. *Prog. NMR Spectr.* **26,** 27-58 (1994).

94. D.A. Case. The use of chemical shifts and their anisotropies in biomolecular structure determination. *Curr. Opin. Struct. Biol.* **8,** 624-630 (1998).

95. D.S. Wishart & D.A. Case. Use of chemical shifts in macromolecular structure determination.. *Meth. Enzymol.* **338,** 3-34 (2001).

96. A.E. Torda, R.M. Scheek & W.F. VanGunsteren. Time-dependent distance restraints in molecular dynamics simulations. *Chem. Phys. Lett.* **157,** 289-294 (1989).

97. D.A. Pearlman & P.A. Kollman. Are time-averaged restraints necessary for nuclear magnetic resonance refinement? A model study for DNA. *J. Mol. Biol.* **220,** 457-479 (1991).

98. A.E. Torda, R.M. Brunne, T. Huber, H. Kessler & W.F. van Gunsteren. Structure refinement using time-averaged J-coupling constant restraints. *J. Biomol. NMR* **3,** 55-66 (1993).

99. D.A. Pearlman. How well to time-averaged J-coupling restraints work?. *J. Biomol. NMR* **4,** 279-299 (1994).

100. D.A. Pearlman. How is an NMR structure best defined? An analysis of molecular dynamics distance-based approaches. *J. Biomol. NMR* **4,** 1-16 (1994).

101. A. Miranker & M. Karplus. Functionality maps of binding sites: A multiple copy simultaneous search method. *Proteins: Str. Funct. Gen.* **11,** 29-34 (1991).

102. X. Cheng, V. Hornak & C. Simmerling. Improved conformational sampling through an efficient combination of mean-field simulation approaches. *J. Phys. Chem. B* **108,** 2004 ().

103. C. Simmerling, T. Fox & P.A. Kollman. Use of Locally Enhanced Sampling in Free Energy Calculations: Testing and Application of the alpha to beta Anomerization of Glucose.. *J. Am. Chem. Soc.* **120,** 5771-5782 (1998).

104. J.E. Straub & M. Karplus. Enery partitioning in the classical time-dependent Hartree approximation. *J. Chem. Phys.* **94,** 6737 (1991).

105. A. Ulitsky & R. Elber. The thermal equilibrium aspects of the time-dependent Hartree and the locally enhanced sampling approximations: Formal properties, a correction, and computational examples for rare gas clusters. *J. Chem. Phys.* **98,** 3380 (1993).

106. J.J. Prompers & R. Brüschweiler. General framework for studying the dynamics of folded and nonfolded proteins by NMR relaxation spectroscopy and MD simulation. *J. Am. Chem. Soc.* **124,** 4522-4534 (2002). See esp. Eq. A14.

107. J.J. Prompers & R. Brüschweiler. Dynamic and structural analysis of isotropically distributed molecular ensembles. *Proteins* **46,** 177-189 (2002). See esp. Eq. 7.

108. B. Honig & A. Nicholls. Classical electrostatics in biology and chemistry. *Science* **268,** 1144-1149 (1995).

109. M.L. Connolly. Analytical molecular surface calculation. *J. Appl. Cryst.* **16,** 548-558 (1983).

110. J. Srinivasan, T.E. Cheatham, III, P. Kollman & D.A. Case. Continuum Solvent Studies of the Stability of DNA, RNA, and Phosphoramidate--DNA Helices. *J. Am. Chem. Soc.* **120,** 9401-9409 (1998).

111. P.A. Kollman, I. Massova, C. Reyes, B. Kuhn, S. Huo, L. Chong, M. Lee, T. Lee, Y. Duan, W. Wang, O. Donini, P. Cieplak, J. Srinivasan, D.A. Case & T.E. Cheatham, III. Calculating Structures and Free Energies of Complex Molecules: Combining Molecular Mechanics and Continuum Models. *Accts. Chem. Res.* **33,** 889-897 (2000).

112. W. Wang & P. Kollman. Free Energy Calculations on Dimer Stability of the HIV Protease Using Molecular Dynamics and a Continuum Solvent Model. *J. Mol. Biol.* **303,** 567 (2000).

113. C. Reyes & P. Kollman. Structure and Thermodynamics of RNA-protein Binding: Using Molecular Dynamics and Free Energy Analyses to Calculate the Free Energies of Binding and Conformational Change. *J. Mol. Biol.* **297,** 1145-1158 (2000).

