AMBER 7
Users' Manual

Authors:
David A. Case (The Scripps Research Institute)
David A. Pearlman (Vertex Pharmaceuticals)
James W. Caldwell (UCSF)
Thomas E. Cheatham III (University of Utah)
Junmei Wang (UCSF)
Wilson S. Ross (UCSF)
Carlos Simmerling (SUNY Stony Brook)
Tom Darden (NIEHS)
Kenneth M. Merz (Penn State)
Robert V. Stanton (UCSF)
Ailan Cheng (Penn State)
James J. Vincent (Penn State)
Mike Crowley (TSRI)
Vickie Tsui (TSRI)
Holger Gohlke (TSRI)
Randall Radmer (UCSF)
Yong Duan (UCSF)
Jed Pitera (UCSF)
Irina Massova (UCSF)
George L. Seibel (for contributions to Amber version 3A while at UCSF)
U. Chandra Singh (for contributions to Amber versions 2 and 3 while at UCSF)
Paul Weiner (for contributions to Amber version 1 while at UCSF)
Peter A. Kollman (UCSF)

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 7 in the literature, the following citation should be used:

> D.A. Case, D.A. Pearlman, J.W. Caldwell, T.E. Cheatham III, J. Wang, W.S. Ross, C.L. Simmerling, T.A. Darden, K.M. Merz, R.V. Stanton, A.L. Cheng, J.J. Vincent, M. Crowley, V. Tsui, H. Gohlke, R.J. Radmer, Y. Duan, J. Pitera, I. Massova, G.L. Seibel, U.C. Singh, P.K. Weiner and P.A. Kollman (2002), AMBER 7, 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 7 to his memory.

Cover Illustration

The figure shows the hydration pattern at the interface of a zinc-finger protein-DNA complex. The protein is represented by its molecular surface, with finger 1 at the bottom right and figure 3 at the top left; the DNA is colored green. Water molecules within 2.3 Å of at least one atom of the protein and one atom of the DNA are shown as spheres. Waters whose computed residence times in the interface are > 0.8 ns are colored red, and more transient bridging waters are colored blue. Figure was prepared by Vickie Tsui, and re-rendered by David Case and Mike Pique. See V. Tsui, I. Radhakrishnan, P.E. Wright and D.A. Case, *J. Mol. Biol.* **302,** 1101-1117 (2000).

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. 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 7 (2002) represents a significant change from the most recent previous version, *Amber 6*, which was released in December, 1999. Briefly, the major differences include:

(1) Several new force fields are available for proteins and nucleic acids. These include versions with polarizable dipoles on atoms, and off-center charges (also called "extra points", and analogous to lone pairs). Amber now provides direct support for TIP3P, TIP4P, TIP5P, SPC/E and POL3 models of water, as well as providing models for chloroform and other organic solvents.

(2) A new "general amber force field" that should be applicable to most organic molecules. The automated code to prepare Amber input files using this force field is a new module, called Antechamber. In most cases, Antechamber can directly convert three-dimensional models into files appropriate for molecular mechanics calculations, automatically assigning atom types, charges and force field parameters.

(3) Implementation of three new variants of the generalized Born (GB) code, including one that appears to provide a better energy balance between surface-exposed and buried atoms.

(4) More efficient PME simulations, with better performance on both single-processor and parallel machines.

(5) Updated scripts for MM_PBSA analysis, making input easier to create and providing more options for analysis of the results.

(6) Free energy calculations using the thermodynamic integration method can now be carried out in *sander*. Many investigations that used to require *gibbs* can now be carried out in a simpler fashion, and free energy studies using the GB model or "extra points" force fields (which are not supported with *gibbs*) can now be undertaken.

(7) New types of restraint forces can be defined that are based on RMS superpositions to reference structures. This "targeted MD" capability can be used to enhance or guide conformational sampling.

1.1. What to read next

If you are installing this package or want to redimension the code, see Section 1.3. New users should continue with this Chapter, and should consult the tutorial information in Section 1.4. 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://www.amber.ucsf.edu/amber.html`. 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 [2], to standard works on simulation methods [3,4], to multi-author compilations that cover many important aspects of biomolecular modelling [5-7]. Looking for "paradigm" papers that report simulations similar to ones you may want to carry out is also generally a good idea.

Users of previous versions of Amber should be aware that most things in this version are backwards-compatible, but there are some key changes. Among the things to be aware of are:

(1) The force field and parameter files have been completely re-organized. Even experienced Amber users will need to read Chapter 2 on "selecting a force field." In particular, there is no default *leaprc* file, although you can create one if you wish.

(2) The *igb* parameter in *sander* now has different meaning than in version 6.

(3) Kinetic energies in *sander* are evaluated in a slightly different way than in previous versions. This improves the (apparent) conservation of total energy. The change will have no effect on the course of trajectories run without temperature coupling (*i.e.* with *ntt*=0); there will be slight differences in temperature-coupled trajectories compared with earlier versions.

(4) There is a new format for *prmtop* files that allows for systems larger than 100,000 atoms, and which will be easier to modify for future development. All the Amber programs will automatically detect and read either the new or old formats. For the benefit of other programs that might still expect the earlier format, such files can still be created, either through a switch in LEaP, or by using *new2oldparm*, which acts as a pure filter, converting a "new" format file on *stdin* to an "old" format file on *stdout*.

(5) The *sander_classic* program has been retired. You can now run non-periodic simulations in *sander* by setting *ntb=0*.

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 (*gibbs, sander, roar* and *nmode*). You need to know where it comes from, and how it gets into the form that the energy programs need. 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 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 *amber7/dat/leap/prep* directory, and is described in Chapter 2. It contains topology for the standard amino acids as

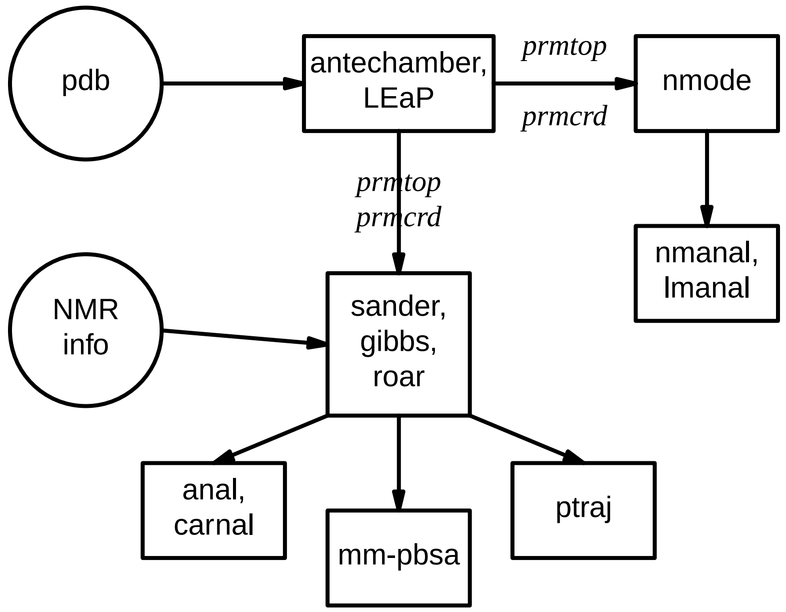


Fig. 1. Basic information flow in Amber

well as N and C-terminal charged amino acids, DNA, and RNA. 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 *amber7/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, gibbs, 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

This 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 intergration may be carried out.

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.

GIBBS is the free energy perturbation program. It is similar to SANDER, but uses the ensemble of generated configurations to calculate the free energy difference between two similar states through either a perturbation or thermodynamic integration approach. The two states are defined by the user in LEaP.

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,) techniques to find transitions states as well as minima, and programs to generate "quasiharmonic" modes (sometimes called Principal Component Analysis) from MD trajectories.

ROAR is a "Penn State" version of sander, that incoporates a variety of features not found in sander itself. The most notable change is the incorporation of the ability to define a part of the system quantum-mechanically, allowing it to interact with other parts of the system that are defined in a molecular mechanics sense. Other features of ROAR include implementation of a Nose-Hoover-chain MD integrator, Ewald summations, and multiple-time-scale integration routines.

1.2.3. Analysis programs

ANAL is for the analysis of structure and especially molecular mechanical energy of a single configuration of a system. It can be run on structures both before and after modification by the energy programs. Running ANAL on the initial configuration of your system is a good way to locate errors in the structure that result in large energies.

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.
CARNAL	is a second molecular dynamics analysis package, with an interface and some features that complement those in *ptraj*.
NMANAL/LMANAL	computes atomic fluctuations and various correlation functions from normal or quasiharmonic modes.
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 7

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/amber7

Using bash, sh, zsh, etc:     set AMBERHOME=/usr/local/amber7
                              export AMBERHOME
```

NOTE: Be sure to replace the "/usr/local" part above with whatever path is appropriate for your machine. You will probably want to place `$AMBERHOME/exe` in your PATH.

(2) Go to the `$AMBERHOME/src` directory, and create a link to the appropriate Machine file, e.g.

```
ln -s -f Machines/Machine.g77 MACHINE   for Linux with g77 compiler
ln -s -f Machines/Machine.sgi_nopar MACHINE   for single-CPU SGI, etc
```

To compile sander for SGI parallel machines, you will need to use Machine.sgi_mpi (for SGI's mpi libraries) or Machine.sgi_mpich (if you have installed MPICH). For most Intel/Linux distributions, you can use Machine.g77 for a single processor version, or Machine.g77_mpich for multiple processors (again, assuming you have MPICH installed.) If you have purchased commerical compilers for Linux, you might want to look at Machine.pgf77 (for the Portland Group compiler), Machine.absoft (for the Absoft Fortran compiler), or Machine.ifc (for the Intel Fortran compiler). Use these as a template if you have some other compiler. If you have an Alpha/Linux machine, try Machine.axp_linux.

(3) Now compile everything:

```
make install
```

Loader warnings (especially on SGI -- see the Machine.sgi file) 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 commenting out that line in the `Makefile`, and seeing if the rest of the suite can be compiled correctly.

(4) In order to use the *cm2, mul,* or *bcc* charge options in *antechamber*, you will need to have a version of MOPAC installed on your machine. There are many places to get this; a version that supports the CM2 charge model can be obtained from *http://comp.chem.umn.edu/mopac/*. You will then need to edit *$AMBERHOME/exe/mopac.sh* to point to the correct location of MOPAC. The syntax should be "mopac_program input_file output_file"; depending on which version of mopac you have, you may need to remove the "<" and ">" symbols in the *mopac.sh* file we provide by default. (Obviously, if you don't plan to use *antechamber* with these charge options, you don't need to do this step!) *Note:*, the version of MOPAC distributed at the above web site is dimensioned by default for only 15 heavy atoms; you will probably want to increase the dimensions in the `SIZES.i` file before compiling this.

(5) 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
```

(6) To test the basic AMBER distribution, type "make test.<program-name>" in the $AMBERHOME/test directory. Currently, <program-name> can be one of the following: leap, sander, gibbs, anal, nmode, resp, antechamber. To test parallel programs, you need first to set the DO_PARALLEL environment variable as follows:

```
setenv DO_PARALLEL 'mpirun -np 4'
```

The number 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.

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 comment out certain lines in the Makefile, and/or go directly to the $AMBERHOME/test subdirectories to examine the inputs and outputs in detail.

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, gibbs* and *roar* are parallel programs; all others are single threaded. For information on parallel roar, see its documentation. 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!

This release supports both shared memory (fortran directives) and message passing (MPI) code for sander and gibbs for particular architectures.

```
          Shared memory: (for gibbs)

              All SGI multiprocessors
              Cray Unicos multiprocessors running Unicos

          Message passing environments: (MPI, for sander and gibbs)

              (Presumably) any MPI-compliant parallel environment; we have
                 tested the following:
              SGI with MPICH library
              SGI with vendor-supplied MPI library
              Cray T3D/E using SHMEM wrappers to intercept MPI calls
              Cray T3E with Cray-supplied MPT library
              IBM POE MPL (SP1/SP2)
              Linux clusters with various MPI libraries, including MPICH
              HP clusters
```

1.3.1.1. Shared memory version

The shared memory version was originally developed by Roberto Gomperts and Michael Schlenkrich of SGI, with the assistance of Thomas Cheatham, specifically for Silicon Graphics multiprocessors. The SGI shared memory version was ported to Cray multiprocessors by Jeyapandian Kottalam and Mike Page of Cray Research and incorporated into the 4.1 distribution by Thomas Cheatham. In Amber 7, the shared memory option only applies to *gibbs*.

Currently the code has only been ported to SGI and Cray multiprocessors. Multitasking is handled via the placement of FORTRAN parallel directives in the source which multitask over loops or parallel regions. Scratch memory is allocated on an as-needed basis using SGI and Cray specific FORTRAN calls to allocate memory.

In general, the shared memory source code is all wrapped with the CPP wrapper:

```
      #ifdef SHARED_MEMORY

          ...general shared memory code...

      # ifdef SGI_MP

          ...SGI specific code...

      # endif
      # ifdef CRAY_MP

          ...Cray specific code...

      # endif

      #endif
```

Specification of either -DSGI_MP or -DCRAY_MP in the MACHINEFLAGS of the MACHINE file leads to auto-setting of the #define SHARED_MEMORY in the code. [Note that specifying -DSHARED_MEMORY alone in the MACHINEFLAGS will not lead to a correctly compiled source.]

Beware that some of the shared memory source is placed in extra files which are included when the appropriate #define's are specified. This is to minimize obfuscation of the source where possible. In particular, this is apparent in force.f (#include forcemp.f) and resnba.f (#include resnbamp.f).

In addition to setting either -DSGI_MP or -DCRAY_MP in the MACHINEFLAGS, optionally one can specify the maximum number of processors for a given executable (which determines allocation of scratch memory) to be compiled in by adding the following define to the MACHINEFLAGS:

```
-DSHARED_MEMORY_MAX_PROCESSORS=N
```

where N is the maximum number of processors to run on. If this is not set, it will default to the current maximum size for a given machine based on default values set in sander.f (sander) and gib.f (gibbs) and in various other routines.

To set the number of processors to run on, set the MP_SET_NUMTHREADS environment variable at runtime. E.g. to run on 4 processors:

```
setenv MP_SET_NUMTHREADS 4
```

1.3.1.2. MPI version

This message passing version was initially developed by James Vincent and Ken Merz, based on 4.0 and later an early prerelease 4.1 version [8]. This version was optimized, integrated and extended by James Vincent, Dave Case, Thomas Cheatham, 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 dynamics (*runmd()*, more optimal) in parallel. Communication is done to accumulate partial forces, update coordinates, etc.

The MPI source code is generally wrapped with the CPP wrapper:

```
#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 MACHINE file (these files end in "_mpi" in the src/Machines directory) then you will need to modify the Machine file as follows:

```
        (1) add "-DMPI " to the MACHINEFLAGS variable.

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

            setenv MACHINEFLAGS "-DMPI -I/usr/local/src/mpi/include"

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

        (4) Add any special compile and load flags to the L0, L1, L2,
            L3, and LOAD variables.
```

To run the resulting codes, you need to use the "mpirun" (or equivalent) command for your system. The naming and syntax of this is unfortunately not well standardized: on the T3E it is called "mpprun"; on DEC OS/1 machines it is called "dmpirun"; etc. Consult your MPI documentation.

For the test suites, you need to set the DO_PARALLEL environment variable to include mpirun or its equivalent, e.g.:

```
    setenv DO_PARALLEL 'mpirun -np 4'
```

will run the test cases with four processors.

1.3.2. Installing on Windows

Most of Amber (including the X-windows parts) will compile and run on Windows using the Cygwin development tools, but doing this requires some knowledge of Unix-style software development tools. You should first have familiarity with Amber on a Unix/Linux system, and then treat porting the parts you need to the new environment as you would port any other Unix-based software to the Cygwin environment. Note that, although tests seem to perform correctly for us, this platform has *not* had extensive testing, and may have unidentified limitations.

Note that the executables constructed in this way require the cygwin dll to run, so that you will have to install the cygwin tools on every machine on which you wish to run Amber. Note also that there are restrictions on distributing programs that link to the cygwin libraries. (See the cygwin Web site for details.) The basic upshot is that we cannot supply pre-compiled Windows binaries created by this mechanism: you will need to compile things yourself. Use *Machine.g77* as your MACHINE file, and edit *amber7/src/sfx.h* to set `$SFX=.exe`.

1.3.3. Testing.

We have installed and tested AMBER 7 on a number of machines, 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 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 among other things 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. As is the case with sander, the trajectory in a Gibbs simulation will diverge, but the resulting free energy should not if the simulation is run to convergence (this is not done because of the time involved). In minimization and dynamics calculations, it is not unusual to see small divergences in behavior after as little as 100-200 cycles.

In general, compiler and optimizer errors are fairly obvious, and result in rather large changes in the output, if you get any output at all. See `test/0README` for examples of acceptable output differences and discussion of peculiarities of various machines.

1.3.4. Memory Requirements.

The AMBER 7 programs as distributed are dimensioned for a medium-sized system (about 30,000 atoms), and you may want to change their dimensions to be more appropriate for the machine you are using, especially if you are running in a tight memory environment. In `src/sander/sizes.h`, you will find parameters that might need to be changed. Instructions for choosing appropriate sizes are given in that file. Make the required changes, then recompile. Note that some sizes related to NMR refinements are defined in `nmr.h`. In *sander*, the MEM_ALLOC flag can be turned on during compilation; this for the most part (but not completely) relieves the user of concerns about memory allocation.

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; if this is comparable to are larger than your machine’s memory, you may need to re-compile with smaller sizes. Generally, all of the test cases should run (using the default sizes) on a machine with 128 Mbytes of memory, but probably not on anything much smaller.

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.

Additional tutorial examples are available on at *http://www.amber.ucsf.edu/amber/*. 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 adivsed to look at the following, which can be found at both at the web site listed above, and on the distribution CD, under *amber7/tutorial*:

DNA	Basic introduction to *LEaP*, *sander*, *carnal* 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.
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.
streptavidin/biotin	Illustrates a ligand-protein binding simulation, using a non-periodic "cap" of waters to hydrate just the active site.
methane/water	A small molecule calculation, illustrating free energy perturbation on methane in a box of water.

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 *amber7/tutorial* on 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 minimzation 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 residue 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 again need to have their residue names changed in an editor from CYS to CYX to reflect this. 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. Type the following command should work in either *tleap* or *xleap*:

```
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,
 &end
eof
sander -i min.in -o 6pti.min1.out -c prmcrd -r 6pti.min1.xyz
/bin/rm min.in
```

This will perform minimization (`imin`) for 200 steps (`maxcyc`), using a nonbonded cutoff of 12 Å (`cut`) and a generalized Born solvent model (`igb=1`); 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 paramters and a history of the progress of the minimization.

Of course, Amber can do much more than the above minimization, and the tutorials in *amber7/tutorial* should be consulted to go further.

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. One significant recent development is that 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 *ff86*, *ff94*, *ff96*, *ff98* and *ff99* (described below). The default in versions 5 and 6 of Amber was *ff94*; a comparable default now would probably be *ff99*, 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 *amber7/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 *amber6/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 *amber7/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 *amber7/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.

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           Newer (and still experimental) force field for quite
                      general organic molecules.

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

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        paramters for methanol
frcmod.chcl3       paramters for chloroform
```

`frcmod.nma`	`paramters for N-methyacetamide`

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.ff02	reduced (polarizable) charges	parm99.dat
leaprc.ff02EP	" + extra points	parm99EP.dat
leaprc.gaff	none	gaff.dat

Notes:

(1) Unlike previous versions of Amber, *there is no default leaprc file* anymore. 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.ff99 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 *ff99* 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 five 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 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 all with *ff94* or *ff99*; the current GB models are not compatible with polarizable force fields. The GB options *igb*=3 or 4 (see Chapter 5) were derived for use with *ff94*. 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. 1999 and 2002 force fields

The **ff99** force field [9] represents a new direction for Amber-related force fields, pointing towards "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. This is probably the best "general purpose" force field included here for biomolecules.

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 [10]. 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 [11]. 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 [10,12]. Beyond small molecules, our intial 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. It is primarily used in conjunction with the *antechamber* program, and users should consult that chapter for more information. A paper describing these parameters is being prepared for publication.

2.4. 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 [13]. 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 modelling 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 Appendix D.

The **ff96** force field [14] differs from *parm94.dat* in that the torsions for ϕ and ψ have been modified in response to *ab initio* calculations [15] 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 idiosyncracies [16,17].

The **ff98** force field [18] 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.

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

The **ff86** parameters are described in early papers from the Kollman and Case groups [19,20]. [The "parm91" designation is somewhat unfortunate: this file is really only a corrected version of the paramters 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.*

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 1985 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* uses a Lennard-Jones 10-12 potential. If you want to run *sander* with *ff86*, you will need to recompile, adding the -DHAS_10_12 flag to your MACHINE file.

2.6. Ions

For alkali ions with TIP3P waters, we have provided the values of Åqvist [21], 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 [22].