114. M.R. Lee, Y. Duan & P.A. Kollman. Use of MM-PB/SA in Estimating the Free Energies of Proteins: Application to Native, Intermediates, and Unfolded Vilin Headpiece. *Proteins* **39,** 309-316 (2000).

115. J. Wang, P. Morin, W. Wang & P.A. Kollman. Use of MM-PBSA in Reproducing the Binding Free Energies to HIV-1 RT of TIBO Derivatives and Predicting the Binding Mode to HIV-1 RT of Efavirenz by Docking and MM-PBSA. *J. Am. Chem. Soc.* **123,** 5221-5230 (2001).

116. S. Huo, I. Massova & P.A. Kollman. Computational Alanine Scanning of the 1:1 Human Growth Hormone-Receptor Complex. *J. Comput. Chem.* **23,** 15-27 (2002).

117. S.R. Niketic & K. Rasmussen. In *The Consistent Force Field: A Documentation*, New York: Springer-Verlag, (1977).

118. C. Cerjan & W.H. Miller. On finding transition states. *J. Chem. Phys.* **75,** 2800 (1981).

119. D.T. Nguyen & D.A. Case. On finding stationary states on large-molecule potential energy surfaces. *J. Phys. Chem.* **89,** 4020-4026 (1985).

120. G. Lamm & A. Szabo. Langevin modes of macromolecules. *J. Chem. Phys.* **85,** 7334-7348 (1986).

121. J. Kottalam & D.A. Case. Langevin modes of macromolecules: application to crambin and DNA hexamers. *Biopolymers* **29,** 1409-1421 (1990).

122. C.I. Bayly, P. Cieplak, W.D. Cornell & P.A. Kollman. A Well-Behaved Electrostatic Potential Based Method Using Charge Restraints For Determining Atom-Centered Charges: The RESP Model. *J. Phys. Chem.* **97,** 10269 (1993).

123. W.D. Cornell, P. Cieplak, C.I. Bayly & P.A. Kollman. Application of RESP charges to calculate conformational energies, hydrogen bond energies and free energies of solvation. *J. Am. Chem. Soc.* **115,** 9620-9631 (1993).

124. P. Cieplak, W.D. Cornell, C. Bayly & P.A. Kollman. Application of the multimolecule and multiconformational RESP methodology to biopolymers: Charge derivation for DNA, RNA and proteins. *J. Comput. Chem.* **16,** 1357-1377 (1995).

125. S. Arnott, P.J. Campbell-Smith & R. Chandrasekaran. In *Handbook of Biochemistry and Molecular Biology, 3rd ed. Nucleic Acids--Volume II*, G.P. Fasman, Ed. Cleveland: CRC Press, (1976). pp. 411-422.

126. W.S. Ross & C.C. Hardin. Ion-induced stabilization of the G-DNA quadruplex: Free energy perturbation studies. *J. Am. Chem. Soc.* **116,** 6070-6080 (1994).

127. A. Vedani & D.W. Huhta. A new force field for modeling metalloproteins. *J. Am. Chem. Soc.* **112,** 4759-4767 (1990).

128. D.L. Veenstra, D.M. Ferguson & P.A. Kollman. How transferable are hydrogen parameters in molecular mechanics calculations?. *J. Comput. Chem.* **13,** 971-978 (1992).

129. F.H. Allen, O. Kennard, D.G. Watson, L. Brammer, A.G. Orpen & R. Taylor. *J. Chem. Soc. Perkin Trans. II* S1-S19 (1987).

130. M.D. Harmony, R.W. Laurie, R.L. Kuczkowski, R.H. Schwendemann, D.A. Ramsay, F.J. Lovas, W.J. Lafferty & A.G. Maki. *J. Phys. Chem. Ref. Data* **8,** 619 (1979).

131. A.J. Hopfinger & R.A. Pearlstein. Molecular mechanics force-field parameterization procedures. *J. Comput. Chem.* **5,** 486-499 (1985).

132. J.F. Cannon. AMBER force-field parameters for guanosine triphosphate and its imido and methylene analogs. *J. Comput. Chem.* **14,** 995-1005 (1993).

133. A. St.-Amant, W.D. Cornell, P.A. Kollman & T.A. Halgren. Calculation of molecular geometries, relative conformational energies, dipole moments, and molecular electrostatic potential fitted charges of small organic molecules of biochemical interest by density functional theory. *J. Comput. Chem.* **16,** 1483-1506 (1995).

134. A.E. Howard, P. Cieplak & P.A. Kollman. A Molecular Mechanical Model that Reproduces the Relative Energies for Chair and Twist-Boat Conformations of 1,3-Dioxanes. *J. Comp. Chem.* **16,** 243-261 (1995).

135. C.J. Cramer & D.G. Truhlar. Implicit Solvation Models: Equilibria, Structure, Spectra, and Dynamics. *Chem. Rev.* **99,** 2161-2200 (1999).

136. P. Beroza & D.A. Case. Calculations of proton-binding thermodynamics in proteins. *Meth. Enzymol.* **295,** 170-189 (1998).

137. J.D. Madura, M.E. Davis, M.K. Gilson, R.C. Wade, B.A. Luty & J.A. McCammon. Biological applications of electrostatic calculations and brownian dynamics simulations. *Rev. Computat. Chem.* **5,** 229-267 (1994).

138. M.K. Gilson. Theory of electrostatic interactions in macromolecules. *Curr. Opin. Struct. Biol.* **5,** 216-23 (1995).

139. M. Scarsi, J. Apostolakis & A. Caflisch. Continuum Electrostatic Energies of Macromolecules in Aqueous Solutions. *J. Phys. Chem. A* **101,** 8098-8106 (1997).

140. T. Simonson. Electrostatics and dynamics of proteins. *Rep. Prog. Phys.* **66,** 737-787 (2003).

141. D. Bashford & M. Karplus. pK_a's of ionizable groups in proteins: Atomic detail from a continuum electrostatic model. *Biochemistry* **29,** 10219-10225 (1990).

142. M. Schaefer & M. Karplus. A comprehensive analytical treatment of continuum electrostatics. *J. Phys. Chem.* **100,** 1578-1599 (1996).

143. S.R. Edinger, C. Cortis, P.S. Shenkin & R.A. Friesner. Solvation free energies of peptides: Comparison of approximate continuum solvation models with accurate solution of the Poisson-Boltzmann equation. *J. Phys. Chem. B* **101,** 1190-1197 (1997).

144. B. Jayaram, D. Sprous & D.L. Beveridge. Solvation free energy of biomacromolecules: Parameters for a modified generalized Born model consistent with the AMBER force field. *J. Phys. Chem. B* **102,** 9571-9576 (1998).

145. M.S. Lee, F.R. Salsbury, Jr. & C.L. Brooks, III. Novel generalized Born methods. *J. Chem. Phys.* **116,** 10606-10614 (2002).

146. B.N. Dominy & C.L. Brooks, III. Development of a Generalized Born Model Parameterization for Proteins and Nucleic Acids. *J. Phys. Chem. B* **103,** 3765-3773 (1999).

147. N. Calimet, M. Schaefer & T. Simonson. Protein Molecular Dynamics With the Generalized Born/ACE Solvent Model. *Proteins* **45,** 144-158 (2001).

148. A. Onufriev, D.A. Case & D. Bashford. Effective Born radii in the generalized Born approximation: The importance. *J. Comput. Chem.* **23,** 1297-1304 (2002).

149. J.D. Jackson. *Classical Electrodynamics.* New York: Wiley and Sons, (1975).

150. F.M. Richards. Areas, volumes, packing, and protein structure. *Ann. Rev. Biophys. Bioeng.* **6,** 151-176 (1977).

151. R. Elber & M. Karplus. Enhanced sampling in molecular dynamics. Use of the time-dependent Hartree approximation for a simulation of carbon monoxide diffusion through myoglobin. *J. Am. Chem. Soc.* **112,** 9161-9175 (1990).

152. A. Roitberg & R. Elber. Modeling side chains in peptides and proteins: Application of the locally enchanced sampling and the simulated annealing methods to find minimum energy conformations. *J. Chem. Phys.* **95,** 9277 (1991).

153. C. Simmerling & R. Elber. Hydrophobic "collapse" in a cyclic hexapeptide: Computer simulations of CHDLFC and CAAAAC in water. *J. Am. Chem. Soc.* **116,** 2534-2547 (1994).

154. C. Simmerling, J.L. Miller & P.A. Kollman. Combined locally enhanced sampling and particle mesh Ewald as a strategy to locate the experimental structure of a nonhelical nucleic acid. *J. Am. Chem. Soc.* **120,** 7149-7155 (1998).