2.7. Solvent models

Amber now provides direct support for several water models. The default water model is TIP3P [23]. 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 *water.lib* file contains TIP3P [23], TIP4P [23,24], TIP5P [25], POL3 [26] and SPC/E [27] 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 (in order to keep track of which water model you are using, say), 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. The input files for a single molecule are in *amber7/dat/leap/prep*, and the corresponding frcmod files are in *amber7/dat/leap/parm*. Pre-equlibrated boxes are in *amber7/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 and SPASMS
```

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 of 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, 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 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 made up 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 a type of calculation known as Free Energy Perturbation. During Free Energy Perturbation, one chemical species is slowly transformed into another and the energy change associated with the transformation is measured. In order to perform a Free Energy Perturbation, 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 Perturbation calculation is not to be performed, the following properties can be left as `null`. They are only used when the "PERTURB" property's value is "true" for that atom, when doing a `saveAmberParmPert` to save a perturbation topology file. (Note that mass is never perturbed.)

`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 Perturbation 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` then the ATOM `type` will not be perturbed in a perturbation calculation. If the `pertType` is a STRING, the STRING is the AMBER force field atom type of the perturbed ATOM. This property is case-sensitive.

`pertCharge`

The `pertCharge` property is a NUMBER. It represents the final electrostatic point charge on an ATOM during a Free Energy Perturbation.

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 RESIDUEs' `connect0` ATOM is usually defined as the UNITs' `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 RESIDUEs' `connect1` ATOM is usually defined as the UNITs' `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 disulphide 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 "ˆ> " | sed "s/ˆ> //" > SOURCEFILE.x
```

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

3.4. Using LEaP

In the next two sections, we describe how to use the tleap and xleap user interfaces. Strategies for using LEaP in research are discussed in a subsequent section: "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 cursor 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 section 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 Free Energy Perturbation 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 "/". (See the "Concepts" section for a discussion of Free Energy Perturbation ATOM types.)

"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 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 cursor 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 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-`Select`ed atoms. While in the `Move` mode, the 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 section 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 cursor to resemble a chalkboard eraser when it is in the viewing window. Clicking the left-button will delete any atoms or bonds under this cursor, one atom or bond per click.

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

When the 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 pointer is in the viewing window is the keyboard spacebar, which centers the view of the molecule, and 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.) The column labelled "PERTURB" should be blank (or "false") if that atom is not being perturbed, and should be set to "true" otherwise.

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*, or *loadAmberPrep*, or use the *zmat* command to build a molecule from a z-matrix. See the Commands section below for desciptions of these options. For case where you do not have a starting structure (in the form of a pdb file) LEaP can be used to build the molecule; you will find, however, that this is not always as easy as it might be. Many experienced Amber users turn to other (commerical and non-commerical) 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 items` 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 are 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.

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

CALA}. If the user models a peptide or protein within LEaP, they may choose one of three ways 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 which participates in disulfide bridges, cystine (CYX). 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. In addition, 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.

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 are end in "N," such as "DAN."

3.5.4. Miscellaneous Residues

Miscellaneous Residue	unit/residue name
TIP3P water molecule	TP3
Periodic box of TIP3P water	WATBOX216
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 Chapter 3 for a discussion of how to use other water models.

A periodic box of 216 TIP3P waters (WATBOX216) 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 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 commmands. 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 default, the bond will be a single bond. 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 WATBOX216 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 WATBOX216 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. 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 disulphide 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 disulphide bonds using the *bond* command.

3.6.31. 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".

3.6.32. 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
```

3.6.33. 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
```

3.6.34. quit

Quit the LEaP program.

3.6.35. 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>
```

3.6.36. saveAmberParm

```
saveAmberParm unit topologyfilename coordinatefilename
```

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

Save the AMBER/SPASMS 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 and SPASMS for calculations. The output of this operation can be used for minimizations, dynamics, and thermodynamic perturbation 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.
```

3.6.37. 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. Note: charges are normally scaled when polarizabilities are used - see scaleCharges for an easy way of doing this.

3.6.38. 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. This command produces topology files and coordinate files that are identical in format to those produced by AMBER PARM and can be read into gibbs for perturbation calculations.

```
> 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.
```

3.6.39. saveAmberParmPolPert

```
saveAmberParmPolPert unit topologyfilename coordinatefilename
```

```
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. Note: charges are normally scaled when polarizabilities are used - see scaleCharges for an easy way of doing this.

3.6.40. 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.)

3.6.41. 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
```

3.6.42. 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 WATBOX216
> scalecharges y 0.875
> solvatebox x y 10
> saveamberparmpol x x.top x.crd
```

3.6.43. 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>
```

3.6.44. 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 "bondi" to use Bondi radii and Tinker screening parameters for generalized Born calculations. These values are recommended when *gblambda* > 1 in the *sander* input.

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 [28], and are the recommended ones when *gblambda* = 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 [29], 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 "pbamber" to use radii optimized by Huo and Kollman for use in PB calculations with Amber charges. [need citations here...]

Set to "gbjsb" or "mgbjsb" to use the radii and screening parameters derived by Jayaram, Sprous and Beveridge [30] for the "regular" or "modified" GB models, respectively. These are the parameters that should be used when *igb* = 3 or 4 in *sander.*

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.

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 Perturbation calculation.
pertType	The STRING is the AMBER force field atom type of a perturbed ATOM.
pertCharge	This NUMBER represents the final electrostatic point charge on an ATOM during a Free Energy Perturbation.

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 disulphide 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.

3.6.45. 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.

3.6.46. solvateBox

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

      UNIT    solute
      UNIT    solvent
```

```
object  buffer
NUMBER  closeness
```

The *solvateBox* command creates a rectangular parallelopiped 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 WATBOX216, which is a snapshot from a room-temperature equilibration for this model. If you want to solvate with other water models (say TIP4P), try the following: *(a)* solvate the system with WATBOX216, using the default TIP3 model; *(b)* use *ambpdb* to convert your *prmtop* file to Brookhaven format; *(c)* restart LEaP, choose the TIP4P water model (instructions are in the Database chapter), then use loadPdb to bring back in the system you have created.

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 WATBOX216, has been loaded already (as is done by the default leaprc):

```
>> mol = loadpdb my.pdb
>> solvateBox sol WATBOX216 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.)

3.6.47. 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 WATBOX216 mol.2.CA 8.0 2.0
Added 3 residues.
```

3.6.48. 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 WATBOX216 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.

3.6.49. 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.)

3.6.50. 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 WATBOX216 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.
```

3.6.51. 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.

3.6.52. 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  }
 }
```

3.6.53. 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 }
```

3.6.54. 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
```

3.6.55. 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, August, 2001, and is designed to be used in conjunction with the "general Amber force field" (*gaff.dat*). References are "Antechamber, an Accessory Software Package for Molecular Mechanical Calculations", by Junmei Wang, Wei Wang and Peter A. Kollman, (manuscript in preparation), and "Development of General AMBER Force Field (GAFF)", by Junmei Wang, Romain M. Wolf, David A. Case and Peter A. Kollman" (manuscript in preparation).

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 field has enjoyed 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 traditional Amber, atom types in GAFF are more general and cover most of the organic chemical space. In total there are 35 atom types. The charge methods used in GAFF are HF/6-31G* RESP and AM1-BCC [31]. 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 because of its simplicity and efficiency.

The van der Waals parameters are same as those used by the traditional AMBER force field. The equilibrium bond lengths and bond angles come 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 field. General torsional angle parameters were extensively applied in order to reduce the huge number of torsional angle parameters to be parameterized. The force constants and phase angles were optimized using our PARMSCAN package [32], 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)].

Two main tests have been carried out (so far) to evaluate GAFF. In the first test, GAFF was used to optimize the crystal structures of 75 molecules, which were also used by Tom Halgren to evaluate his MMFF [33]. We achieved comparable performance to that of MMFF. The RMS deviations of bond length and bond angle are 0.025 Å and 2.4^o, respectively. This is encouraging, given that the functional form of GAFF is much simpler than that of MMFF. In a second test, GAFF was used to calculate the intermolecular energies of 26 base pairs, which have high-level ab initio energies available. The performance of GAFF was comparable to that of traditional Amber force fields (*ff94* and *ff99*).

By design, GAFF is a complete force field (so that missing parameters rarely occur), it covers almost all the organic chemical space, and it is compatible with the Amber macromolecular force fields. We believe that the combination of GAFF with traditional Amber will offer an useful molecular mechanical tool for rational drug design, especially for things like binding free energy calculations.

4.1. Principal programs

The *antechamber* program itself is the main program for 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. Some additional suggestions for parameters can be generated by running *parmcal*.

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: *mopac*, *atomtype*, *bcc*, *bcctype*, *espgen*, *respgen* and *prepgen*. It may also generate lots of intermediate files (all in capital letters). If there is a problem with *antechamber*, you may want to run its subprograms separately; these are described below.

```
Usage: antechamber
               -i   input file name
               -o   output file name
               -a   additional file name, optional
               -fi  input file format
               -fo  output file format
               -fa  additional file format, optional
               -c   charge method, optional
               -cf  charge filename, optional
               -nc  net molecular charge (integer)
               -m   multiplicity (2S+1), default is 1
               -rn  residue name, default is MOL
               -rf  residue topology file name in prep input file,
                       default is molecule.res
               -mp  mopac program name, default is mopac.sh
               -mk  mopac keywords (enclose in quotes)
               -gk  gaussian keywords (enclose in quotes)
               -at  atom type: can be gaff or amber, default is gaff
               -j   atom type and bond type prediction index, optional
                    both: assign both atom and bond types, default
                    at:   assign only atom types
                    bt:   assign only bond types
               -s   status information, can be 0 (brief) ,1 (the
                        default) and 2(verbose)
               -pf  remove intermediate files: can be yes (y) or no
                        (n), default is no
```

```
(-i, -o,-fi, and -fo must be specified; other arguments 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   |    Modified PDBl      mpdb    4
 AMBER PREP (int)   prepi    5   |    AMBER PREP (car)   prepc   6
 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
--------------------------------------------------------------------

             List of the Charge Methods

 charge method     abbre.  index |   charge method      abbre. index
---------------------------------------------------------------------
 RESP              resp     1   |    AM1-BCC            bcc     2
 CM2 (Kollman)     esp      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.prep -fo prepi -c resp
antechamber -i g98.out -fi gout -o sustiva_bcc.prep -fo prepi -c bcc
antechamber -i g98.out -fi gout -o sustiva_gas.prep -fo prepi -c gas
antechamber -i g98.out -fi gout -o sustiva_cm2.prep -fo prepi -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 when creating Amber prep files; thus 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 lower case, and the Amber atom types are always 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. Note that *parmchk* only creates parameters for the gaff force field.

4.1.2. parmchk

Parmchk reads in an ac file or a prep input 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 problem parameters indicated with "ATTN, need revision".

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

Example:

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

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

4.1.3. parmcal

Parmcal is an interactive program to calculate the bond length and bond angle parameters, according to rules outlined in *cite paper here*.

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

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, 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, care 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, that looks like this:

```
ATOM      1  CG  TP      1      -1.959   0.102   0.795  1.00  0.00
ATOM      2  CD1 TP      1      -1.249   0.602  -0.303  1.00  0.00
ATOM      3  CD2 TP      1      -2.071   0.865   1.963  1.00  0.00
ATOM      4  CE1 TP      1      -0.646   1.863  -0.234  1.00  0.00
ATOM      5  C6  TP      1      -1.472   2.129   2.031  1.00  0.00
ATOM      6  CZ  TP      1      -0.759   2.627   0.934  1.00  0.00
ATOM      7  HE2 TP      1      -1.558   2.719   2.931  1.00  0.00
ATOM      8  S15 TP      1      -2.782   0.365   3.060  1.00  0.00
ATOM      9  H19 TP      1      -3.541   0.979   3.274  1.00  0.00
ATOM     10  H29 TP      1      -0.787  -0.043  -0.938  1.00  0.00
ATOM     11  H30 TP      1       0.373   2.045  -0.784  1.00  0.00
ATOM     12  H31 TP      1      -0.092   3.578   0.781  1.00  0.00
ATOM     13  H32 TP      1      -2.379  -0.916   0.901  1.00  0.00
```

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

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

This command says the input format is pdb, output format is prepin, and the BCC charge model is to be used. This produces the output file *tp.prepin*, (shown in the box on the next page). The format of this file is that of the now-obsolete *prep* program, but LEaP can also read this, using the loadAmberPrep command. Generally, users should have no need to modify this file.

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

```
parmchk -i tp.prepin -f prepi -o frcmod
```

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 terms were assigned a default value. (As *gaff.dat* continues to be developed, there should be fewer and fewer missing *tp.prepin*

```
    0    0    2

This is a remark line
molecule.res
TP     XYZ  0
CORRECT     OMIT DU   BEG
  0.0000
   1  DUMM  DU    M    0  -1  -2     0.000      .0        .0      .00000
   2  DUMM  DU    M    1   0  -1     1.449      .0        .0      .00000
   3  DUMM  DU    M    2   1   0     1.522   111.1        .0      .00000
   4  CG    ca    M    3   2   1     1.540   111.208   180.000    -0.119
   5  H32   ha    E    4   3   2     1.106    67.689    -5.945     0.143
   6  CD1   ca    M    4   3   2     1.400   120.476   114.483    -0.114
   7  H29   ha    E    6   4   3     1.016   119.603  -105.804     0.135
   8  CE1   ca    M    6   4   3     1.399   120.112   103.689    -0.137
   9  H30   ha    E    8   6   4     1.172   119.429   145.095     0.133
  10  CZ    ca    M    8   6   4     1.400   119.867    -0.280    -0.112
  11  H31   ha    E   10   8   6     1.172   106.739   174.651     0.133
  12  C6    ca    M   10   8   6     1.400   120.043     0.105    -0.145
  13  HE2   ha    E   12  10   8     1.080   119.962   179.978     0.130
  14  CD2   ca    M   12  10   8     1.400   120.059     0.129     0.017
  15  S15   sh    M   14  12  10     1.400   120.111   179.881    -0.256
  16  H19   hs    E   15  14  12     0.999   109.520    59.997     0.191



LOOP
  CD2   CG

IMPROPER
  CD2  CD1   CG  H32
   CG  CE1  CD1  H29
  CD1   CZ  CE1  H30
   C6  CE1   CZ  H31
  CD2   CZ   C6  HE2
   C6   CG  CD2  S15

DONE
STOP
```

parameters to be estimated by *parmchk*.) 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
loadAmberPrep tp.prepin
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, after the loadAmberPrep step, you can insert commands 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 effect, 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 *amber7/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) Most failures caused 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 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 to *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 <junmei@cgl.ucsf.edu> or <jwang@tbc.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 assign the atom types. You may find the default definition files in $AMBERHOME/dat/antechamber: ATOMTYPE_AMBER.DEF (amber), ATOMTYPE_GFF.DEF (general force field). ATOMTYPE_GFF.DEF is the default definition file

```
Usage atomtype -i inputfile (ac)
               -o outputfile (ac)
               -p amber or gff, it is suppressed by "-d" option
               -d atom_type_definition_file, optional
```

Example:

```
atomtype -i sustiva_resp.ac -o sustiva_resp_at.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. bcc

Bcc first reads in an ac file with assigned atom types and bond types according to AM1-BCC definitions. Then the bcc parameter file (default is BCCPARM.DAT 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: bcc -i input_file_name
           -o output_file_name
           -p bcc_parm_file_name (optional)
```

Example:

```
bcc -i comp1.dat -o comp1_bcc.ac
```

This command reads in *comp1.dat* generated by bcctype and does bond charge corrections to get AM1-BCC charges. *comp1_bcc.ac* is an ac file with the final AM1-BCC charges.

4.3.3. bcctype

Bcctype is a program to assign the atom types and bond types according to 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 have chance 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 problem bond, a "!!!" is appended at the end of the line. In intial tests, the current version works for most of the organic molecules (>95% overall and >90% for charged molecules).

```
Usage: bccprep -i  input file name
               -o  output file name
               -f  file format (ac or mol2)
               -p  atom type definition file, optional
               -j  prediction index
                   at:   bcc atom type only
                   bt:   bcc bond type only
                   both: bcc atom type and bond type
```

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
   bcctype -i $mol_dir.ac -f ac -o $mol_dir.dat
end
exit(0)
```

The above script finds all the files with extension of "ac", then calculates the Mulliken charges using *antechamber*, then predicts the atom and bond types with bcctype. Finally, you need to run bcc to get the AM1-BCC charges.

4.3.4. prepgen

Prepgen generates the prep input file from an ac file. Both the internal and Cartesian coordinate prep input files are supported. It is recommanded to use internal coordinates since the Cartesian may not always work. In default, the program generates a mainchain itself. However, you may also specify the mainchain atom in file 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 reside 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(ac)
               -o  output_file
               -f  format (car or int, default is int)
               -m  mainchain_file
               -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) output file and extract the ESP information. An esp file for 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 write out sustiva.esp, which can be used by resp program. Note that this program replaces shell scripts formerly found on the Amber web site that do equivalent tasks.

4.3.6. respgen

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

```
Usage respgen -i inputfile (ac)
              -o outputfile
              -f 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.respin1 -o sustiva.respout1 -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. They 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 related to generating molecular mechanics models for molecules. These are listed in alphabetical order.

4.4.1. crdgrow

Crdgrow can read an incomplete pdb file (at least three atoms in this file) and a prep input file, and then generate 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 you wanted, crdgrow will generate the coordinates of the missing atoms.

```
Usage: crdgrow -i input file name (pdb)
               -o output file name (pdb)
               -p prepin file name
```

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. delphigen

Delphigen can read in an ac file and generate the charge and radius file for delphi calculations.

```
Usage: delphigen -i  input file name (ac)
                 -c  charge file name
                 -r  radius file name
                 -m  modified pdb file name (optional)
                 -p  radius parameter file name (optional)
```

Example:

```
delphigen -i sustiva_resp_at.ac -r sustiva.radius -c sustiva.crg
     -m sustiva.mpdb
```

This command reads in sustiva_resp_at.ac and generate the radius file sustiva.radius (the radius definition file - RADIUS.DAT is in $ACROOT/dat) and the charge file sustiva.crg. A file in mpdb is also generated.

4.4.3. parmjoin

Parmjoin combines a force field file and an additional force field file (frcmod file)

```
Usage: parmjoin -p   parm file - input
                -m   frcmod file
                -o   parm file - output
```

Example:

```
parmjoin -p FFPARM.DAT -m frcmod -o newparm.dat
```

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:

$$
\begin{aligned}
U(\mathbf{R}) = & \sum_{bonds} K_r\,(r - r_{eq})^2 && \textit{bond} \\
& + \sum_{angles} K_\theta\,(\theta - \theta_{eq})^2 && \textit{angle} \\
& + \sum_{dihedrals} \frac{V_n}{2}\,(1 + \cos[n\phi - \gamma]) && \textit{dihedral} \\
& + \sum_{i<j}^{atoms} \frac{A_{ij}}{R_{ij}^{12}} - \frac{B_{ij}}{R_{ij}{}^{6}} && \textit{van der Waals} \\
& + \sum_{i<j}^{atoms} \frac{q_i q_j}{\varepsilon R_{ij}} && \textit{electrostatic}
\end{aligned}
$$

"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 atomic polarizability. 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 parallelopiped 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, either using "vacuum" potentials or with a generalized Born/ surface area model for aqueous solvation. The latter is a promising development, but users should remember that much less is known about the quality of the results for this option than for the more conventional explicit solvent simulations.

(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 mixing of the "unperturbed" and "perturbed" Hamiltonian, can now be carried out. For many types of free energy simulations, TI is the method of choice; users may still wish to use *gibbs* to access other types of calculations, such as free energy perturbation.

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.

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

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

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 [34-37].

Generalized Born. When *igb*=1, we use the "pairwise" generalized Born model introduced by Hawkins, Cramer and Truhlar [38,39], which is based on earlier ideas by Still and others [40-43]. Radii are the Bondi radii [44], optionally with slight modifications of the different types of hydrogen atoms; [28] the overlap paramters 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 [45]. The implementation is by David Case, who thanks Charlie Brooks for inspiration.

When *igb*=2, modifications outlined by Onufriev, Bashford and Case are used [46]; options *igb*=3 and *igb*=4 implement the GB models proposed by Jayaram, Sprous and Beveridge [30].

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 [47].

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 rad
```

`-O` Overwrite output files if they exist.

---

file	unit	in/out	purpose
mdin	5	input	control data for the min/md run
mdout	6	output	user readable state info and diagnostics -o stdout will send output to stdout (to the terminal) instead of to a file.
mdinfo	7	output	latest mdout-format energy info
prmtop	8	input	molecular topology, force field, periodic box type, atom and residue names
inpcrd	9	input	initial coordinates and (optionally) velocities and periodic box size
refc	10	input	(optional) reference coords for position restraints; also used for targetted MD
mdcrd	12	output	coordinate sets saved over trajectory
mdvel	13	output	velocity sets saved over trajectory
mden	15	output	extensive energy data over trajectory
restrt	16	output	final coordinates, velocity, and box dimensions if any - for restarting run
inpdip	19	input	polarizable dipole file, when indmeth=3
rstdip	20	output	polarizable dipole file, when indmeth=3

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)
 &end
Group input for restrained atoms
 1.0                             (force constant for restraint)
RES 1 58                         (all atoms in residues 1-58)
END
END
```

2. "Plain" molecular dynamics run

```
 molecular dynamics run
 &cntrl
    imin=0, irest=1, ntx=7,                  (restart MD)
    ntt=1, temp0=300.0, tautp=0.2,           (temperature control)
    ntp=1, taup=0.2,                         (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)
 &end
```

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) `&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', &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	Multiple `&rst` namelists; these are read from the DISANG *filename* given in section THREE. One `&rst` definition is given per restraint.
FIVE	NOESY volume restraints	Read only if NMROPT= 2 and a *NOEEXP=filename* was given in 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* was given 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 NTR=1, IDECOMP=1, ITGTMD=1 or IBELLY=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 " `&end` " 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.

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.
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 the GROUP input which follows.	
	= 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. 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
= 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 the solvent dielectric constant 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 both to truncate nonbonded pairs (on an atom-by-atom basis) and to ignore distant pairs in computing the effective Born radii; here a larger value than the default is generally required.

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.