155. C. Simmerling, M.R. Lee, A.R. Ortiz, A. Kolinski, J. Skolnick & P.A. Kollman. Combining MONSSTER and LES/PME to Predict Protein Structure from Amino Acid Sequence: Application to the Small Protein CMTI-1. *J. Am. Chem. Soc.* **122,** 8392-8402 (2000).

156. K.A. Sharp & B. Honig. Electrostatic interactions in macromolecules: Theory and experiment. *Annu. Rev. Biophys. Biophys. Chem.* **19,** 301-332 (1990).

157. M.E. Davis & J.A. McCammon. Electrostatics in biomolecular structure and dynamics. *Chem. Rev.* **90,** 509-521 (1990).

158. M.K. Gilson, K.A. Sharp & B.H. Honig. Calculating the electrostatic potential of molecules in solution: method. *J. Comput. Chem.* **9,** 327-35 (1988).

159. J. Warwicker & H.C. Watson. Calculation of the electric potential in the active site cleft due to. *J. Mol. Biol.* **157,** 671-679 (1982).

160. I. Klapper, R. Hagstrom, R. Fine, K. Sharp & B. Honig. Focussing of electric fields in the active stie of Cu, Zn superoxide dismutase. *Proteins* **1,** 47-59 (1986).

161. M.E. Davis & J.A. McCammon. Solving the finite-difference linearized Poisson-Boltmann equation -- a comparison of relaxation and conjugate gradient methods. *J. Comput. Chem.* **10,** 386-391 (1989).

162. A. Nicholls & B. Honig. A rapid finite difference algorithm, utilizing successive over-relaxation to solve the Poisson-Boltzmann equation. *J. Comput. Chem.* **12,** 435-445 (1991).

163. D. Bashford. An object-oriented programming suite for electrostatic effects in biological molecules. *Lect. Notes Comput. Sci.* **1343,** 233-240 (1997).

164. M.E. Davis & J.A. McCammon. Dielectric boundary smoothing in finite difference solutions of the Poisson equation: An approach to improve accuracy and convergence. *J. Comput. Chem.* **12,** 909-912 (1991).

165. B.A. Luty, M.E. Davis & J.A. McCammon. Electrostatic energy calculations by a finite-difference method: Rapid calculation of charge-solvent interaction energies. *J. Comput. Chem.* **13,** 768-771 (1992).

166. A. Warshel & M. Levitt. Theoretical studies of enzymic reactions: Dielectric, electrostatic and steric stabilization of the carbonium ion in the reaction of lysozyme. *J. Mol. Biol.* **103,** 227-249 (1976).

167. M.J. Field, P.A. Bash & M. Karplus. A combined quantum mechanical and molecular mechanical potential for molecular dynamics simulations. *J. Comput. Chem.* **11,** 700-733 (1990).

168. W. Yang & T.-S. Lee. A density-matrix divide-and-conquer approach for electronic structure calculations of large molecules. *J. Chem. Phys.* **103,** 5674-5678 (1995).

169. S.L. Dixon & K.M. Merz, Jr.. Semiempirical molecular orbital calculations with linear system size scaling. *J. Chem. Phys.* **104,** 6643-6649 (1996).

170. S.L. Dixon & K.M. Merz, Jr.. Fast, accurate semiempirical molecular orbital calculations for macromolecules. *J. Chem. Phys.* **107,** 879-893 (1997).

171. R.V. Stanton, D.S. Hartsough & K.M. Merz, Jr.. An examination of a density functional/molecular mechanical coupled potential. *J. Comput. Chem.* **16,** 113-128 (1994).

172. R.V. Stanton, L.R. Little & K.M. Merz, Jr.. An examination of a Hartree-Fock/molecular mechanical coupled potential. *J. Phys. Chem.* **99,** 17344-17348 (1995).

173. J. Gao. Absolute free energy of solvation from Monte Carlo simulations using combined quantum and molecular mechanical potentials. *J. Phys. Chem.* **96,** 537-540 (1992).

174. R.V. Stanton, D.S. Hartsough & K.M. Merz, Jr.. Calculations of solvation free energies using a density functional/molecular dynamics coupled potential. *J. Phys. Chem.* **97,** 11868-11870 (1993).

Index

This index is designed to help locate information for particular variable names. The Table of Contents should be used to identify subject areas.