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; users should understand that all (current) GB models have limitations, and should especially proceed with caution in using values of *igb* > 1. Generalized Born simulations can only be run for non-periodic systems, where *ntb*=0.

IGB

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

= 1 The "standard" pairwise generalized Born model is used, with parameters described by Tsui and Case [28]. 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 a on-going 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 [29].

= 2 Use a modified GB model under development by A. Onufriev, D. Bashford and D.A. Case; the main idea has been published [46], but the actual implementation here is an elaboration of this initial idea. 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 2.9. Full details are in a manuscript in preparation. Users should be aware that this model is still under development, and these parameters may change; all but the most adventurous should probably wait until the work is published, in order to see its strengths and weaknesses compared to the more conventional *igb* =1 option. With this option, you should set use the command "set default PBradii bondi" in setting up the *prmtop* file.

= 3 Uses the parameterization derived by Jayaram, Sprous and Beveridge called "GB" [30]. You should use the command "set default PBradii gbjsb" in LEaP, in order to get the correct radii and screening parameters. Setting *igb* =3 also resets the offset to zero, as required by this model.

= 4 Uses the "modified" parameterization derived by Jayaram, Sprous and Beveridge called "MGB" [30]. You should use the command "set default PBradii mgbjsb" in LEaP, in order to get the correct radii and screening parameters. We have primarily used this option for "snapshot" analysis, with *mm_pbsa*; use of this option for molecular dynamics simulations should be undertaken carefully, and may not

		give good results. Setting *igb* =4 also resets the offset to zero, as required by this 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. The theory for this is presented in J. Srinivasan, M.W. Trevathan, P. Beroza and D.A. Case. Application of a pairwise generalized Born model to proteins and nucleic acids: Inclusion of salt effects. *Theor. Chem. Accts.* **101,** 426-434 (1999). 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, however.	
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 [47].	
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 [48].	

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 in Group format at the end of all other input from file "mdin". Group input is described in the Appendix. This option is not available when *igb>0*.
NTR	Flag for restraining specified atoms in Cartesian space using a harmonic potential. Note: the restrained atoms are read in GROUP format (from the *mdin* file) after all other input (except for the belly groups). The force constant is provided in the group input, and multiple groups can be defined, each with a different force constant. 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

5.6.7. 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). At each step, sander performs a best-fit of the reference structure to the simulation structure and calculates the RMSD for the selected atoms. The energy term has the form:

```
E = 0.5 * TGTMDFRC * NATC * (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 group input (NATC). The RMSD is the root mean square deviation and is mass weighted. The atoms to be used for the RMSD calculation must be provided in group input in the same style as is used for Cartesian restraints (NTR=1), with the exception that the force constant is defined using the *tgtmdfrc* variable (see below), and is not specified in the group input. 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.

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. The atoms selected for the best-fit and RMSD calculation are the same−one cannot fit to one group and penalize the RMSD for a different group of atoms.

5.6.8. Energy minimization.

MAXCYC	Maximum number of cycles of minimization. Default 1.
NCYC	After NCYC cycles the method of minimization would be switched from steepest descent to conjugate gradient method. Default 10.
NTMIN	Flag for the method of minimization. = 0 Full conjugate gradient minimization. The first 10 cycles are steepest descent at the start of the run and after every nonbonded pairlist update. = 1 For NCYC cycles the steepest descent method is used then conjugate gradient is switched on (default). = 2 Only steepest descent method is used.
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.
DXM	The maximum step length allowed. Default 0.5.
DRMS	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.

5.6.9. Molecular dynamics.

NSTLIM	Number of MD-steps to be performed. Default 1.
NSCM	Flag for the removal of translational and rotational center-of-mass 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 travelled 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.

5.6.10. Temperature regulation.

TEMP0 Reference temperature at which the system is to be kept, through a "weak-coupling" scheme [49]. Note that for temperatures above 300K, the step size should be reduced since increased distance travelled 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 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.

HEAT If ABS(HEAT) .GE. 1.0E-06, all the velocities are multiplied by HEAT. This only affects the initial velocities assigned at TEMPI. Default 0.0.

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); the other options do not correspond to common ensembles. Some aspects of the "weak-coupling ensemble" (NTT=1) have been examined, and roughly interpolate between the microcanonical and canonical ensembles [50].

= 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 [49]. A single scaling factor is used for all atoms.

	= 4	Randomly choose velocities when the temperature changes by more than DTEMP from the target temperature TEMP0.
DTEMP		For NTT = 4, if the difference between the system temperature and TEMP0 is more than DTEMP, the velocities will be reassigned at TEMP0. Default 5.0. This is not compatible with *temp0les* > 0.
VRAND		If *vrand>0*, the velocities will be randomized to temperature TEMP0 every *vrand* steps. If *vrand<0*, the velocities will be randomized once, at step −*vrand*. Default value of 0 does neither of these.
TAUTP		Time constant for heat bath coupling for the system. Default 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, therefore 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.
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.

5.6.11. 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, *press0*. 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, analgous to temperature coupling [49]. 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 90^o).

	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 of 0.2 could be used for equilibration to more quickly get the desired density.
PLEVEL	Sets the parallelization level for constant pressure simulations. A value of zero is used for minimization, which means that only the force() routine is parallelized. The default value of 1 is the most correct for dynamics, but requires that the velocities be distributed to all processors on every step. A value of 2 avoids distribution of velocities, at the cost of a small error in computing the pressure--basically, the fact the a single molecule could span several processors is ignored in computing the virial. For typical problems with a few solute (macro)molecules and many water molecules, this error is negligible. This value is only relevent when *ntp* > 0 and -DMPI is set in the MACHINE file.

5.6.12. SHAKE bond length constraints.

NTC — Flag for SHAKE to perform bond length constraints [51]. (See also NTF in the **Potential function** section.) 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 [52]. Since SHAKE is an algorithm based on dynamics, the minimizer is not aware of what it is doing; for this reason, minimizations generally should be carried out without SHAKE.

= 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 [52].

= 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 '`.

5.6.13. 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.

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

= 1 Cap will be activated except that the cap atom pointer will be modified

= 2 Cap will be inactivated

MATCAP The cap atom pointer. This is the number of the last non-cap atom. If IVCAP = 1 then the pointer in the *prmtop* file will be overwritten by this number. Default 0.

FCAP The force constant for the cap restraint potential.

5.6.14. 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.

5.6.15. 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 standard 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, an *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 accuaracy 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
OPT_INFL	When this is set to 1 (default) sander does a least squares optimization of B-spline prefactor. This gives better energies with similar force errors with *frc_int=0* when compared with *opt_infl=0*. If *opt_infl* and *frc_int* are both set to 1, the resulting force errors are equivalent to the optimized P3M method published by Pollock and Glosli, although the present method is a bit simpler [see Darden *et al.* J. Chim. Phys. 94, 1346 (1997)]. No computational overhead is involved, so opt_inlf = 1 is recommended unless agreement with previous sander versions is sought
EW_COEFF	Ewald coefficient, in Å^{-1}. Default is determined by *dsum_tol* and *cutoff*. If it is explicitly entered here, then this 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), *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. Used 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.
FRC_INT	The smooth PME model computes the forces by differentiating the energy expression. This is the behavior that is used when *frc_int* is 0, the default. When *frc_int = 1*, force interpolation is used; this conserves momentum, but requires two additional FFT's at each step, and hence is slower.
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.

5.6.16. 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

```
C1------C2------C3------C4
|       |       |       |
|       |       |       |
Ep1     Ep2     Ep3     Ep4
```

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.

5.6.17. Polarizable potentials

The following parameters are relevant for *polarizable potentials*, that is, when *ipol* is set to 1 in the `&cntrl` namelist. Currently polarizability can be run with *ew_type*=1 or 2 and with *frc_int*=0 or 1. These variables are set in the `&ewald` namelist.

INDMETH	If indmeth is 0 1 or 2, 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 (1-st, 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).

5.6.18. 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:

$$\Delta A \equiv A(\lambda = 1) - A(\lambda = 0) = \int_0^1 \langle \partial V / \partial \lambda \rangle_\lambda d\lambda \qquad (1)$$

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:

$$\Delta A \approx \langle \partial V / \partial \lambda \rangle_{\frac{1}{2}} \qquad (2)$$

This might 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:

$$\Delta A \approx \sum_{i=1}^{n} w_i \langle \partial V / \partial \lambda \rangle_{\lambda_i} \qquad (3)$$

Some weights and quadrature points are given in the accompanying table; other formulas are possible [53], 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:

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

$$V(\lambda) \ = \ (1-\lambda)V_0 + \lambda V_1 \tag{4}$$

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 [54]. 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 $\lambda \rightarrow 1$ [54], so that setting *klambda* = 4 with disappearing atoms should work well with Gaussian quadrature methods.

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; you would need to use *gibbs* for situations like this.

(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 depronated 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.]

(4) Users should also be aware that this option has not been extensively tested. It might be wise to run a short thermodynamic perturbation calculation in gibbs and to compare the results to sander. In spite of the performance hit cited above, putting this facility in Amber allows us to use the generalized Born model, and to take advantage of improvements in PME and parallelization that are not in gibbs.

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	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)*
IMULT	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.
SOFTR	Varies the soft-repulsion non-bond force constant. Has no effect if ISFTRP.LE.0.
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:

ISTEP1 ≤ ABS(delta_residue) ≤ 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:

VALUE1 ≤ distance ≤ 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 3 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.

(For these cards, ISTEP1 and IMULT are ignored).

#	If formatted input is being read (&formwt was specified), any line which starts with a pound symbol (#) is considered a comment line, and will be skipped.
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. If either IDMPAV has not been set, or DUMPAVE is not specified, this file will not be written.

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.

Any empty DISANG file is sometimes useful, if you wish to include weight changes but have no internal angle constraints. Note that, unlike earlier versions of Amber, the `&rst` namelists should 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 your *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 statment, 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 *makeVOL_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 [55]. 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 [56]. As with *omega*, *taumet* can be different for different sub-molecules, 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 *makeVOL_RST* program which provides a facility for turning human-readable names into atom numbers, and also assists in dividing a large molecule into submolecules.

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 [57]; more recent empirical parametrizations seem to give improved results [58,59].
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	Resiude 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 suscepibility 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 traslational 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,
 &end
```

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=5000,
 ncyc=500,ntmin=1,dx0=0.0001,dxm=1.0,dele=1.0e-07,
 drms=.1,cut=10.,idiel=0, scee=2.0,
 nmropt=2,pencut=0.1, ipnlty=2,
 &end
 &wt type='REST', istep1=0,istep2=1,value1=0.,
     value2=1.0,  &end
 &wt type='END'  &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; hence, *iscale* should be set to 5 in the `&cntrl` namelist. 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 [60]. The factor of 10^5 is just to make the values commensurate with atomic coordinates, since both the coordiantes 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.

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.
DOBS	Value of the observed dipolar splitting, in Hz.
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 rougly commensurate with coordinates in Angstroms, the alignment tensor values must be multiplied by 10^5.
DWT	The relative weight of each observed value. Default is 1.0. The penalty function is thus: $E^i_{align} = D^i_{wt}(D^i_{calc} - D^i_{obs})^2$ where D_{wt} may vary from one observed value to the next.
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 [61].
DCUT	Controls printing of calculated and observed dipolar couplings. Only values where abs(dobs - 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. 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 addtional "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 [62]. 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
```

    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, are are not discussed here. The principal sander-specific tool is *sviol*, which prepares tables and statisitics of energies, restraint violations, and the like.

5.13.1. 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 [62-68]. 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.

5.13.2. 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

other options:
   -help      (gives you this explanation, overrides other parameters)
   -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)
```

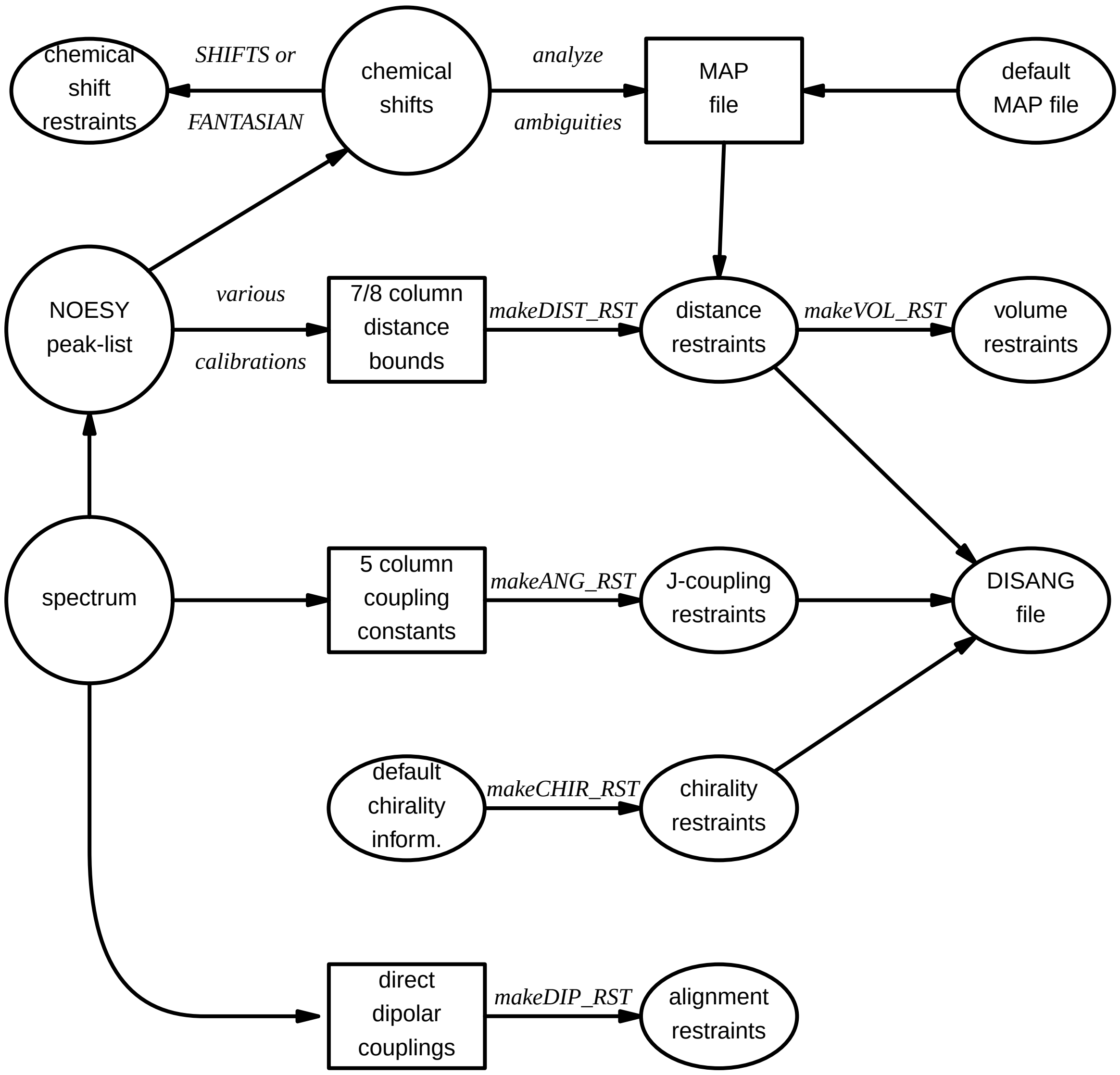


Fig. 1. Notation: *circles* represent logical information, whose format might differ from one project to the next; *solid rectangles* are in a specific format (largely compatible with DIANA and other programs), and are intended to be read and edited by the user; *ellipses* are specific to Sander, and 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*.

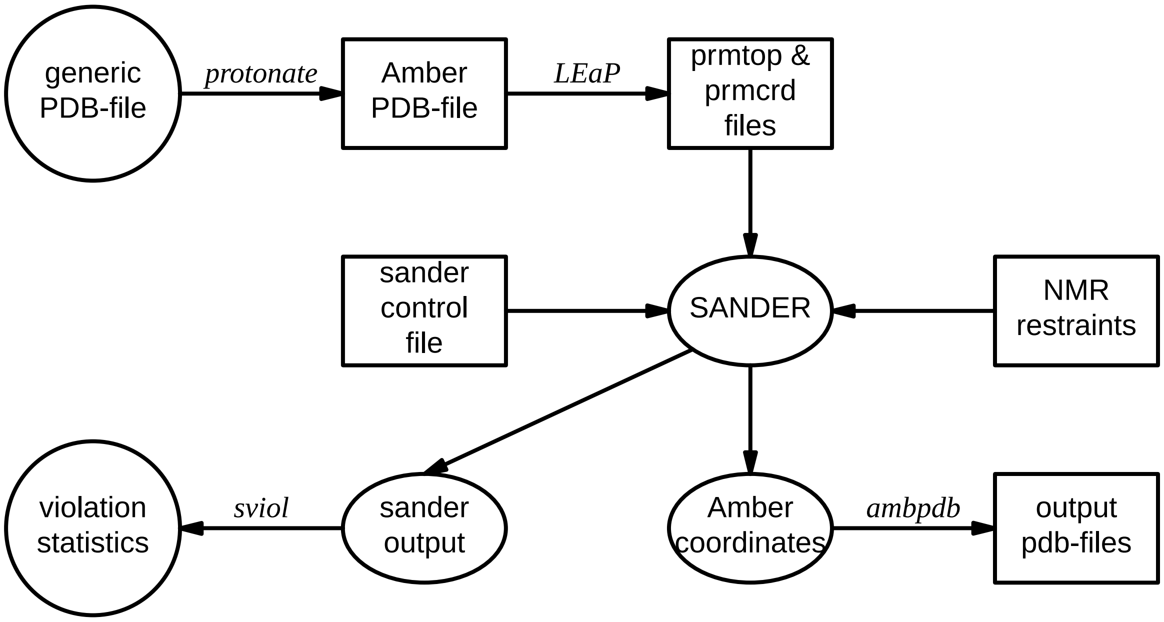


Fig. 2

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. 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.

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-occuring 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 collective 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 users 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 ambigous 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.

5.13.3. 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 own 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 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 psuedorotation 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 four angles NU0-NU4. Then the definition for NU0-NU4 should also apper 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 a appropriate Karplus relation. See the discussion of the variable `RJCOEF`, above.

5.13.4. 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.

5.13.5. NOESY volume restraints: makeVOL_RST

Refinement directly against measured NOESY volume restraints is also possible in *sander*. The makeVOL_RST is used to prepare detailed input for this.

```
Usage:  makeVOL_RST [-c cutoff] [-p <pdb-file>] [-v <volume-file>]

Defaults: cutoff is 5. Ang., <pdb-file> is INPDB, <volume-file> is PEAKS
```

The input files are as follows:

(1) the volume-file has one line per peak (free format); the first two columns of each line give the Amber atom numbers for the protons involved in the peaks. (Usual convention: use the last atom number for methyl or aromatic delta and epsilon protons.) The remaining columns give the intensities at the various mixing times.
The *first line* of the volume-file should have the following (free format): nmix, (emix(i),i=1,nmix), where nmix is the number of mixing times and emix gives the mixing times (in seconds).
This file would generally be created by running makeDIST_RST using the "-svf" (sander volume format) option. See the instructions for makeDIST_RST for more information.

(2) the pdb-file is a Brookhaven-format file (made by "ambpdb"), giving atom names and coordinates. Atom numbers in this file must correspond to those used in the volume-file, and in the Amber runs themselves.

On output, *stdout* contains a file that, with some minimal hand editing, should serve as input to SANDER. More explicitly, it creates groups and &noeexp namelists that inlcude many of the variables you will need. Any missing variables should be added by hand. [I do this since it seems easier to hand-edit the output file than to arrange for a lot of pass-throughs from this program into the output file.]

During refinement, measured NOESY volumes are included in penalty functions that depend upon $(I - I_0)$ where I_0 is the experimentally measured value, and I is the value corresponding the current conformation; the functional form of the penalty depends upon the *ipnlty* variable. Careful experimentation will undoubtedly be required for each data set to define a reasonable penalty function. Simply weighting each observed peak equally (with the default values of *awt* and *arange*) is almost certainly a bad idea, since this effectively gives too much influence to the strong peaks at the expense of longer-range information.

5.13.6. 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.

5.13.7. 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.)

5.13.8. Time-averaged restraints.

The model of the previous sections involves the "simgle-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

$$\bar{r} = (1/C) \left\{ \int_0^t e^{(t'-t)/\tau} r(t')^{-i} dt' \right\}^{-1/i} \tag{1}$$

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

$$< \phi > \; = \; \tan^{-1}(< \sin(\phi) > / < \cos(\phi) >) \tag{2}$$

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 [69,70]. torda huber 1993 .lf 8875 _manual.me

5.13.9. 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 satisifies 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.

5.13.10. 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)
 &end
 &ewald
    eedmeth=5,                    (use r dielectric)
 &end
#
# 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.,    &end
 &wt type='TEMP0', istep1=1001, istep2=3000, value1=1200.,
            value2=1200.0,     &end
 &wt type='TEMP0', istep1=3001, istep2=15000, value1=0.,
            value2=0.0,     &end
#
# 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,     &end
 &wt type='TAUTP', istep1=3001,istep2=11000,value1=4.0,
            value2=2.0,     &end
 &wt type='TAUTP', istep1=11001,istep2=13000,value1=1.0,
            value2=1.0,     &end
 &wt type='TAUTP', istep1=13001,istep2=14000,value1=0.5,
            value2=0.5,    &end
 &wt type='TAUTP', istep1=14001,istep2=15000,value1=0.05,
            value2=0.05,    &end
```

(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,  &end
 &wt type='REST', istep1=3001,istep2=15000,value1=1.0,
            value2=1.0,  &end

 &wt type='END'  &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, &end
#
# 3 ALA HA      4 GLU HN        4.00
#
 &rst iat=  42,  50, r3= 4.00, r4= 4.50,  &end
#
# 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,   &end
#
# .......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, &end

 &rst iat= 233, 235, 237, 249, r1=-190.0,
      r2=-160.0, r3= -80.0, r4= -50.0, &end

#  .......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.,  &end
#
#  chirality for residue  1  atoms:   CB SD HG2 HG3
 &rst iat= 5 , 11 , 9 , 10 , &end
#
#  chirality for residue  1  atoms:   N C HA CB
 &rst iat= 1 , 18 , 4 , 5 , &end
#
#  chirality for residue  2  atoms:   CA CG2 CG1 HB
 &rst iat= 22 , 26 , 30 , 25 , &end#
#
  ......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.,  &end
#
#  trans-omega constraint for residue  3
 &rst iat= 41 , 39 , 37 , 22 , &end
#
#  trans-omega constraint for residue  4
 &rst iat= 51 , 49 , 47 , 41 , &end
#
#  ......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 makeVOL_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, made by makeVOL_RST :

```
 &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,
 &end
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, fj=0.1, dfac_al=37*-0.46623, 54*-0.21341,
  s11=3.883, s22=53.922, s12=33.855, s13=-4.508, s23=-0.559,
  id(1)=188,   jd(1)=189,  dobs(1)=  6.24,
  id(2)=208,   jd(2)=209,  dobs(2)= -10.39,
  id(3)=243,   jd(3)=244,  dobs(3)= -8.12,

  ....
  id(91)=1393,   jd(91)=1394,  dobs(91)= -19.64,
&end
```

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*',
 &end
 &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*',
 &end
```

5.14. 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 10ˆ-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 10ˆ-15 up to 10ˆ-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 10ˆ-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,
&end
```

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,
&end
```

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,
&end
```

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,
&end
&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,
&end
&ewald
  nfft1=60,nfft2=60,nfft3=60,order=6, ew_coeff=0.35,
&end
```

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,
&end
&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,
&end
&ewald
 ew_coeff=0.35,
&end
```

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 10ˆ-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 5x10ˆ-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. 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`.

Information on using `addles` for sander.LES is given here. The gibbs version is not yet ready.

6.1. Background.

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 [73]. 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 [74] 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 [74]. 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 [75-78]. 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.

6.2. 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.

6.3. 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
~ 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=(solv2OO.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 veocities, 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 [79].

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.

6.4. 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
```

6.5. 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.

6.6. Case studies: Examples of application of LES

6.6.1. 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 [76].

6.6.2. 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.

6.6.3. 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 7 CD.

6.7. 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.) [80,81] 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 *morita.chem.sunysb.edu/˜carlos/moil-view.htm*l 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) LES does not currently work with the generalized Born solvation model.

7. Gibbs

```
Usage: gibbs [gibfile] [-O] -i gibin -o gibout
            -p prmtop -c inpcrd -r restrt
            -ref refc -x mdcrd -v mdvel -e mden
            -inf mdinfo -ms micstat
            -cm constmat -cs cnstscrt -a patnrg
```

`-O`	Overwrite output files.

7.1. Introduction

This is a guide to *gibbs*, the AMBER module concerned with free energy calculations. This module of the AMBER suite of programs calculates the free energy difference, ΔG, between two states "0" and "1":

$$\Delta G = G_1 - G_0 \tag{1}$$

In LEaP, *State 1* is the default state and State 0 is defined by setting the perturbed atom parameters in the "Edit selected atoms" table in the xleap Unit Editor and using the saveAmberParmPert command to make the topology and coordinate files. (Note that this convention is "backwards" compared to that in *sander*.)

The free energy difference is calculated in a series of incremental steps which connect physical states 1 and 0 through a series of not-necessarily-physical intermediates. The character of the system at each of these intermediate steps is related to a parameter λ.

7.2. Free Energy Techniques Available in GIBBS

There are several techniques available in GIBBS/AMBER 4.0 for evaluating the free energy difference between two states, all based on various statistical mechanical relationships. These include:

(1) Free Energy Perturbation (FEP) Window Growth: The free energy is calculated at discrete and uniformly spaced intervals of λ using the formulae:

$$G_{\lambda(i+1)} - G_{\lambda(i)} \;=\; -\,RT\,\ln < \exp -[(V_{\lambda(i+1)} - V_{\lambda(i)})/RT\,] >_{\lambda(i)} \tag{2}$$

$$\Delta G \;=\; G_1 - G_0 \;=\; \sum_i G_{\lambda(i+1)} - G_{\lambda(i)} \tag{3}$$

where G_0 and G_1 are the free energies of states 0 and 1, respectively, $V_{\lambda(i)}$ is the potential energy function representative of state $\lambda(i)$, and $<>_{\lambda(i)}$ means use the ensemble average of the enclosed quantity, representative of state $\lambda(i)$. The ensemble is evaluated from an MD trajectory run with $V = V_{\lambda(i)}$ The user specifies the numbers of equilibration (NSTPE or NSTMEQ) and data collection (NSTPA or NSTMUL) steps for each $\lambda(i) \rightarrow \lambda(i+1)$ "window".

(2) Slow growth − the same as window growth, except lambda changes by a small amount at every step. Lambda changes slowly enough that it is assumed the system remains in equilibrium at every step (i.e. NSTPE=0, NSTPA=1). Thus the ensemble average in Equation (2) is replaced by its instantaneous value at each step.

(3) Thermodynamic integration − instead of Equations (2) and (3), we use

$$G_1 - G_0 = \int_0^1 < \partial V/\partial\lambda >_\lambda \, d\lambda \qquad (4)$$

to calculate the free energy difference. In practice, the integral is approximated by a summation over discrete intervals in λ.

(4) Dynamically Modified Windows − the equations of FEP (2 and 3) are used as described for method 1 above. But instead of using pre-chosen uniformly-spaced intervals of λ, the width ($\delta\lambda = \lambda(i+1) - \lambda(i)$) of each window is determined during the run, based on the recent value of the slope, $\partial G/\partial\lambda$, of the accumulated free energy versus λ curve. This allows the simulation to be run more "slowly" when the free energy is changing very quickly, and more "quickly" when it is not.

(5) Dynamically Modified Thermodynamic Integration − Uses the same λ adjustment algorithm as for FEP (method 4), but the intervals in λ correspond to the points at which the integrand in Equation (4) is evaluated to approximate the integral.

(6) Potential of Mean Force (PMF) Calculations − the user can elect to constrain any chosen set of internals (distances, angles, torsions) to a chosen lambda-dependent pathway. By selecting the appropriate option (NCORC=1), the contribution to the free energy from such constraints will be calculated. This constitutes a PMF calculation. PMF calculations can be carried out as part of either a FEP Window Growth or Dynamically Modified Windows run (1 and 3 above).

7.3. Understanding the Output

(a) Window growth, slow growth, dynamically modified windows: At specified intervals during the simulation, the energies calculated up to that point will be reported in the format:

```
Current Lambda =   0.850000
Last F.E. update: Lambda =   0.800000  Step =    4000  Method = F.E.P.
Accumulated "forward" quantities (Nonbond change)
   Lam+d_lam = 0.850000    F_energy  =   +0.64300
   ELEC =      0.000    NONB =     +0.643    14NB =      0.000
   14EL =      0.000    BADH =      0.000
Accumulated "reverse" quantities (Nonbond change)
   Lam-d_lam = 0.750000    F_energy  =   -0.62130
   ELEC =      0.000    NONB =     -0.621    14NB =      0.000
   14EL =      0.000    BADH =      0.000
```

When the free energies reported were last updated, the values of lambda and step number were as given on the second line. Note that the *current* values of λ and Step may be different, if the free energies have not yet been updated to reflect the ensemble now being generated. Also reported on the second line is the method being used to calculate free energy differences: F.E.P. is Free Energy Perturbation (standard or Dynamically Modified Windows); T.I. is Thermodynamic Integration (standard or Dynamically Modified Windows); Slow Growth is self explanatory.

Both "forward" and "reverse" accumulated free energies are reported. By default, GIBBS carries out "double-wide sampling", which means that at every value of λ we calculate the free energies both for going $\lambda \rightarrow \lambda + \delta\lambda$ and for going $\lambda \rightarrow \lambda - \delta\lambda$. The values "Lam+d_lam" and "Lam-d_lam" which are reported were the values at the last free energy update. If there were no sampling errors in our calculations, the independent sums of the "forward" and "reverse" values over the entire simulation would be the same, except for sign. Their actual difference gives us a *lower bound* on the error. By convention, the "forward" energy always corresponds to the energy for the process represented by λ increasing $0 \rightarrow 1$. Similarly, the "reverse" energy corresponds to the process represented by λ decreasing $1 \rightarrow 0$. *This is true regardless of the direction in which the actual simulation was run.*.

Along with the total accumulated free energies in the "forward" and "reverse" directions, a component breakdown of the energies is given. Components listed include: ELEC (electrostatics, except 1-4's); NONB (non-bonds, except 1-4's); 14NB (1-4 nonbonds); 14EL (1-4 electrostatics) and BADH (bonds, valence angles and torsion angles). *Note that for Windows and Dynamically Modified Windows, these components are only estimates*. For slow growth and thermodynamic integration, they are exact.

If PMF calculations are performed, a sixth component will be listed, CORC. The procedure used to perform a PMF makes it difficult to separate contributions due to the constraints themselves from those due to non-bonded/electrostatic interactions. For this reason, in these cases CORC will reflect the sum total of all three types of contributions and the individual non-bonded/electrostatic contributions will be reported as 0's.

(b) Thermodynamic integration: The output is similar to that described above, except that, because of the integral which must be evaluated in thermodynamic integration (TI) (Equation 4), double-wide sampling is not possible. Thus, only a "forward" set of energies is reported. Again, by convention, these value have the sign appropriate for the $0 \rightarrow 1$ conversion, regardless of the direction in which the simulation was actually run.

If the calculation of individual entropy/enthalpy contributions is requested, these will also be included in the output, following the same forward/reverse conventions as above.

7.4. Defining States and Obtaining Appropriate Starting Coordinates

The default state from which to start the perturbation is usually λ=1. However, you can equilibrate at either λ=1 or λ=0 (or any arbitrary value of λ) as follows:

Set ISLDYN (line 14) to +-2 or +-3;
Set NRUN (line 5) to 1;
Set NSTLIM (line 8) to the number of steps of equilibration desired;
Set ALMDA (line 14) to the value of λ at which equilibration is to take place;
And set NSTMEQ (line 14) to any value greater than NSTLIM.

The program is capable of handling periodic boundary conditions with the solute in a solvent bath either with constant volume or constant pressure. Additionally, it is possible to decouple the free energy into electrostatic and van der Waals contributions, if desired.

7.5. Suggested introductory references

The papers listed here emphasize the theory and experience with the AMBER programs; general reviews are available that cover many other apsects of the field [54,82-84]. The basic operation and capabilities of *gibbs* are described in a series of papers by Dave Pearlman, Peter Kollman, and their co-workers [1,85-87]. Because it has so many options, the *gibbs* program has been used a lot for comparative studies of free energy methods [88-91]. Some recommendations for free energy calculations, with specific reference to the options in *gibbs*, are given at the end of this chapter. Thermodynamic integration calculations can also be carried out with *sander* (see that chapter).

7.6. Assigning files

GIBBS incorporates a file assignment protocol which is easy to use, and which will work on all computers. In addition, file assignments can optionally be specified using flags on the command line.

```
gibbs [gibfile] [-O] [-i PIN] [-p PPARM] [-c PINCRD]
          [-o POUT] [-r PREST] [-inf PINFO] [-ms MICSTAT]
          [-cm CONSTMAT] [-cs CNSTSCRT] [-a PATNRG]
          [-x PCOORD] [-v PVEL] [-e PEN] [-ref PREFC]
```

where PIN, PPARM, etc. are replaced by the appropriate filenames to be assigned. The meanings of the various files are given below.

If "gibfile" is present, it must be the first option given, and this file will be read to make the file assignments. In this case, any remaining flags are ignored. Otherwise, all assignments are made using command-line flags. Any flags not specified default to the given name (e.g. if -o is not specified, output would be in file POUT).

GIBBS I/O FILE ASSIGNMENTS

file	unit	purpose
INPUT:		
PIN	5	Control data for the run (described below).
PPARM	8	Topology file (created by LEaP)
PINCRD	9	Initial positions and (optionally) velocities.
PREFC	10	Reference coordinates for optional position restraints (only if NTR = 1)
OUTPUT:		
POUT	6	Formatted results and diagnostics
PREST	16	Restart coordinates and velocities. For restarts, this file should be assigned to PINCRD.

```
PINFO      7    Short file containing a summary of current energies.
                For monitoring runs which are executing.

MICSTAT   27    A concise summary of important energy information
                for each window/interval.

CONSTMAT  28    Contains data related to the matrix of free
                energy data generated. Only used when IPER>0 for
                one or more of the constraints/restraints defined
                with INTR > 0 (see line 13).

CNSTSCRT  42    Contains data required when generating
                a matrix of free energies corresponding to two
                independent sets of constraints (IPER>0 and INTR>0;
                see line 13).

PCOORD    12    Archived coordinate sets (if NTWX > 0)

PVEL      13    Archived velocity sets (if NTWV > 0)

PEN       15    Archived energy related data (if NTWE > 0)
```

7.7. Control parameters

The title (line 1) must be the first line in PIN. All remaining standard flags are entered in the namelist *&cntrl.*

TIMLIM Time limit for the job (in seconds). Default = 999999.

IREST Flag to restart the run.

= 0	Normal start (default)
= 1	Job to be restarted. The accumulated free energies, current value of lambda, and other required quantities are read from the end of the input coordinate file (PINCRD). This file should be the PREST file written by the simulation being restarted.

IBELLY Flag for belly type dynamics.

= 0	No belly run (allow all atoms to move; default).
= 1	Belly run. The subgroups of atoms which are allowed to move are read as groups from file PIN. See the section on GROUP in the Appendices.

ICHDNA — Option to modify the charge of end hydrogens during in vacuo simulations. Without this option, molecular dynamics calculations on nucleotides will result in bonding between the 5’ and 3’ hydrogens and the corresponding phosphate groups.

= 0	no charge modification (default)
= 1	modify charge

IPOL — for inclusion of polarizabilities in the force field.

= 0	non polar calc (no polarizabilities read from "prmtop"; default).
= 1	turn on polarization calculation.

I3BOD — For 3-body terms with a polarization calc.

= 0	No 3-body terms to be defined. Default.
= 1	Read and use 3-body interaction definitions (see card 18). 3-Body terms only have an effect when polarization is turned on (IPOL=1).

NTX — Option to read the initial coordinates and velocities.

Options 1-3 are used when no set of starting velocities is available (e.g. when starting from a set of minimized coordinates). Options 4-5 are used when: 1) a starting set of velocities is available (e.g. after MD equilibration or on an MD RESTART); and 2) The coordinates/velocities were generated with MD run either without periodic boundary conditions, or with constant VOLUME periodic boundary conditions. (Box dimensions, if any, are taken from the PARM file). Options 6,7 are used when both a starting set of velocities are available and the coordinates/velocities were generated with MD run using constant PRESSURE periodic boundary conditions. Note: box dimensions only appear in coordinate files written (as PREST) after simulations using periodic boundary conditions (constant volume or constant pressure).

= 1	X is read; no velocity information read (Amber format); default
= 2	X is read; no velocity information read (unformatted)
= 4	X and V are read (unformatted)
= 5	X and V are read (Amber format)
= 6	X, V and BOX are read (unformatted)
= 7	X, V and BOX are read (formatted)

NTXO — Option to write the final coordinates and velocities.

	= 0	X, V and BOX are written to file 'PREST' (unformatted)
	= 1	X, V and BOX are written to file 'PREST' (Amber format); default.
IG	The seed for the random number generator. The MD starting velocity is dependent on the random number generator seed. The generator works most effectively when the seed is large and an odd or a prime number (e.g. 71277, the default).	
TEMPI	Initial temperature, default is 0.0. If TEMPI > 1.0e-06, the velocities are taken from a maxwellian distribution with TEMPI (K). Choosing a low initial temperature (e.g. 10K) allows the calculation to reach the equilibrium conditions with the residual forces in the system during the initial steps. TEMPI is ignored if NTX > 3.	
HEAT	If ABS(HEAT) .GE. 1.0E-06, all the velocities are multiplied by HEAT. Default is 0.0.	
NTB	Flag for periodic boundary conditions. If NTB .EQ. 0 then the boundary conditions are NOT applied. The periodic box may be rectangular or monoclinic depending on the value of BETA.	
	= 0	no periodicity is applied; default.
	= 1	constant volume
	= 2	constant pressure.
IFTRES	Flag to remove the nonbonded cutoff from the solute.	
	= 0	ALL solute - solute nonbonded interactions are calculated, and the boundary conditions are not applied to the solute. For simulations of highly charged solutes in a water bath, it can be useful to calculate ALL solute - solute nonbonded interactions in order to reduce electrostatic problems. Note that this option is intended for small solutes, and will generate many more nonbonded pairs than the normal method if the solute is large. This option is useful for DNA and counterions. Note: if counterions are added in edit, then they are considered part of the solute.
	= 1	Nonbondeds are evaluated normally; default.
		Note: IFTRES will only have an effect when periodic boundary conditions are employed (NTB > 0). When NTB=0, IFTRES=1 behavior (normal nonbond generation) always occurs.
BOXX(1..3)	Lengths of the edges of the periodic box. If IBXRD > 0, then the values specified here will be used. Otherwise, the values specified here are ignored and the values in the PARM output file (if NTX < 7) or the values in PINCRD (if NTX >= 7) will be used.	
BETA	Angle between the x- and z- axes of the box in degrees. The y- axis is assumed to be orthogonal to the other axes. (0 < BETA < 180). The information given for BOX(1..3) above applies to BETA as well. Non-orthogonal systems do not currently work correctly. Therefore, if IBXRD > 1, BETA must be set to 90.0, which is the default.	
IBXRD	If IBXRD > 0, then the values of BOX(1..3) and BETA specified here will be used. Otherwise, the values in the PPARM or PINCRD file will be used (see	

above).

NRUN Number of MD-runs of NSTLIM steps to be performed; default is 1. Since the restart coordinates are written only at the end of each run, it is sometimes desirable to break a long run into a series of shorter steps. If NRUN is set > 1, one should ensure that the number of equilibration+data_collection steps (if performing windows/TI) divides evenly into NSTLIM (line 8). The number of picoseconds of molecular dynamics is equal to the product of NRUN X NSTLIM X DT.

NTT Switch for temperature scaling. Note that several of he temperature coupling options available here are new to version 4 of GIBBS. Several of these are rather ad-hoc, and may not result in a thermodynamically relevant ensemble. (They may be useful when using MD strictly to sample conformational space). For free energy calculations, it is recommended you stick with NTT = 0 (constant energy), NTT = 1 (constant temperature) or NTT = 5 (constant temperature, separate solute/solvent temperature scaling).

< 0 Re-assign random velocities whenever the current temperature deviates by more than DTEMP from DTEMP0 (target temperature), and every ABS(NTT) steps. Velocities are assigned in a Maxwellian distribution. By default, velocities are are reset for all atoms. If NSEL > 0 (see below), NSEL atoms are selected at random each time a velocity reassignment is to take place, and only those atoms have their velocities reassigned. (Be sure to set DTEMP0 to a very large value if you wish to disable its action with this option).

Note that the procedure which assigns velocities makes the assignments as if all particles possessed three independent degrees of translational freedom. If SHAKE is used, this will not strictly be the case, and the effective temperature immediately after velocity assignment will be higher than the target temperature. As velocity contributions along the constrained directions are dissipated, the temperature will rapidly adjust towards the target.

= 0 Classical dynamics. Never rescale/reassign velocities after the start. [The total energy (kinetic + potential) is conserved; same as in older versions of GIBBS.]

= 1 Constant temperature, using the Berendsen coupling algorithm. A single scaling factor for velocities is used (same as in older versions of GIBBS). This is the default.

= 2 Constant temperature, using the Berendsen coupling algorithm. But only consider the solute temperature in determining the velocity scaling on each step. Could result in solvent atoms having very high temperature, and not generally recommended.

= 3 Constant temperature, using Berendsen algorithm. But only rescale when temperature deviates from TEMP0 by more than TEMP0. Single scaling factor.

= 4 When temperature deviates from TEMP0 by more than DTEMP, do one quick scale of the velocities to bring them back to TEMP0. Otherwise, do not scale.

= 5 Constant temperature, using the Berendsen coupling algorithm, and with separate solute/solvent velocity scaling factors. This option is recommended as a replacement for NTT=1, and can help alleviate the "cold solute/hot solvent" problem.

TEMP0 Reference temperature at which the system is to be kept if NTT not = 0. Default is 298.

DTEMP The deviation allowed in the constant temperature MD-runs (read but ignored if NTT=0,1,2 or 5). Default is 10.

TAUTP Temperature relaxation time when NTT .gt. 0. This is a damping factor which prevents abrupt changes in the system, if the temperature exceed specified deviations. Generally, values for TAUTP should be in the range of 0.1-0.4. Smaller values of TAUTP result in "tighter" coupling. Default is 0.1.

TAUTS If NTT=5, then TAUTP is the temperature relaxation time for the solute, while TAUTS is the relaxation time for the solvent. If is specified as 0.0, TAUTS is set equal to TAUTP. Generally, TAUTS should be in the range of 0.1-0.4, with smaller values resulting in "tighter" coupling. If NTT.NE.5, TAUTS is read but ignored. Default is 0.1.

ISOLVP Only used if NTT = 2 or 5 (sep. solute/solvent temp coupling)

= 0 default solvent atom pointer is used. If periodic boundary conditions are being used, this is the last solute atom. Otherwise, it will be the last atom of the system (which results in no separate solute/solvent coupling). Note that counterions are by default considered part of the _solute_.

> 0 Gives the number of the last atom to be considered part of the "solute". ISOLVP should generally be specified if NTT = 5 and NTB = 0. ISOLVP only affects temperature scaling.

NSEL Only used if NTT < 0 (random velocity reassignments)

= 0 When velocity reassignment takes place, velocities for all atoms are reassigned (default).

> 0 When velocity reassignment takes place, NSEL atoms are randomly selected, and only the velocities for those atoms are reassigned.

DTUSE The value of d_TEMP used in approximating the temperature derivatives by finite differences. DTUSE is only used when individual enthalpy/entropy values are being calculated (ISANDE = 1, line 12). DTUSE should generally be <= 1.0 (larger values often cause floating overflows/ underflows). Default is 1.0.

NTP Flag for constant pressure dynamics. This option MUST be set to 1 or 2 when the MD calculation is done with constant pressure periodic boundary conditions (NTB=2, line 4).

= 0 Classical dynamics without any Pressure Monitoring (default)

= 1 MD with isotropic position scaling

= 2 MD with anisotropic diagonal (x-,y-,z-) position scaling

NPSCAL Flag for the type of scaling in case of constant pressure run.

	= 0	Uniform coordinate Scaling (default)
	= 1	Sub molecules Center of mass Scaling
PRES0		Reference pressure at which the system is maintained (when NTP > 0) in units of bars, where 1 bar ˜ 1 atm. Default = 1.0.
COMP		Inverse compressibility of the system when NTP > 0. The unit is in 1.0E-06/bar (the default value of 44.6 is recommended).
TAUP		Pressure relaxation time when NTP .gt. 0 The recommended value is between 0.1 and 1.0 ps^{-1}. Default is 0.4.
NDFMIN		Number of degrees of freedom that will be subtracted from the total number of degrees of freedom to account for center of mass removal, belly runs, etc. (This will be a value between 0 and 6). By default (if NDFMIN.GE.0), this value will be set automatically. For nearly all simulations, you should accept the default calculated when NDFMIN = 0. If you set NDFMIN<0, then ABS(NDFMIN) additional degrees of freedom will be subtracted *in addition to* the number calculated automatically. This option is provided so that you can account for systems containing extended linear moities that reduce the true number of degrees of freedom from that which would be calculated by a simple 3N-6 determination. For example, if you used a linear triatomic molecule for your solvent, you would need to set NDFMIN = -(number of solvent molecules).
NTCM		Flag for the removal of translational and rotational motion from the initial velocities. NOTE: this flag is automatically set to 0 if belly option is used.
	= 0	The translational and rotational motion about the center of mass is not removed (default)
	= 1	The above motion is removed and NTCM is reset to 0. If velocities are being periodically reassigned according to a Boltzmann distribution (NTT < 0) and NTCM = 1, then center of mass motion will be removed after each reassignment.
NSCM		After NSCM steps the above motion will be removed again if NTB .EQ. 0. This flag should be set to -1 if the belly option is used. This results in NSCM .EQ. 90 000 000 steps. Default is -1.
ISVAT		Residue-based periodic imaging flag ISVAT is ignored when periodic boundary conditions are not used.
	= 1	Residue-based periodic boundary conditions are used (default). For each residue, imaging is determined based on the position of the atom in the residue which is closest to the residue's initial center of mass. Both solute and solvent atoms are imaged on a residue basis. Each atom of any solute or solvent residue "sees" the same image of any interacting residue.
	= 2	Same as 1, except that for each atom of the _solute_, different whole-residue images on interacting residues may be used. Can be useful when a solute residue is fairly long in one or more dimensions. The code required to implement ISVAT=2 does not vectorize, and may result in a substantial hit to performance on vector machines. For this reason, ISVAT=1 should be used except where ISVAT=2 is clearly

required.

= 3 No residue-based periodic imaging. Separate imaging is done for each atom-atom pair. This is the way imaging was done in versions ≤ 3 of GIBBS (and MD). In typical operation, you would NOT want to use this option. Setting ISVAT<3 allows a cutoff of as large as ~ 1/2 the smallest box dimension to be used. When ISVAT=3 with periodic boundary conditions, a much smaller cutoff/box ratio must be used.

NSTLIM

> 0 Number of MD-steps per run to be performed. NRUN such runs will be carried out.

= -1 Continue simulation until done, or until TIMLIN is exceeded. This option is often used with dynamically modified procedures (since we don't know at the outset how many total steps will be required). This is the default.

INIT Flag for different starting procedures. If option NTX is less than 5, INIT should be equal to 3. If option NTX is greater than or equal to 5, this option should be equal to 4.

= 3 V(T-DT/2) is obtained by calculating force(T); default.

= 4 Input V(T-DT/2) is used for the starting velocity

T The time at the start (psec). Only for your own use. Not important for the simulation. Default is 0.0.

DT The time step (psec); default is 0.001. (Note that in the special case where window growth is requested by using the unrecommended flag combination (IFTIME = 0 and ISLDYN = 0; line 14), DT is replaced by the value of DTA on line 15).

VLIMIT Limiting velocity; default is 0.0. If .ne. 0.0, then any component of the velocity that is greater than abs(VLIMIT) will be reduced to VLIMIT (preserving the sign), and a warning message will be printed. This can be used to avoid occasional instabilities in molecular dynamics runs. VLIMIT should generally be set (if at all) to a value like 20., which is well above the most probable velocity in a Maxwell-Boltzmann distribution at room temperature. Note that although it is anticipated that use of a liberal (large) value of vlimit should not adversely affect the statistics accumulated during a free energy simulation, this has not yet been definitively demonstrated.

IVEMAX Maximum times VLIMIT may be exceeded. If IVEMAX >0, then IVEMAX specifies the number of times the limiting velocity VLIMIT can be exceeded in a simulation. If VLIMIT is exceeded >= IVEMAX times, the simulation will stop. If IVEMAX =0, there is no limit on the number of times VLIMIT can be exceeded. Default is 0.

NTC Flag for SHAKE to perform bond length constraints. Constraining the bond lengths removes the highest frequency motions from the system and usually allows somewhat larger timesteps to be used.

= 1 SHAKE is not performed (default)

= 2 bonds involving hydrogen are constrained. No bonds which are part of the pert group are constrained.

= 3 all bonds are constrained

TOL Relative geometrical tolerance for bond constraints in SHAKE. Smaller values give tighter tolerances. The (default) recommended value is <= 0.0005 Angstrom

TOLR2 Relative geometrical tolerance for angle and torsion constraints (radians). Smaller values give tighter tolerances. The (default) recommended value is <= 0.0001 rad.

NCORC Constraint energy flag.

= 0 No constraint contributions to the free energy are calculated (default).

= 1 The contributions to the free energy from any constraint whose equilibrium value changes with lambda will be calculated. This includes: A) Any constrained internals defined at the end of the input (see flag INTR, line 13); and B) any SHAKE-en bonds (see NTC).

If NCORC=1 is specified, the program will determine which atoms of the system have positions which are dependent on the constraints, and all of these will effectively be included in the "perturbed group". This forces some time- consuming calculations. If no constraints are changing with lambda, be sure to set NCORC=0.

The procedure used to perform a PMF makes it difficult to separate contributions due to the constraints themselves from those due to non-bonded/electrostatic interactions. For this reason, in these cases CORC will reflect the sum total of all three types of contributions and the individual non-bonded/ electrostatic contributions will be reported as 0's.

Note: If you are using a "belly" with NCORC=1, you must ensure that all residues of the pert group are part of the moving belly, and that, additionally, any residues sharing constrained bonds with the pert group (if any) are part of the moving belly.

ISHKFL Flag which determines what the program will do in the event of a SHAKE/internal constraint failure.

= 0 Program halts immediately. This is what the old versions of Amber did.

= 1 Program will write a restart file containing the coordinates before the failed call to the constraint routine (+ velocities, if applicable). The program will then halt.

> 1 The coordinates will not be constrained on any iteration for which the constraint routine fails. If constraint failure occurs on more than ISHKFL-1 contiguous steps, the program will stop as described for ISHKFL=1. This is the default

ITIMTH Defines which method should be used to calculate constraint free energy contributions when NCORC=1 and the Thermodynamic Integration method (IDIFRG=1) approach is being used.

= 0 Use the Potential Forces (PF) method (default).

	= 1	Use the Constraint Forces (CF) method.
	=-1	Use the PF method, override program warnings about constraints within closed rings.
		Two methods for determining the constraint free energy contributions during TI have been derived in the literature. The PF method appears to be more efficient, and so is the default. However, PF method cannot be used when any constraints of the system which are changing with lambda (and hence contribute to the free energy) are part of a closed ring. In this case, the CF method must be used. The program will flag any constraints of the perturbed group which are part of a closed ring, and will stop with a warning if TI is used with PF in such a case. If none of these constrained bonds change with lambda, you can still use the PF method, but must specify ITIMTH=-1 here to ensure you have considered whether this will be appropriate. It is suggested you NOT set ITIMTH=-1 automatically, but only after ensuring that it will be appropriate.
JFASTW	Fast water definition flag. By default, the system is searched for TIP3P waters, and special fast routines are used for these molecules. There are two types of fast routines specific to TIP3P water: 1) A faster, analytic SHAKE algorithm for 3-point water; 2) A faster routine to calculate non-bonded TIP3P-TIP3P water interactions. In normal operation, the program defaults will be acceptable. However, in rare instances (e.g. for debugging purposes, or when the user has redefined the definition of a TIP3P water), one may wish to inhibit the use of these fast routines and/or redefine the default definition used in Amber to define TIP3P waters. This option makes this possible.	
	= 0	Normal (default) operation. The default AMBER definition of TIP3P water is used, and the fast water routines are used where appropriate.
	= 1	Use the fast routines for water SHAKE and non-bonds, but redefine the names the program uses to recognize TIP3P waters. The redefinition names are provided below.
	= 2	Use the fast water routine for SHAKE. Do not use the fast water routine for non-bonds.
	= 3	Use the fast water routine for SHAKE. Do not use the fast water routine for non-bonds. Redefine the names the program uses to recognize TIP3P waters. The redefinition names are provided below (line 17).
	= 4	Do not use fast water routines for either SHAKE or non-bonds.
NTF	Flag for force evaluation. Typically set to the same value as NTC.	
	= 1	complete interaction is calculated (default)
	= 2	bond interactions involving H-atoms omitted, except bonds in the perturbed group (use with NTC = 2, see above SHAKE options)
	= 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
NTID	Flag for solvent pairlist behavior.	
	= 0	only the first atom of each solvent molecule is used when generating the non-bonded pairlist for a periodic system (for water, this is the oxygen). If this atom lies within the specified cutoff, the entire solvent molecule is included in the non-bonded pairlist. This can result in a substantial speedup in non-bonded pairlist generation, and is recommended when using water as the solvent. This is the default.
	=86	all atoms in a solvent molecule are considered when generating the non-bonded pairlist for a periodic system. If any atom of the solvent molecule lies within the specified cutoff, all atoms of the solvent molecule will be included in the non-bonded list. This is the behavior of versions of AMBER <= 3.0.
		A value of NTID=0 is suggested for calculations using water as a solvent. For calculations using larger solvent molecules, one should carefully consider whether using only the first atom is appropriate. Regardless of the value of NTID, all atoms of the *solute* are considered when deciding whether to include a second residue in the interacting non-bonded list for the solute residue. NTID will have no affect for non-periodic systems.
NTNB	Flag for non-bonded pair list generation.	
	= 0	no pair list will be generated (unlikely you would choose this).
	= 1	pair list will be generated (default)
NSNB	After NSNB steps the non-bonded pair list will be updated. Default = 50.	
IDIEL	Type of dielectric function to be used.	
	= 0	distance dependent dielectric function (for in vacuo simulations of "aqueous" systems).
	= 1	constant dielectric function (always use with explicit solvent, e.g. water); default.
IELPER	Flag to control the "electrostatic decoupling" of the perturbation energy	
	= 0	Regular run; no electrostatic decoupling (default).
	= 1	Only the electrostatic contribution to the free energy is calculated keeping the geometry and the VDW parameters pertaining to LAMBDA = 1.
	=-1	Only the non-electrostatic (VDW, etc.) contributions to the free energy are calculated and the system changes from that characteristic of LAMBDA = 1 to 0 (or from that characteristic of LAMBDA = 0 to LAMBDA = 1 depending on the signs of IFTIME or ALMDEL).

In electrostatic decoupling, two runs have to be performed, one for electrostatic and the other for VDW etc. contributions. This is useful when a polar or charged group is being established or removed. However, the LAMBDA = 1 state must pertain to the established group, and the LAMBDA = 0 to the removal of the group. The decoupling MUST go through the following perturbation cycle: electrostatic LAMBDA = 1 -> 0 with LAMBDA(vdw) = 1, followed by van der Waals LAMBDA = 1 -> 0. If the simulation is started at LAMBDA = 0, then reverse the above procedure. In this way, charges never appear on atoms which do not possess a vdw radius which avoids very close contacts due to charge-charge attractions.

Notes: (1) Two separate runs are needed to fully carry out the decoupling calculation. (2) In the IELPER=+1 phase, any added restraints/constraints (if INTR > 0) will be fixed at the values they have when lambda=1. (They will still only be applied, however, over the ranges specified). (3) The free energy contribution from internal constraints is never calculated during the IELPER=+1 phase (it is calculated during the IELPER=-1 phase).

```
---------------------------------------------------------
  To summarize:

   IELPER        internals/vdw      electrostatics
      +1       fixed @ lambda=1         vary
               (non-pert) values

      -1          vary               fixed @ lambda=0
                                          (pert) values
---------------------------------------------------------
```

IMGSLT Flag to control the Solute-Solvent interaction in the case of PB simulation

= 0 The Boundary condition is applied to solute-solvent interactions (default)

= 1 No Solute-Solvent imaging. Solute does not see image solvent. This assumes that the solute is centered in the periodic system, and is not free to migrate. Do not use this with mobile solutes. This option is mainly useful for large solutes.

IDSX0 Flag which controls how the mixed van der Waals parameters are calculated for atom pairs where one atom vanishes (at either lambda=1 or lambda=0). (See Ref. 6).

= 0 r*(state where one atom vanishes) = r*(non-vanishing atom) (This is the way AMBER has done this in the past) Default.

> 0 r*(lambda) will be calculated so that r*(state where one atom vanishes) = IDSX0/1000 r*(state where both atoms exist) = r*(A) + r*(B)

= -1 results in r*(state where one atom vanished) = 0.0

ITRSLU During a periodic boundary conditions simulation, controls whether SOLUTE molecules which exit the primary image box will be translated back into the central box. SOLVENT molecules which exit the central image box are always translated back into the box. A molecule is considered to have floated out of the central box if the first atom of the molecule exits the box.

= 1 Both SOLUTE and SOLVENT molecules which exit the primary image box will be translated back into the box. The system will be translated every 500 steps so that the center of geometry of the solute is centered in the primary image box. (Default; recommended for most systems).

= 2 Same as 1, except that the system as a whole is not periodically translated to keep the solute centered in the primary image box.

= 0 Only SOLVENT molecules will be translated back into the primary image box. SOLUTE molecules are not translated.

IOLEPS Controls how parameter mixing is performed for non-bonded interactions.

= 0 Mixing of epsilon (well-depth) van der Waals parameters done as

$$\varepsilon(\lambda) \ = \ \lambda * \varepsilon(mixed, \lambda = 1) \ + \ (1 - \lambda) * \varepsilon(mixed, \lambda = 0)$$

Mixing of electrostatic interactions done as

$$q_1 q_2(\lambda) \ = \ \lambda * q_1 q_2(\lambda = 1) + \ (1 - \lambda) * q_1 q_2(\lambda = 0)$$

This is the default

= 1 Mixing of epsilon done as

$$\varepsilon(\lambda) = \sqrt{(\varepsilon_i(\lambda)\varepsilon_j(\lambda))}$$

Mixing of electrostatics done as

$$q_1 q_2(\lambda) = q_1(\lambda) q_2(\lambda)$$

Setting IOLEPS=1 forces mixing to be done as in older versions (e.g. 3.0, 3A) of AMBER. The "new" mixing scheme (IOLEPS=0) has several advantages, including A) a finite derivative for van der Waals interactions involving an atom which "disappears" at one end point; and B) Interaction between pairs of atoms where one/both atoms "disappear" at both end points never contribute to the energy. [One side- benefit of this is that it allows duplicate topologies; thus one can perform perturbations using the "CHARMM" methodology, if desired]. Note that if IDIFRG = 1 (thermodynamic integration), the epsilon parameters are always mixed as described for IOLEPS = 0.

INTPRT Determines which energies contribute to the calculation of the free energy change.

= 0 No intra-perturbed group energies are accumulated (Default; same as pre-4.0 versions of AMBER)

= 1 intra-pert. group non-bond energies accumulated as well (but no 1-4’s).

= 2 intra-pert. group non-bond energies accumulated (including 1-4’s).

	= 3	intra-pert group internal energies accumulated (bonds, angles, torsions)
	= 4	intra-pert group non-bond and internal energies accumulated
	= 5	intra-pert group non-bond, 1-4, and internal energies accumulated
		Note: If any PMF contributions are being calculated (NCORC = 1, line 9), *all* intra-perturbed group non-bonded contributions will be calculated if INTPRT = 1,2,4 or 5 (when NCORC=1, 1-4's are not broken out separately).
ITIP	By default (ITIP=0), GIBBS assumes that if you are running a periodic boundary conditions (PBC) simulation with solvent, the solvent is TIPNP water. A special characteristic of this solvent model is that there are no h-bond (10-12) interactions between any pair of solvent molecules. A potential speedup is thus obtained by skipping all such h-bond interactions. If you choose to use a solvent model where there should be h-bond (10-12) interactions calculated between pairs of solvent molecules, set ITIP to any value other than 0. Note that in either case, all 10-12 interactions between solvent and solute molecules will still be determined normally.	
CUT	The primary cutoff distance for the non-bonded pairs. Default = 8.0.	
SCNB	The scale factor for 1-4 vdw interactions; if (SCNB .EQ. 0.0) then SCNB = 2.0, which is the default.	
SCEE	The scale factor for 1-4 electrostatic interactions There is no namelist default, since the 1991 and previous force fields used 2.0, while the 1994 force field uses 1.2.	
DIELC	Dielectric constant for the electrostatic interactions; if (DIELC .LE. 0.0) then DIELC = 1.0. Default is 1.0.	
CUT2ND	An (optional) secondary cutoff. If CUT2ND > 0.0, then at every nonbonded update (every NSNB steps), the energies and forces due to interactions in the range CUT< Rij <= CUT2ND will be determined. These energies and forces will be added to the non-bonded interactions within CUT distance at every timestep. The idea is that long-range interactions change more slowly than short range interactions, and thus this dual cutoff method allows one to include longer-range information at only a moderate additional cost. Default is 0.0.	
CUTPRT	An (optional) alternative cutoff to be used for interactions with the perturbed group. If CUTPRT and CUT2ND are both defined, interactions in the range CUTPRT < Rij <= CUT2ND will constitute the secondary cutoff range for interactions with the perturbed group. Default is 0.0.	
NTPR	Flag for printing energy related quantities. for every NTPR steps these quantities will be output. Default is 100.	
NTWX	Flag for packing the coordinates. For every NTWX steps the coordinates will be dumped through file 'PCOORD' in format (10F8.3). If NTWX=-1 (default), no dumping will be performed.	
NTWV	For every NTWV steps the velocities will be written in file 'PVEL' in format (8F8.4). If NTWV=-1 (default), no dumping will be performed.	

NTWE	Every NTWE steps energy info is written in file 'PEN' in formatted form. If NTWE=-1 (default), no dumping will be performed.
NTWXM	After NTWXM steps the NTWX switch will be inactive. Default is 999999.
NTWVM	After NTWVM steps the NTWV switch will be inactive. Default is 999999.
NTWEM	After NTWEM steps the NTWE switch will be inactive. Default is 999999.

IOUTFM — Flag for format of velocity and coordinate sets

= 0	Formatted (default)
= 1	Binary

ISANDE — Flag to output enthalpies and entropies, as well as free energies. Note that these quantities are typically an order of magnitude or more less precise than free energy values, and will be much more sensitive than free energies to the completeness of the ensemble statistics collected. See the discussion following the input description for more information. Setting ISANDE = 1 will also force the printing of the integrand quantity $< \partial V / \partial \lambda >$ when Thermodynamic Integration is being performed (see the IDIFRG flag, 14.6). This can be useful if the user wishes to apply an alternative integration algorithm. Default is 0.

IPERAT — Request that free energy components or derivatives be calculated. Note that free energy components can be determined during any standard free energy simulation. Free energy derivatives can only be calculated in a special simulation where lambda does not change.

= 0	No free energy components or derivatives will be calculated (default).
= 1	Report free energy components. Components will be be reported in file PATNRG on a per-atom basis.
= 2	Report free energy components. Components will be be reported in file PATNRG on a per-residue basis.
= 3	Report free energy components. Components will be be reported in file PATNRG on a per-molecule basis.
= 4	Calculate/report free energy components or derivatives (depending on the flag ICMPDR). Values will be reported in file PATNRG for the atoms/groups defined at the end of input using GROUP input.

For free energy components, free energies will be logged as defined by the GROUP definition, subject to the condition that only those atoms which are part of the perturbed group or which move with an added CONstraint will ultimately be included. All atoms not explicitly included in a group will be put in a final single group. For free energy derivatives, derivatives will be logged only for those atoms included in a group definition. Any atom of the system may be designated as part of any group (but each atom will be a member of at most one group). Typically, you will place individual atoms in their own groups when calculating derivatives.

IATCMP — If free energy components are being reported, by default only the total free energy per atom/residue/molecule/ group is reported. By setting IATCMP > 0, one can force the components to be broken down into electrostatic, non-bonded and internal contributions. IATCMP has no affect when free energy derivatives are being calculated.

	= 0	Do not break free energy components into contributions (default).
	= 1	break free energy componenets into contributions.
NTATDP		Free energy components/derivatives will only be reported every NTATDP steps. Note that if free energy components are being logged, a free energy report will occur at a particular multiple of NTATDP steps only if the free energy accumulators have been updated since the last report. For free energy derivatives, energies will be reported every NTATDP steps in all cases. If NTATDP = -1, it is set to NTPR; default is 0.
ICMPDR		
	= 0	no free energy derivatives (default).
	= 1	If IPERAT=4, log the free energy derivatives with respect to charge and the non-bonded parameters epsilon and r*. If the contributions of constraints to the free energy are being calculated (NCORC = 1), then derivatives with respect to constraints in the perturbed group (and added constraints) will also be calculated. Free energy derivatives can only be calculated for lambda = 0 or lambda = 1. It is sufficient to define a "null" perturbed group in PARM if you simply wish to determine the non-bonded free energy derivatives of specified atoms.
NCMPDR		If free energy derivatives are being calculated (IPERAT=4 and ICMPDR=1), NCMPDR gives the number of steps of effective "equilibration." After the first NCMPDR steps, the accumulators for the free energy derivatives are cleared and reset. Free energy derivatives reported from this point forward will only reflect averaging since the accumulators were cleared. Some people prefer to use a post-processing program to analyze free energy derivatives. Such programs can usually "remove" a given initial portion of the free energy derivative information from subsequent totals. In such a case, you may wish to set NCMPDR=0 here (no "equilibration" phase), and pick the amount of data to discard in the post-processing program.
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).
	= 0	Coord/velocity archives will include all atoms of the system (default).
	< 0	Coord/velocity archives will include only the solute atoms.
	> 0	Coord/velocity archives will include only atoms 1->NTWPRT.
NTR		Flag for restraining specified atoms.
	= 0	Classical MD (default)

=
MD with restraint of specified atoms

NTRX	Flag for reading the cartesian coordinates for restraint from unit PREFC. Note: the program expects coordinates for all atoms from which a subset is selected by the GROUP input which follows.	
	= 0	binary form
	= 1	formatted form (default)
TAUR	The relaxation time for restraint. Default is 1.0 ps.	
INTR		
	= 0	No additional internal restraints or constraints will be read (default).
	> 0	Additional internal restraints/constraints will be read following the normal input. Storage will be allocated for a maximum of INTR added restraints/constraints. These restraints/constraints can be used for e.g. a PMF calculation.
IBIGM	To calculate the free energy contributions of a constraint (if NCORC=1), the free energy at lambda±d_lambda is evaluated by shifting the value of the constraint to its value at lambda±d_lambda. This change in the value of the constraint can be effected either by performing half of the shift at each end/side of the internal, or by performing the entire shift at one end.	
	= 0	Half of the shift is performed at each end of the internal.
	= 1	The entire shift occurs at the end/side of the internal which results in fewer atoms being moved. This is the default. The number of atoms whose positions change with shifting the constraint affects how quickly the calculation can be performed. Setting IBIGM = 1 can significantly speed up some calculations (e.g. when rotating a ring about a constrained torsion which joins it to a protein), and IBIGM should typically be set to 1 for *in vacuo* simulations. In all cases, GIBBS determines which interatomic nonbonded distances depend on constraint values, and only these are recalculated when NCORC=1.
ISFTRP	Causes the 6-12/10-12 functions used for non-bonded interactions to be replaced by "soft repulsion" terms of the form $$RWELL * (r^2 - r*^2)^2$$ where r* is the optimal interaction distance between a pair of atoms, calculated from their respective van der Waals radii. This function is sometimes useful in structure refinement, but should *not* typically be used in free energy calculations. Atoms in the perturbed group are always treated by normal (6-12 or 10-12) non-bonded forces, regardless of the value of ISFTRP.	
	= 0	regular 6-12/10-12's. No soft repulsion. Default.
	= 1	replace 6-12's by soft repulsion.
	= 2	replace 10-12's by soft repulsion, as well.
RWELL	Force constant (in kcal/mol) used for soft repulsion interactions. Default is 5.0.	
IFTIME	Mutation flag. If ISLDYN=0, then if IFTIME = 0 (default) a standard Window Free Energy Perturbation will be carried out. The perturbation will start	

at lambda = ALMDA, and proceed in equally spaced intervals of delta(lambda) = ALMDEL until 1 (ALMDEL > 0) or 0 (ALMDEL < 0) is reached. At each value of lambda, NSTPE steps of equilibration and NSTPA steps of data collection (see line 15) will be performed, and energy evaluated using Equation 2. If IFTIME =±1 A "Slow Growth" perturbation will be carried out. The simulation will start at lambda = ALMDA, and will be run in either the 0->1 direction (IFTIME = +1) or 1->0 direction (IFTIME = -1). CTIMT gives the number of psec of dynamics which would be used to perform the complete change 0->1 (or 1->0). The actual length of the simulation will depend on the starting value ALMDA. **NOTE** IFTIME is included for backwards compatibility with input files created for previous versions (< 4) of AMBER. However, it is strongly recommended that you use the ISLDYN flag to specify the type of simulation desired. If ISLDYN.NE.0, IFTIME is ignored.

CTIMT	The total length of the MD simulation (in psec) to be carried out in performing a slow growth simulation which transforms state lambda = 0 into lambda = 1 (or vice-versa). Note that this variable does not control the number of steps which will actually be run. For example, if CTIMT = 10psec, ALMDA = 0.0, ISLDYN = +1, and NRUN*NSTLIM*DT = 5psec, only half of the desired simulation would be carried out. The remainder would have to be carried out by a restart. CTIMT is only used when ISLDYN = ±1 or (IFTIME=±1 and ISLDYN = 0). Default is 0.0.
ALMDA	The starting value of lambda for this simulation. The value can be on the inclusive interval 0.0-> 1.0. Default is 1.0. ALMDA = 1 corresponds to the "initial" state and ALMDA = 0 corresponds to the "final" state. Intermediate "states" are defined by a linear combination of the parameters representative of (lambda = 0) and (lambda = 1). For restart simulations (IREST=1, line 2), ALMDA is read directly from the restart file, and the value specified here is ignored.
ALMDEL	For _Standard_ (fixed width) Window and TI simulations, ABS(ALMDEL) gives the width of each window or integration interval. If double-wide sampling is used with Window Growth (default), at each value of lambda, the free energies to both +ALMDEL and -ALMDEL are evaluated. This results in "double wide sampling" (see the introductory text). If (IFTIME=0 and ISLDYN=0), the sign of ALMDEL determines the direction of the change. If ISLDYN=±3, the sign of ISLDYN determines the direction of the change. ALMDEL should be chosen so that the free energy change over any interval is not too large. It has been suggested (somewhat arbitrarily) that as a rule the free energy change/window should not exceed 2RT. ALMDEL is only used when ISLDYN = ±3 or (IFTIME=0 and ISLDYN = 0). Default is 0.1.
ISLDYN	Free Energy Method flag. It is recommended that you use this flag exclusively, and ignore IFTIME. Default is -3.

= ±1	Perform a Slow Growth simulation. The simulation will be started at ALMDA, and CTIMT psec will be required to complete the conversion to the end (0 or 1). If ISLDYN = +1, the simulation will be carried out in the direction 0-> 1. If ISLDYN = -1, the simulation will be carried out in the direction 1-> 0.

= ±2 Perform a Dynamically Modified Window simulation. The simulation will be started at ALMDA and progress either in the direction 0-> 1 (if ISLDYN = +2) or 1-> 0 (if ISLDYN = -2). The numbers of equilibration and data collection steps performed at each window are given by NSTMEQ and NSTMUL (on this line). If IDIFRG = 0, the energy will be evaluated at each interval using Equation 2 (FEP). If IDIFRG = 1, thermodynamic integration will be carried out using Equation (4).

= ±3 Perform a "standard" Window Growth simulation (with fixed width lambda intervals). The perturbation will start at lambda = ALMDA, and proceed in equally spaced intervals of delta(lambda) = abs(ALMDEL) until 1 (ISLDYN > 0) or 0 (ISLDYN < 0) is reached. At each value of lambda, NSTMEQ steps of equilibration and NSTMUL steps of data collection (see this line) will be performed. If IDIFRG = 0, the energy will be evaluated at each interval using Equation 2 (FEP). If IDIFRG = 1, thermodynamic integration will be carried out, using Equation (4).

IDIFRG Thermodynamic integration flag.

= 0 No thermodynamic integration (default).

= 1 If windows or dynamically modified windows have been specified, the energy will be calculated using thermodynamic integration (TI) (Equation 4). The integrand will be evaluated at the endpoints of each "window", and the integral will be approximated using the trapezoidal rule (see the discussion following the input description). In addition to the integrated free energy, if ISANDE is set = 1 (see flag 12.10), the value of $< \partial V / \partial \lambda >$ will be output at every energy update, so a different integration algorithm can be applied by the user, if desired. If slow growth has been requested, setting IDIFRG=1 has the effect of performing the slow growth summation using the non-averaging equivalent of the TI equation (4), rather than the FEP equation (2).

NSTMEQ number of steps of equilibration to be used for each window if ISLDYN = ±2 or +-3. (Note that if windows are instead requested using the flag combination IFTIME = 0 and ISLDYN = 0, NSTPE is used). Default is 2.

NSTMUL number of steps of data collection to be used for each window if ISLDYN = ±2 or +-3. (Note that if windows are instead requested using the flag combination IFTIME = 0 and ISLDYN = 0, NSTPA is used). Default is 2.

NDMPMC Every NDMPMC windows, statistics will be dumped to the statistics file (MICSTAT). The statistics file contains a condensed format record of the free energy for each window interval. The MICSTAT file is not written with slow growth, or if NDMPMC is set < 0. By default NDMPMC=100. NDMPMC cannot exceed 100.

IDWIDE Allows double-wide sampling to be turned off with FEP.

= 0 Double-wide sampling performed when FEP windows are being calculated (default).

= 1 Double-wide sampling turned off when FEP windows are being calculated.

Double wide sampling means at each value we calculate the free energy in both the "forward" and "reverse" direction. This gives an intra-run consistency check (lower bound on the error), but requires we calculate every interval twice. The simulation can be run in roughly half the time, without the forward/reverse consistency check, by setting IDWIDE=1. The nature of thermodynamic integration (IDIFRG=1) is such that double wide sampling is never carried out. IDWIDE has no effect for such calculations.

IBNDLM By default (IBNDLM=0), lambda±d_lambda is constrained to the range 0<lambda±d_lambda<1. If IBNDLM=1, then lambda±d_lambda can exceed the range 0->1. Useful when doing PMF-type calculations. Ignored for regular slow growth.

IAVSLP The current dG/dLAMBDA slope will be approximated by a linear fit to the Accumulated G vs. LAMBDA data for the previous IAVSLP windows. Maximum value = 1000; default is 8.

IAVSLM Until IAVSLM windows have been collected, the window spacings will be fixed at ALMDL0 (line 14c). When IAVSLM windows have been collected, the slope will be calculated over all available windows, until IAVSLP windows are available. Default is 2.

```
i.e. #_windows < IAVSLM : dLAMBDA = ALMDL0
    IAVSLM <= #_windows < IAVSLP :
          dLAMBDA calculated from slope over #_windows
    #_windows >= IAVSLP :
          dLAMBDA calculated from slope over previous IAVSLP
          windows
```

If IAVSLM=-1, window widths will be fixed at ALMDL0 until IAVSLP windows are available.

ISLP Determines the direction in which the slope is calculated.

= 0 (default) use the appropriate value of ISLP (-1 or 1) to calculate the slope from energies calculated in the same direction as the simulation (recommended).

= 1 the slope is calculated from the forward (0->1) energy at each step.

=-1 the slope is calculated from the reverse (1->0) energy at each step.

= 2 the slope is calculated using the average of the redundant free energy values (from double wide sampling) over the interval in the direction opposite to the simulation, i.e. G(reverse[curr window] - G(forware[prev window])/2 or G(forward[curr window] - G(reverse[prev window])/2 for simulations run 0->1 and 1->0, respectively. This option can be useful when very few points are used to evaluate each slope (e.g. IAVSLP = 2).

= 3 the slope is calculated using the average of the forward and reverse energies at each lambda.

For best results in most cases, the slope should be calculated in the same direction as the simulation. This is the default behavior (ISLP=0). With thermodynamic integration, or when double-wide sampling is defeated, ISLP has no effect. Only options ISLP=0 or ISLP=3 should typically be used when AMXRST > 0.

CORRSL	If the correlation coefficient for a linear fit to the previous IAVSLM windows is < CORRSL, the number of windows over which the slope is calculated will be halved (for this determination of the slope only), and the slope calculated again. This process continues until the correlation coefficient is > CORRSL. Default is 0.8.
AMXMOV	The target free energy change per window. If M is the slope over the previous IAVSLP windows, the next value of dLAMBDA is chosen as dLAMBDA = AMXMOV/M Note that when double wide sampling is defeated (IDWIDE=1) while using a window FEP technique (IDIFRG=0), the free energy change at a window is defined as the total ("forward" + "reverse") energy change. This differs from the definition when double wide sampling is used, where the free energy change at a window is approximately 1/2 * ("forward" + "reverse"). Thus, AMXMOV should be suitably increased when IDWIDE = 1. Default is 0.1.
IAVDEL	Number of windows over which the forward and reverse energies will be compared. If IAVDEL<0, no comparisons will be carried out. IAVDEL should always be set <0 when thermodynamic integration is used (IDIFRG = 1). Maximum value = 1000; default is -1.
IAVDEM	The relationship between IAVDEL and IAVDEM is analogous to that between IAVSLP and IAVSLM. Default is 2.
AMXDEL	If < ABS (DA(for)-DA(rev)) > .GT. ABS(AMXDEL) then the next dLAMBDA will be scaled as [< ABS (DA(for) - DA(rev)) > / AMXDEL] **2 * dLAMBDA If AMXDEL < 0, then scaling occurs in all cases. Default is 1.0.
ALMDL0	Until enough intervals have been calculated to allow determination of dG/d_lambda and d_lambda consistent with IAVSLP and IAVSLM, an interval width of ALMDL0 will be used. Default is 0.0001.
DLMIN	The minimum allowable window width. Default is 1.0D-6.
DLMAX	The maximum allowable window width Default is 0.1.
AMXRST	If the free energy change, dG, over any window is greater than AMXRST, then the data collection phase for that window will be re-performed using a reduced value of dLAMBDA. The new value of dLAMBDA is determined as dLAMBDA(new) = (dLAMBDA(old)/dG) * AMXMOV. AMXRST should not be set too close to AMXMOV, or too many windows will be recalculated (which is inefficient). By default, AMXRST=5.*ABS(AMXMOV).
NORSTS	If this is a restart run, and NORSTS=1, then the restart information relating to dynamically modified windows is not read (cold start for the dynamically modified windows). NORSTS is ignored if this is not a restart run. Normally, NORSTS should be set to the default of 0.
NTSD	The statistics relating to dynamically modified windows are written to file POUT every NTSD. If NTSD=0, then NTSD is set equal to NTPR (line 12),

and these statistics will be output every time the standard energy information is printed. Default is 0.

ALMSTP(1) Allows the values of AMXMOV, DLMIN, DLMAX, AMXRST, and NTSD to be different for different ranges in LAMBDA.

> 1 or < 0
the values defined in lines 14a-14c will remain in effect for the whole run.

> 0 and < 1
the values defined in lines 14b-14d will remain in effect for the range of LAMBDA ALMDA-> ALMSTP(1). In this case, _additional line(s)_ are read with the values of the above variables over various ranges of LAMBDA. Each line has the format

```
AMXMOV, DLMIN, DLMAX, AMXRST, NTSD, ALMSTP(I)

FORMAT(4F14.9,I5,F14.9)
```

These lines are read until ALMSTP(I) > 1 or ALMSTP(I) < 0. Each set of values applies to the range in LAMBDA ALMSTP(I-1) -> ALMSTP(I). Note that the for the last line, ALMSTP(I) must be greater than 1, or less than 0 (not equal to). This is avoid machine precision problems. Note also that, at present, "namelist"-format input always assumes ALMSTP(1) < 0 (i.e. AMXMOV, DLMIN, etc. remain fixed over the entire run). If you wish to use the functionality described above for ALMSTP(), you must use formatted input.

NSTPE The number of steps of Equilibration before collecting the Free Energy Statistics. For each window the system is equilibrated for NSTPE steps. (When ISDYN=±2 or ±3, NSTMEQ serves the same purpose). Default is 2.

NSTPA The number of steps for data collection. The averaging is performed over this number of steps. (When ISLDYN=±2 or ±3, NSTMUL serves the same purpose). Default is 2.

DTA The time-step used for window runs specified by IFTIME=0 and ISLDYN=0. All other runs use the time-step specified on line 8. Default is 0.001

IVCAP Flag to control Cap Option. The Cap option is to solvate a spherical portion of a solute and to hold the solvent from evaporating through a half-harmonic potential.

= 0 Cap will be in effect if it is passed from the PARM file (default).

= 1 Cap will be activated except that the Cap atom pointer would be modified.

= 2 Cap will be inactivated.

NATCAP The Cap atom pointer It is the last Non-Cap atom number. If IVCAP.EQ.1 then the pointer passed from the PARM file will be overwritten by this number. Default is 0.

FCAP The Force Constants for the Cap Atoms. Default is 0.0

WATNAM	The residue name the program expects for TIP3P waters. Default is "WAT".
OWTNM	The atom name program expects for the TIP3P oxygen. Default is "O ".
HWTNM1	The atom name program expects for the TIP3P 1st H. Default is "H1 ".
HWTNM2	The atom name program expects for the TIP3P 2nd H. Default is "H2 ".

The following card is read *only* if I3BOD.NE.0; (This information must be provided in the formatted form given, even if namelist format input is used above.)

```
      - 18A-    1) N3B,  NION           FORMAT(2I5)
```

The number of 3body interactions to be defined, and the number of ions in the system.

Next, include N3N cards 18B to define all 3-body interactions:

```
      - 18B-    1)AT1(I)  2)AT2(I)  3)ACON1(I)  4)BETA31(I)  5)GAMMA31(I)
                                    6)ACON0(I)  7)BETA30(I)  8)GAMMA30(I)

                  FORMAT(A4,A4,2X,6E10.3)
```

AT1(I)	The second atom in this 3-body interaction.
AT2(I)	The third atom in this 3-body interaction.
ACON1(I)	The pre-exponential factor for this 3-body interaction for the lambda = 1 state.
BETA31(I)	The beta value for this 3-body interaction, for the lambda = 1 state.
GAMMA31(I)	The gamma value for this 3-body interaction, for the lambda = 1 state.
ACON0(I)	The pre-exponential factor for this 3-body interaction for the lambda = 0 state.
BETA30(I)	The beta value for this 3-body interaction, for the lambda = 0 state.
GAMMA30(I)	The gamma value for this 3-body interaction, for the lambda = 0 state.

```
        - 19 -      IDENTIFICATION OF ATOMS WITH POSITION CONSTRAINTS
                         *** ONLY IF NTR = 1 ***
```

Constraint reference atoms are obtained by first reading coordinates for the entire structure through file 'PINCRD' or 'PREFC', then specific constraint atoms are selected by group. See the section on GROUP in the Appendices for format. Does not support a namelist convention.

```
      - 20 -      IDENTIFICATION OF ATOMS FOR BELLY RUN
                       ***** ONLY IF IBELLY .GT. 0 *****
```

The belly atoms are loaded as groups. Consult the GROUP section in the Appendices for a description of how to define a group. The group definition immediately follows the end of the

&cntrl namelist. *The GROUP input does not support a namelist convention.*

```
- 21 -      DEFINITION OF GROUP INPUT FOR FREE ENERGY COMPONENTS
            OR DERIVATIVES ***** (ONLY IF IPERAT = 4) *****
```

For free energy components, free energies will be logged as defined by the GROUP definition, subject to the condition that only those atoms which are part of the perturbed group or which move with an added CONstraint will ultimately be included. All atoms not explicitly included in a group will be put in a final single group.

For free energy derivatives, derivatives will be logged only for those atoms included in a group definition. Any atom of the system may be designated as part of any group (but each atom will be a member of at most one group). Typically, you will place individual atoms in their own groups when calculating derivatives.

Note that in GIBBS, GROUP input supports two new features that can be helpful in defining the input for free energy components or derivatives. Both allow the creation of multiple single-atom groups:

```
ATOM -IAT1 IAT2
```

(1st atom number negative) will place each atom from IAT1 to IAT2 in its own group.

```
RES -IRES1 -IRES2
```

(both residue numbers negative) will place each atom of every residue in the range IRES1->IRES2 in a separate group. Group definition syntax is otherwise the same as described in the manual.

```
- 22 -      DEFINITIONS OF INTERNAL RESTRAINTS/CONSTRAINTS
                  *** ONLY IF INTR > 0 (line 13) ***
```

Setting INTR > 0 allows the user to define here a series of internal restraints and constraints whose force constants and equilibrium values are a function of lambda. *Restraint/constraint definitions must be entered in the formatted form shown below, not in a namelist.* `Restraints/constraints are read in as pairs of lines:`

```
line A: IAT1,IAT2,IAT3,IAT4,IUMB,IZE,ITOR,RLMDA1,RLMDA2
        FORMAT (7(I5,1X),2F10.5)
line B: RKEQ1,REQ1,RKEQ2,REQ2,IPER,IPER2
        FORMAT (4F10.5,2I5)
```

As many restraints/constraints may be defined as are desired. A blank record signals the end of the input. This data must be entered in the formatted form shown. *It does not support a namelist convention.*

IAT1-->IAT4 The absolute atom numbers for the atoms defining the restraint. If an atom number is <0, the absolute value of the atom number is used (additional behavior for <0 values is defined when IZE=1; see below).

```
IAT3 = IAT4 = 0    :  Bond restraints/constraints
IAT4 = 0           :  Angle restraints/constraints
IAT1->IAT4 non-zero:  Torsion restraints/constraints
```

RLMDA1

RLMDA2 The restraint/constraint will be applied only over the range in lambda (RLMDA1, RLMDA2).

RKEQ1

REQ1 The force constant in kcal/mol and equilibrium value, respectively, for the restraint/constraint at lambda = RLMDA1.

RKEQ2

REQ2 The force constant in kcal/mol and equilibrium value, respectively, for the restraint/constraint at lambda = RLMDA2. If RLMDA1=RLMDA2, the force constant and eq. value are fixed at RKEQ1 and REQ1 (RKEQ2 and REQ2 are ignored). RKEQ1 and RKEQ2 are ignored for constraints (ITOR=2). If REQ1=999. or REQ2=999., the corresponding equilibrium value is set to the current value of the internal coordinate (as determined from the input set of coordinates PINCRD). If ABS(REQ1) > 1000, the corresponding equilibrium value is set

```
REQ1 < 0:    REQ1 = current_value - [ABS(REQ1)-1000.]
REQ1 > 0:    REQ1 = current_value + [ABS(REQ1)-1000.]
```

If ABS(REQ2) > 1000, REQ2 is analogously reset.

Intermediate $K_{eq}(\lambda)$ and $R_{eq}(\lambda)$ are determined by linear interpolation between the force constants and equilibrium values at RLMDA1 and RLMDA2. No restraint/constraint is applied outside the range (RLMDA1,RLMDA2).

IZE

= 0 The restraint/constraint defined here is used _in addition to_ other parameters corresponding to this atom sequence from parm (if any).

= 1 The restraint parameters defined here _replace_ overlapping parameters from parm (if any) for this atom sequence. When IZE=1, any atom number IAT1->IAT4 which was specified as < 0 has a special meaning: It allows a "wildcard" match to the corresponding atom number when replacing parameters from parm. For example, the sequence -1 3 8 -14 would result in a torsional restraint which would replace parameters for all torsions centered on the bond between atoms 3 and 8. IZE is read but ignored when ITOR=2 (constraints).

IUMB Determines the type of restraint.

= 0 The restraint is to be considered part of the molecular force field. The free energy contribution from the restraint is calculated by the standard formula (c.f. Equation 2).

= 1 The restraint is considered to be an "umbrella" term. The effects on the ensemble of the restraint are evaluated using the following function in place of Equation 2:

$$\Delta G = -RT \ln(< e^{-\Delta V/RT} e^{\phi/RT} >_{V+\phi} / < e^{\phi/RT} >_{V+\phi}) \quad ,$$

where ϕ is the sum of all umbrella restraint terms and ΔV is as described for Equation 2. IUMB is ignored for constraints (ITOR=2). IUMB = 1 will not work correctly with slow growth or thermodynamic integration.

ITOR Functional form/constraint flag.

= 0 If this is a torsional restraint, a potential of the form

$$K_{tor}\ (\tau - \tau_0)^2$$

is used. This functional form is always used for bonds and angles (ITOR = 0 has no effect for bonds/angles).

= 1 If this is a torsional restraint, a potential of the form

$$K_{tor}\ (1 - \cos(\tau - \tau_0))$$

is used. (ITOR = 1 has no effect for bonds/angles).

= 2 Then a constraint, rather than restraint, is applied to the corresponding internal coordinate. This is applicable to all types of internal coordinates (distances, angles, torsions). If NCORC = 1 (line 9), then an effective "potential of mean force" (PMF) contribution to the free energy will be calculated for this internal coordinate. General "holonomic" internal constraints are used, as described in Reference 7.

When ITOR = 2 (internal is being constrained), IZE is ignored, and the following occurs:

For bonds and angles, if the constrained internal matches an internal in the topology file, force constant parameters for matching internal will be set to 0.

For torsions, if the constrained internal matches an internal in the topology file, A) forces for all torsions centered on the same bond will be omitted B) The contributions to the free energy of all torsions centered on the same bond as the constraint will be calculated. This is necessary because several torsions can be centered on a central bond, and there is no fixed relative arrangement for these torsions.

IPER IPER can be used to define a simulation where two internal coordinates will be varied with two independent values of lambda. Such a simulation can be used to generate a free energy internal-internal map (sort of a free energy equivalent to a Ramachandran map) to be generated.

The output of this option is somewhat complex, and is intended for post-processing by a separate program. Any 2-D run of value will necessarily be very compute-intensive, and a number of issues must be considered before undertaking such a simulation. This option should generally be avoided by the novice user. If you are considering performing such a simulation, you are *urged* to read Reference 8 (see above) first. For use with the IPER flag, we define:

primary lambda
: the "normal" lambda; that is, the lambda used in standard GIBBS runs to describe how the system varies between the initial and final states.

secondary lambda
: a second lambda, which is translated from 0->1 at each value of the "primary" lambda.

= 0 This restraint will vary with the primary lambda; i.e. the equilibrium value and force constant will be a function only of the primary lambda. This is standard behavior.

> 0 This restraint will vary with the secondary lambda; i.e. the equilibrium value and force constant will be a function only of the secondary lambda. Lambda will be varied from 0->1 for this restraint in a series of IPER equally-spaced intervals (windows). The "secondary" lambda is not used unless one or more restraints are defined with IPER > 0.

The number of windows used for each "primary" restraint will be the same, and the number used for each "secondary" restraint will be the same. The first IPER(I) > 0 sets the number of windows used for _all_ secondary restraints.

If secondary restraint(s) are requested, the value of IPER2 (see below) corresponding to the first value of IPER(I) > 0 defines the number of windows used for every primary restraint. Note that any dynamically modified window or slow growth flags (card 14) will be defeated in this case.

When calculating PMF-type energies (if NCORC=1), constraints will be applied in two cycles. First, dG will be calculated for +-d(internal) for only those internals for which IPER=0. Then a dG will be calculated +-d(internal) for only those internals for which IPER>0.

Any parameters (other than constraints) that vary with lambda will only change when lambda for the primary constraints changes.

If IPER > 0, window or dynamically-modified window growth must have been requested (line 14). IPER cannot be set > 0 with slow growth or with thermodynamic integration (IDIFRG > 0).

The matrix of energies from a 2-D run is contained in file CONSTMAT. A matrix can be generated with either IDWIDE = 0 or IDWIDE = 1, but it is strongly recommended that IDWIDE = 1 (no double-wide sampling) be used. In this case, five free energy difference are evaluated from each ensemble, corresponding to moves from (lam1, lam2) to (lam1, lam2+d_lam2), (lam1, lam2-d_lam2), (lam1+d_lam1, lam2), (lam1-d_lam1, lam2), (lam1-d_lam1, lam2-d_lam2). This set allows the whole free energy map to be evaluated most efficiently (see the Pearlman and Kollman reference [8] noted above).

	The "secondary" lambda always changes in the "forward" direction, always starts at 0.0, and always ends at 1.0. After lambda has gone from 0->1. The primary lambda is incremented one step, the secondary lambda is reset to 0, and another cycle of secondary lambda changes occurs. At the start of each cycle of changes in the "secondary" lambda, the current coordinates are stored in file CNSTSCRT.
IPER2	If IPER > 0 for a particular restraint/constraint ("secondary" restraints defined), IPER2 gives the number of "windows" used in translating the "primary" lambda from 0 to 1. See the description of IPER above. If IPER > 0, IPER2 fixed-width windows will be used for the "primary" restraints, regardless of the behavior requested by ISLDYN, etc. (lines 14-ff).

7.8. Choices Affecting Free Energy Calculations

David A. Pearlman
(minor modifications by DAC)

The development of ever-more-powerful computers, combined with the wide dissemination of modeling packages like AMBER, puts the power to perform valuable calculations in the hands of an increasingly large number of scientists. It is tempting to say that, given the increasing sophistication of such programs, all one needs is the appropriate hardware and software to perform good experiments.

But this is not the case. As modeling programs have grown more sophisticated, they have sprouted an ever-increasing array of options−options which must be properly chosen, if worthwhile results are to be obtained. And even if the options are appropriately set, one must ensure that the program is properly suited for the chosen application. Nowhere in AMBER is this more true than the GIBBS free energy module.

Here we discuss several issues which impinge on developing an appropriate GIBBS input file, and on interpreting the results produced. One is also strongly encouraged to review the literature referenced here and in the preface to the GIBBS program.

7.8.1. What method should be used to calculate the free energy?

GIBBS offers five choices of method for calculating the free energy difference between two states. These include the general classes slow growth, free energy perturbation, and thermodynamic integration, as well as dynamically modified variants of the latter two. These were described in the introduction to GIBBS. As yet, it has not been shown conclusively what method is "best" for any particular type of problem.

(1) Slow growth: This method to some extent has had a bad reputation, since an implicit assumption−that λ changes slowly enough that the system can be assumed to be in equilibrium at each step−does not strictly hold. The consequences of this "Hamiltonian lag" have not been fully quantified. There is recent interest in converting the original algorithms into "fast growth" techniques [92]; these approaches look very interesting but will not be covered here.

(2) Window Growth: The equations of window growth, or Free Energy Perturbation (FEP) are exact, and, in principal, if one has the computer resources to perform sufficient sampling, one can obtain very accurate results. In practice, FEP suffers from two significant

difficulties. The first is that, in reality, we do not always sample to convergence. Unfortunately, no reliable test to prove convergence has been developed. The second problem with FEP is that Equation (2) requires that we obtain the ensemble average of a quantity which relies of the *difference* between the potential functions representative of both states $\lambda(i)$ and $\lambda(i+1)$. But the average is evaluated from the ensemble of states visited when MD is run using the potential function for state $\lambda(i)$. Thus, if states $\lambda(i)$ and $\lambda(i+1)$ are too dissimilar, it will be very difficult to obtain reliable statistics. Reducing the spacing between adjacent λ states helps circumvent this problem, but at a significant additional cost. And even then we do not have any reliable methods for assuring the problem has been avoided.

(3) Thermodynamic Integration (TI): TI is appealing because it avoids the problem in sampling the exponential of V($\lambda(i+1)$)-V($\lambda(i)$) described for FEP above. But TI has its own problem: The driving equation of TI is an integral (Equation 4), which in practice must be calculated approximately by evaluating the integrand at discrete values of λ. Of course, TI is also susceptible to errors when a simulation is not run sufficiently long to obtain a converged value of the averaged quantity which serves as the integrand.

Note that we approximate the integral using the trapezoidal algorithm, i.e.

$$\Delta G_i \ = \ G(\lambda(i+1)) + G(\lambda(i)) \ = \ (< \partial V/\partial\lambda >_{\lambda(i+1)} - < \partial V/\partial\lambda >_{\lambda(i)}) \ \ (\lambda(i+1) - \lambda(i))/2 \quad . \quad (5)$$

This integration method should be reasonably accurate in most cases. But in case the user wishes to try their own integration scheme, setting ISANDE = 1 with TI will also force reporting of the values of $< \partial V/\partial\lambda >_{\lambda(i)}$ and several other averages at every evaluation point (the other values reported relate to calculating the enthalpy/entropy, as described below).

(4) Dynamically Modified Windows (DMW): In dynamically modified windows [85] the $\delta\lambda$ spacing between consecutive windows in FEP or TI is continually changing, to achieve a relatively constant free energy change per window. This should improve the efficiency of the calculation, by focusing proportionately more simulation time on those ranges of λ where the free energy is changing more rapidly. We have, in fact, shown that dynamically modified windows significantly improve the sampling efficiency of FEP simulations for model compounds. The biggest drawback to DMW is that, because we do not know *a priori* the exact shape of the free energy versus λ curve when we start a simulation, we cannot predict with certainty how long the simulation will take to go to completion. This caveat noted, it appears that DMW would be beneficial to most FEP and TI simulations.

7.8.2. Enthalpies and entropies

GIBBS allows the user to request that the enthalpy and entropy changes be reported in addition to the free energy (which is always reported). Two different schemes are used to calculate these quantities, depending on the free energy calculation method. Note that in either case, the enthalpy and entropy are necessarily dependent on being able to reliably extract small differences between averages of (often large) total system energies. In the case of free energy, on the other hand, we need only measure the average of a potential difference or a derivative. For this reason, enthalpy/entropy estimates are typically more than an order of magnitude less accurate than their free energy counterpart. One should be very cautious when interpreting them.

For FEP, the approximate equations given by Fleischman and Brooks [93] are used. These approximate the required temperature derivatives by a finite difference. The equations used are derived from the FEP expression, and the sum of the resulting (enthalpy - T*entropy) will equal the reported free energy.

For TI, the enthalpy and entropy are evaluated using exact-form integral relationships given by Yu and Karplus [94] (and in other places). The (enthalpy - T*entropy) calculated by this method will not necessarily equal the reported total free energy; the difference between the two quantities can be taken as a crude indication of the reliability of the enthalpy/entropy values. The integrals are approximated by the trapezoidal rule, as described above (Equation (5)).

7.8.3. Mixing rules for vanishing atoms

By default, the optimal interaction r^*_{ij} between two atoms i and j is given by

$$r^*_{ij}(\lambda) = r^*_i(\lambda) + r^*_j(\lambda) \tag{6}$$

This is fine when neither atom "vanishes" at either λ endpoint. But now consider the case where atom i vanishes at λ=0. Then

$$r^*_{ij}(0) = r^*_i(0) + r^*_j(0) = r^*_j(0) \quad . \tag{7}$$

Thus, r^*_{ij} never gets smaller than $r^*_j(0)$. At λ=0, the mixed well depth, $\varepsilon(0)$, will also be 0. But at any value of λ just slightly >0, $\varepsilon \neq 0$, and suddenly a steric "gap" between atoms i and j of r^*_j will be required. This can lead to sampling inefficiencies. A better solution is to shrink $r^*_{ij}(\lambda)$ to a user-chosen small value as one of the atoms "vanishes". This is the effect of variable IDSX0 (line 10).

7.8.4. Using Dynamically Modified Windows

The theory of DMW, and some exploratory applications, are described elsewhere [85]. A sample input for GIBBS is shown below, follow by a few important explanatory notes.

```
line
14     0  40.00000  0.00000  -0.02500  +2  0  100  100  0  0  0  0
14a    8    2   0   0.8000000   0.0100000
14b  -10   20   0.0001000
14c   1.0D-5  1.D-10   1.0D-2        0.10000000    0    0  -1.00

(format compressed to fit page)
```

Line 14
We set ALMDEL = 0, ISLDYN=+2, IDIFRG=0, NSTMEQ=100, and NSTMUL=200. This results in dynamically modified window FEP, with 100 steps of equilibration and 100 steps of data collection per window.

Line 14a:
On the next line, we set IAVSLP = 8, IAVSLM=2, and CORRSL=0.8. This means that, at most, the 8 most recently calculated (λ, accumulated_free_energy) points will be used in approximating the $\partial G/\partial\lambda$ slope. IAVSLM=2 means that as soon as 2 points are available, we will calculate the slope from all available points, until the maximum of 8 is reached. If the best-fit line through the points fits the data with a correlation coefficient (CC) < 0.8, then the number of points used in the current slope determination will be halved, the slope and CC will be recalculated, and the comparison against CC will be performed again. A minimum of two points are always used to calculate the slope.

AMXMOV, which is set to 0.01 here, is the target change in free energy per window we are aiming for. The $\delta\lambda$ change on the next step is calculated as

$$\delta\lambda = \frac{AMXMOV}{(\partial G/\partial\lambda)} \tag{8}$$

Note that since we don't know *a priori* what the free energy versus λ curve will look like, we do not know exactly how many steps will be required to complete the simulation. The total number of MD steps required will depend both on AMXMOV and on NSTMEQ and NSTMUL (line 14). NSTLIM can be set to -1 on line 8 to force the program to continue until the total required number of steps have been performed. Also note that the value of AMXMOV used will often depend on the magnitude of the total anticipated free energy change. For example, one would not typically want to use AMXMOV=0.01 and NSTMEQ=NSTMUL=100 if the total energy change is 50 or 100 kcal/mole, as it can be for certain electrostatic changes.

line 14b:
IAVDEL < 0, which means that the $\Delta G_{forward} - \Delta G_{reverse}$ comparisons will not be used in scaling the widths of λ windows. The viability and reliability of changes made using these types of comparisons has not yet been established.

line 14c:
ALMDL0 is set to 1.0D-5. This means that the first IAVSLM window steps (before we have enough points to calculate a slope) will be made with this small step size. This step is chosen to be small in case the energy is changing quickly in this region.

DLMIN is set to 1.0D-10. Typically, a value of DLMIN such as this would have no effect, since it is unlikely that the slope and AMXMOV would be such to require a step this small in the first place. At any rate, steps calculated to be smaller than DLMIN are reset to DLMIN. DLMIN can be valuable in some cases when one wishes to limit how slowly a simulation can progress.

DLMAX is set to 1.0D-2. Setting an appropriate value for DLMAX is important. If the G versus λ curve has any points of inflection, we might calculate a slope of approximately 0 at one or more points. In this case, the simple formula used to determine the next step size would indicate a very large step (as large as 1.0, the whole simulation length). This would be incorrect, as the slope could clearly turn significantly non-0 in a future range of λ. DLMAX bounds the change in such cases.

AMXRST is set to 0.10. The slope we calculate is only an approximation of the "true" instantaneous slope, and the current slope is only an estimate of the slope over the next λ interval (window). Thus, it is possible that when we calculate the actual free energy change over the next window, it will be an unacceptable amount larger than the target value. In such a case, we may want to decrease the λ step size for this window and re-evaluate the energy. AMXRST is the largest allowable energy for a step. If the energy is > AMXRST, the $\delta\lambda$ stepsize is reduced, and the energy for the window recalculated. Note that setting AMXRST too close to AMXMOV will result not only in too many windows being reevaluated (inefficient), but can also lead to biased sampling.

ALMSTP(1) is set to -1.0. If 0 < ALMSTP(1) < 1.0, one can prescribe that the values of AMXMOV, DLMIN, DLMAX and AMXRST vary over different ranges in λ, as described in the input discussion.

7.8.5. Potential of Mean Force (PMF) calculations

It is often of interest to determine the free energy difference between two states which differ in conformation, rather than in composition. For example, one might be interested in the free energy profile for rotation about a ring in a protein. Such a profile can be determined by performing a PMF simulation. To perform such a simulation, one must be able to define conformation as a function of lambda within the context of an otherwise free MD simulation. Fortunately, methods have been developed which allow selected internal coordinates to be constrained to chosen values, while otherwise affecting the MD trajectory only minimally. The best known of these is the SHAKE method for bond constraints. The methods of SHAKE can be extended to be generally applicable to angles and torsions. One can calculate the free energy changes that accumulate as the internal constraints are translated from those of the initial state to those of the final state. If one graphs the free energy changes as a function of the restraint target values (themselves a function of λ), one gets the free energy profile for conformational changes.

Any constraint with a target value which is itself a function of λ will contribute to the free energy as lambda changes. This means that if SHAKE is used to constrain bonds of the perturbed group, and any of those bonds "grow" or "shrink" during the simulation, there will be a corresponding contribution to the free energy. In earlier work, this contribution has been overlooked, but we have shown that it must be included to reliably calculate free energies using the FEP method [86]. The contribution in such a case can be calculated simply by setting NCORC=1.

Constraints other than SHAKE-en bonds can be defined by setting INTR > 0 (line 14) and providing the definitions after the standard input (see above). Any internal coordinate can be used; Be aware, however, that any internal coordinate which is part of a closed ring will present a special set of (often tricky) considerations (see below). In typical use, no compositional (or topological) change is performed when a PMF simulation is being carried out. A GIBBS-format topology file is still required from LEaP, though. Here, one does not define any per-atom perturbations ("edit molecule" / Selection / Edit selected atoms to turn per-atom pert on/off) and does a

```
> saveamberparmpert molecule nullpert.top nullpert.crd
```

In general, PMF calculations within GIBBS may be performed with any method − FEP, TI or slow growth. Note that there is one scenario where *only* the TI (with "constraint forces") method may be used: when any constrained internals whose target values change with lambda lie within a closed loop. The loop can either be part of the molecular topology, *or as a result of the added topology of the constraint(s)*. To understand why neither FEP nor TI with "potential forces" can be used in such a case, you must recognize that for these latter methods, part of the procedure for calculating constraint contributions requires that we determine which atoms of the system are affected by a rigid body translation/rotation about the constrained internals. But the requisite set of atoms is not unambiguously defined when the constraint lies within a closed loop. Fortunately, the "constraint force" implementation of TI doesn't require that we make such a determination.

It is important to note that PMF calculations are typically very compute-intensive. For FEP, Gibbs will determine which non-bonded pairs have an interatomic distance which varies with one or more constraints, and only these are re-evaluated in determining $V_{\lambda(i+1)}$. This helps reduce the amount of computer time required for a FEP simulation, although the total amount of time can remain high. The additional cpu overhead for calculating constraint energies with TI is negligible in all cases.

While we have a good error check for some torsional PMF's (the free energy values after rotating 360° should be the same), we typically have no reliable way of determining that for other simulations enough sampling has carried out to determine a converged PMF curve. Our best guard against spurious results is careful consideration of the specific problem and the inherent relaxation timescales of the surroundings.

7.8.6. Error estimates and convergence

One of the thorniest issues related to free energy calculations is estimating the error in the results. This error is often estimated in one of four ways:

(1) Two separate free energy simulations can be run, one with λ progressing from $0 \rightarrow 1$, the second with λ progressing from $1 \rightarrow 0$. These two calculations should yield the same free energy value, and the difference between them (the "hysteresis") gives a lower bound on the estimate in the calculation. Errors derived in this way often underestimate the actual error.

(2) The difference between "forward" and "backward" values for a single run. As described in the introduction, when FEP or slow growth is performed, double-wide sampling can be carried out. This ultimately results in two pseudo-independent values for the free energy, one calculated from the sum of all the $\lambda(i) \rightarrow \lambda(i+1)$ energies, and the other calculated from the sum of all the $\lambda(i) \rightarrow \lambda(i-1)$. If the results were exact, these values would be the same. In practice, they will not be, and their difference gives a crude lower bound on the inherent error. Error estimates derived in this manner tend to be even less reliable than those estimated using method (1), and are usually worthless for slow growth type runs.

(3) Two or more simulations are run under equivalent but different conditions. For example, starting with different randomly assigned sets of velocities. The difference between the free energies provide an estimate to inherent errors. These estimates are subject to the same problems as (1) above.

(4) A series of simulations is run which differ in the respective amounts of sampling done. For example, simulation 1 might use 100 steps of equilibration and 100 steps of data collection at each window, while simulation 2 used 200 steps of each. If the value from the shorter simulation was accurate, the value from the second simulation should be acceptably close to it. If it is not, the simulation must be run even longer to confirm convergence. This method probably provides the best insurance that convergence has been reached, but it is not definitive, and it is also the most costly.

It must be understood that none of the above methods allows a completely reliable error estimate. At best, they provide a *lower bound* on the error. A large apparent error is a good indication that the results obtained are not appropriately converged. But a low apparent error does not necessarily indicate a converged and accurate simulation.

7.8.7. Changing parameters versus dual topologies

In "standard" operation, free energy changes in GIBBS are effected by transforming the potential representative of state 1 to that representative of state 2. The topology of the system does not change. To make atoms non-interacting at one of the endpoints, they are assigned zeroed non-bond and electrostatic parameters at this endpoint.

The extended mixing rules which can be used in GIBBS (IOLEPS = 0, line 10) allow a second method to be used. One result of these new mixing rules is that if any pair of atoms "exist" only at mutually exclusive endpoints (e.g. atom i exists in state 1 but not state 2; atom j exists in state 2, but not in state 1), then effectively no non-bonded interactions are ever calculated between them. This means that, in lieu of the "standard" method which uses a single topology, we can define dual topologies, one corresponding to the $\lambda = 0$ endpoint, and the other corresponding to the $\lambda = 1$ endpoint. For the former topology, all non-bonded parameters would be defined to be 0 in the $\lambda = 1$ state. Similarly, all non-bonded parameters for the latter topology would be 0 at $\lambda = 0$. The two topologies would then never "see" each other at intermediate values of λ. Defining dual topologies can aid in performing free energy calculations where bond connectivities must change. Dual topologies is the method incorporated into the "CHARMM" program.

On an efficiency basis, the relative merits of the two methods have not been established. Additional discussion of the two methods can be found elsewhere [86].

8. ptraj

The current version of `ptraj` (previously called `rdparm`) 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). For more information about `rdparm`, see either the supplied HTML file "rdparm.html" or the section on `rdparm` that follows the discussion of the `ptraj` commands.

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 it 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-orthorhomic 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
```

correlation *name mask1 mask2 tstep tcorr tmax* out *filename*

This command will compute the correlation functions of an internuclear vector from the center of mass of atoms in *mask1* to the center of mass of atoms in *mask2* based on three parameters which must be specified: *tstepfR, which (because of a bug) should always be 1; tcorr*, the maximum snapshot for which correlation functions are to be computed; and *tmax*, the maximum snapshot in the simulation to be used. Note that *tcorr* and *tmax* are integers that correpond to snapshot numbers; users must know the actual separation between frames to convert a "snapshot-number" to an actual time. The output will be stored in a file named *filename* following the keyword out and can be directly sent to various plotting programs. This command replaces the 3-step process of mdovrly, mdextract and mdcorrp2 that was used in earlier versions of Amber.

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 travelled 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-orthorhomic unit cells) and the shortest imaged distance will be saved (unless the "noimage" keyword is specified which disables imaging).

grid *filename nx x_spacing ny y_spacing nz z_spacing mask1* [origin | box] [negative] [max *fraction*]

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-orthorhomic 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 recommmended) since each of the possible imaged distances (27) are checked to see which is closest to the center.

principal *mask*

Principal axis align the atoms specified in *mask*. This is more or less functional as there are issues with degenerate eigenvalues and swapping, however it basically works...

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, <i>JACS</i> 94, 8205-8212 (1972) or Harvey & Prabhakaran, <i>JACS</i> 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, <i>JACS</i> 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 psueodorotation 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).

vector *name mask* {principal [x | y | z] | dipole | *mask2* | box} [out *filename*]

This command will keep track of a vector value (and it's origin) over the trajectory; the data can be referenced for later use based on the *name* (which must be unique among the vector specifications). If the optional keyword "out" is specified, 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 normalized, nor converted to appropriate units, nor is the value likely defined if the atoms in the mask are not overall charge neutral.

mask2: store the vector between the center of mass of the atoms in *mask* with the center of mass of the atoms in the additional mask (*mask2*). The center of mass of the first mask is the reference point.

box: store the box coordinates of the trajectory. The reference point is the origin or (0.0, 0.0, 0.0).

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. hydrogen bonding facility

ptraj 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 interations 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.

8.6. 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
         printExluded
         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 handing.  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         --- this 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 documention.

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".

printExluded	Print the excluded atom list.
printLennardJones	Print out the Lennard-Jones parameters.
printTypes	Print out the atom types.
quit	Quit the program.

9. Carnal

Usage:

```
carnal [-O] < analin > analout
carnal [-O] -i analin -o analout -p parm
```

−O Overwrite output files.

CARNAL is a coordinate analysis program that uses a flexible command language. In addition to conventional trajectory measurements, it allows comparisons between multiple streams of coordinates. It has many of the capabilities of ANAL and MDANAL, which it was intended to eventually replace. It also provides set-theoretic group specification, cartesian vector oriented measurements, hbond analysis, output of distributions (including radial), selection of coordinate sets from streams, interpretations of md streams in terms of windows, and format conversion.

```
_____________________________________________________________________
|                                                                   |
|                               Stage 1               Stage 2        |
|                               (CARNAL)            (You supply)     |
|                                                                   |
|  ___________         stream(s)             tables,               |
|  | Dynamics |----->|          ___________   coords               |
|  -----------       |--------->| static & |                       |
|                    |- - - - ->| dynamic  |---------> numerical    |
|  ______________    |- - - - ->| analysis |           analysis     |
|  | Minimization|-->|         /-----------           & display    |
|  --------------            /   ^                                  |
|                           /    |                                  |
|             parm file(s)/     |                                  |
|                                commands                           |
|                                (analin)                           |
|___________________________________________________________________|
```

9.1. Introduction

CARNAL input is essentially a programming language that lets one specify variables and perform operations on them. The control input file for CARNAL is referred to as *analin*. The commands in this file name the other input and output files as well as the measurements to be performed. These commands are described in detail in the syntax specification and examples below. The −o file or standard output contains carnal's interpretation of the analin input and summary data for the run.

Note that periodic boundary conditions are only applied to DISTRIBUTION DIST measurements. This is because CARNAL allows measurements across multiple coordinate sets, but could be enabled in the normal case of measuring within a single set.

9.1.1. Input

CARNAL takes as input one or more "streams" of input coordinates, which are lists of restart, mdcrd, or Amber pdb format files. It detects formats transparently. The formats can be mixed in a stream, but the files must have the same number of atoms in the same order defined by the parm file for that stream. Each stream may have a different parm file. The −p argument or the first parm file defined in analin is used as the default parm file if no parm file is specified for a stream. At least one stream must be specified; the first one is used as the default when a stream reference is expected. CARNAL will also load static coordinate sets for comparison with the individual sets in the streams. NOTE: periodic boundary conditions are only applied to DISTRIBUTION DIST measurements; this is because of difficulties in measuring across 2 streams, but could (and at some point will) be enabled in the normal case of measuring within a stream. Compressed input files ending in .Z or .gz are uncompressed transparently, allowing disk space to be saved (the disk version remains compressed).

9.1.2. Output

CARNAL writes as output tables of measurements (scalar measurements or 0/1 hydrogen bond occupancies), distributions (including radial) and coordinate sets in mdcrd, restrt or pdb format. Summary data is also written to the file named with the −o argument on the command line, or standard output if no −o argument is given.

9.2. Analin introduction

This introduction is intended to give the feel of the analin language via an overview of the syntax and a simple example. A complete syntax definition and more complex examples are given below.

9.2.1. Summary of Analin Sections

These are the required sections in the analin input file syntax. Comments follow '!'s. In actual analin files, a '#' at the beginning of a line turns it into a comment. There are 4 main sections, each begun by a keyword in this order:

```
FILES_IN            ! name parm, coord/restrt/pdb files
FILES_OUT           ! name output tables, coord/restrt/pdb dumps
DECLARE             ! describe items to be measured
OUTPUT              ! direct declared stuff to output files
END                 ! end input, start execution: STOP may be
                    ! substituted for debugging: program stops
```

Things are declared in the first 3 sections and referenced in the last 2 sections. When something is declared it is given an id for referencing it later.

9.2.2. A Simple Analin Example

Select some coord sets from mdcrd files and output them in pdb format:

```
FILES_IN
  PARM   p1 ketop;           ! keyword, id, filename
  STREAM s1 kecrd kfcrd;     ! keyword, id, 2 filenames
```

```
FILES_OUT
  COORD  c1 /tmp/ke.p PDB;   ! keyword, id, filename, output format
DECLARE
                             ! no declarations for this simple case
OUTPUT
  COORD  c1 s1 SELECT (1 3 5 200);
                             ! keyword, files_out id, files_in id:
                             ! command to select sets 1, 3, 5, 200
END
```

In this case, coord sets 1, 3, 5, and 200 from the concatenated stream of mdcoord files kecrd and kfcrd will be output to pdb files /tmp/ke.p.1 /tmp/ke.p.3 /tmp/ke.p.5 /tmp/ke.p.200.

9.3. Analin Syntax Specification

Notes

Aspects to be changed are indicated by ’CHANGE:’. Definitions must precede references: you cannot refer to something defined later in the file. Characters reserved for explicit purposes are:

```
# - . % ( ) & | , ! ?
```

A ’#’ as the first character of a line makes the line a comment. The format is entirely free, i.e. statements can be spread across multiple lines with any indentation and with comment lines embedded. Lines may not exceed 80 characters − re-read the preceding sentence if you think that causes a problem. In the syntax below, items in [] brackets are optional and items within {} braces are descriptions rather than token-by-token matchings.

---FIRST SECTION = "FILES_IN"

Input coordinates may be MD crd dump, inpcrd/restrt or Amber output pdb format.

FILES_IN

PARM id filename;

Amber Parm file. Multiple parms can be defined; the 1st one (or one defined by the [-p parm] argument) becomes the default for STREAMs that don’t specify a parm.

STREAM strid [parmid]

[NOBOX] [ATOM n] [WIN x y] file1 file2 ... ;

At least 1 STREAM must be specified. STREAM files are read sequentially at each step. If > 1 STREAMs are named, they can be compared at each step. If no parmid is given, the first one defined is used by default. The NOBOX and ATOM options allow CARNAL to handle certain discrepancies between the parm topology and the input stream. If these options are inaccurate, synchronization may be lost, resulting in garbage. ATOM is for reading in a stream that has *fewer atoms that the prmtop* - such a stream might have been created earlier using the ATOM option in the COORD section. All coordinate sets in a stream must have the same number of atoms.

NOBOX

Mdcrd files for periodic simulations have box coordinates after each coordinate set. Carnal automatically detects the presence of periodic conditions from the parm topology and allows for reading the box coordinates in mdcrd. However, minimization restrt files (as well as constant volume mdcrd files previous to 4.1) do not include the box coordinates. The NOBOX option allows carnal to read min.rst and old constant volume mdcrd files correctly.

ATOM n

Read only n atoms (more may be defined in the parm file). I.e. coordinates for only n atoms are in the stream. Implies NOBOX, *i.e.* if a box is specified in prmtop, it is ignored.

WIN

Means, "skip x sets, use y sets" repeatedly. This is for analysis of periodic equilibration / data collection runs such as gibbs.

STATIC statid [parmid]
[NOBOX] [ATOM n] file1 file2 ... ;

STATIC files are read at the beginning and remain in memory for comparison with STREAM coordinates. Each static set in an id can be referenced by 'id%1', 'id%2' etc. See STREAM above for "NOBOX" and "ATOM n" descriptions.

---SECOND SECTION = "FILES_OUT"

FILES_OUT

TABLE tabid filename ;

In the tables, there is one "logical row" per coordinate set measured, so a given measurement over a trajectory occupies a column. For example, the Nth item directed to the table (in the OUTPUT section, below) would form the Nth numerical column and the Ith measurement of that value would be in the Ith logical row. The logical rows are wrapped so that a row continues through a series of lines in a single file beginning with keys of columns by grepping for the key letter. If there is demand, rows can instead be spread across multiple files (filename.0 filename.1 ...) or just tabbed continuously within a file (harder to read visually). Thus, the format is:

```
L0   m1    m2    m3    m4    ...   |
L1   m11   m12   m13   m14   ...   |   1st coord set
L2   m21   m22                     |
L0   m1    m2    m3    m4    ...    |
L1   m11   m12   m13   m14   ...    |  2nd coord set
L2   m21   m22                      |
```

where the measurements are m1, m2, ... m22 extending over a logical line consisting of lines 'L0', 'L1', and 'L2'.

COORD crdid file [APPEND] [BLANK] [format] ;

APPEND

Add to end of named file if it exists.

BLANK

Write a blank line after each set.

Format symbols are 'PDB' 'RST' and 'CRD'. The default format is CRD.

HBOND hbid base_file [TABLE] [LIST];

TABLE

Output table of occupancies in base_file.tab. This table has two sections. The first part is a key that lists the possible hbonds in order. The format is:

```
# 1  (ADE  2 O5′ )--(HB   1 H   ) .. (ADE  2 O1′ )
# 2  (ADE  2 O5′ )--(HB   1 H   ) .. (ADE  2 N7  )
```

The second part consists of a matrix of 0's and 1's. Each column is for a given hbond pair according to the numbering in the key section, and each row (line) is for a coordinate set. The format is '0' if no hbond is happening, '1' if it is. The Unix 'awk' utility can be used to extract column(s) of interest for further occupancy analysis or plotting, e.g. "egrep -v '^#' | awk '{print $2, $5, $8}' base_file.tab" where the 'egrep' command strips out the key section and the 'awk' command selects the columns of interest. Note that if there are too many columns for awk to handle, the 'perl' utility may be needed.

LIST

Output list of per-hbond-per-set to base_file.lis for extensive analysis. The format is:

```
1 ( ADE  2 N6   )( THY  5 O4   ) 2.930768 9.125721
2 ( ADE  2 N6   )( THY  5 O4   ) 2.957820 3.151730
```

where the 1st number is the number of the coordinate set (starting with 1), followed by donor, acceptor, distance and angle (in radians). Atoms are specified by residue name, residue number, and atom name. (See OUTPUT HBOND STAT for summary hbond info, including fraction of occupancy.)

HBOND specifications are given in the OUTPUT section.
Summary info is printed to standard output.
CHANGE someday: at least one of {TABLE, LIST} must be given, and the OUTPUT HBOND statement is req'd to do any hbond analysis.

DISTRIBUTION dbid filename [DAP][MIN];

DAP

Put number of intervals on 1st line.

MIN

For DIST option, below. For each ’solvent’ atom, write out min distance to ’solute’ for the run (multiple lists separated by a ’%’ are output if WIN is chosen with DIST). This is to figure out which waters to keep in a second pass dumping COORDs. List output goes to filename.min.

The definition of contents of the file is described in the OUTPUT section.

---THIRD SECTION = "DECLARE"

Each object is bound to a crd set; if not bound explicitly, the default is stream 0. References to that object inherit the binding of the object, except for within a GROUP statement (GROUP (GROUP id)...). I.e. in general an optional [crdset] is not allowed after a declared id is used (binding at reference rather than creation), for now.

"Points" can be atoms (specified by "number [crdset]" or "atom_name residue_number [crdset]") or centers of geometry or mass of groups of atoms (see GROUP definition below). For example, "PLANE id 12 34 58;" specifies the plane formed by atoms 12, 34 and 58 in the default stream, and "PLANE id OD1 2 ND1 4 OD1 6;" specifies the plane formed by atom name OD1 in residue 2, etc. "PLANE id gid1 gid2 gid3;" specifies the plane formed by the centers of geometry of three groups.

DECLARE

----Group is defined by set theoretic operations. Group attributes include center of geometry or mass, moment of inertia, and radius of gyration.

GROUP id [crdset] (((set OP set) OP set) ...) ;

The group is defined on the default stream unless [crdset] is given. The nesting in parentheses determines the order of evaluation.

OP can be either ’&’ or ’|’
where ’&’ = intersection, ’|’ = union
and set can be any one of: { (ATOM numlist),
(ATOM [NAME|TYPE] namelist),
(RES numlist),
(RES NAME namelist),
(SOLUTE)
(GROUP namelist_of_groupids),
!set }

In the (GROUP) set, the groups are OR’d together. The NAME and TYPE options allow the use of ’?’ as a wild card matching any single character.

Allowing expressions:

groupid%center

center of geometry - default if groupid is used as a point

groupid%cmass

center of mass

groupid%momin
moment of inertia
groupid%radgyr
radius of gyration

For example, groupid%radgyr could be included in an OUTPUT / TABLE list (see below) and thus form a column of a table.

CUTRES id x y z cut;
CUTRES id groupid cut;

CUTRES produces a list in AMBER GROUP format (Appendix B) of all residues with any atom within *cut* of the given center point, or if a *groupid* is given, within *cut* of the group, including the residues in the group itself. It is intended primarily for analyzing a single coordinate set to generate a group of atoms within an area of interest that will be allowed to move in a belly dynamics or perturbation simulation.

AXIS id {2 points} ;

AXIS axid1 1 2 ;
Atoms 1 -> 2 in stream 0 by default.
AXIS axid1 1 st1id 2 st2id ;
Atom 1 in stream/static st1 -> atom 2 in stream/static st2.
AXIS axid2 grp1id grp2id%cmass;
grp1 center of geometry to grp2 center of mass: note that the groupids may be the same or groups may be defined on different streams.

PLANE plid { 3 points or 2 axes } ;
A plane is treated as its normal vector where appropriate.

ANGLE angid { 3 points or 2 axes/planes };
Planes are interpreted as normal vectors.

TORSION tid { 4 points or 3 axes/planes };
Planes are interpreted as normal vectors. Note that in the averages printed to standard output, carnal section).

TORSION tid BACKBONE [residue1 [residue2]] [crdset] ;
Find all torsions involving backbone bonds (between Amber main chain atoms), starting with residue1 (default: 1st residue) and ending with residue2 (default: last residue). If first and last residues' terminal backbone atoms are bonded to each other, torsions involving them are included.

DIST dsid { 2 of: points, axes, planes };
Select 2 points, 2 axes, or point and axis or plane. [planes and axes are not supported yet]

IMAGE imageid groupid ;
IMAGE imageid groupid%cmass ;

Only for periodic systems. Place the system so that groupid (center of geometry or center of mass) is at the center of the box, and image all residues accordingly. Cleans up Ewald runs. Uses groupid's stream. Imageid can be used as a streamid in measurements. NOTE: this is not guaranteed to give the desired, intact system on the first try − it depends on the transformations that Ewald has made and the size of the box. For example, the center of geometry of a DNA duplex could be in the center of the box, but the strands could be on the edges. Successive transformations using trial and error may be required to restore an Ewald trajectory to normal appearance, and different transformations may be required for different frames. Also, intact molecules can be broken up by this option if centering makes some residues project out of the box (possibly an indication that the box size used was too small).

RMS rmsid [FIT] groupid ;
RMS rmsid [FIT] groupid streamid ;
RMS rmsid [FIT] groupid streamid refcrdid ;
RMS rmsid [FIT] groupid staticid [ATOM] [RES];
RMS rmsid groupid2 prevrmsid ;

Using atoms in groupid, measure rms of one coordinate set to another. If FIT is selected, the current coordinate set is first positioned for minimum mass-weighted rms of the group on the reference coordinates; this also allows the rmsid to be used to determine the resulting (non-fitted) RMS measurement on other groups (as in the prevrmsid case above), and the FITted rmsid can be used like a stream for other measurements, as well as output via a COORD statement. Per-residue and per-atom RMS values within the group can be output in a TABLE by rmsid%residues or rmsid%atoms.

The first and simplest case above uses groupid to compare the default stream to its first set. The second case compares a named stream (rather than the default) to its first set. The third case specifies both the stream and a reference set for comparison; this reference set could be a static (single) set or another stream (comparing successive sets in each stream).

The fourth example, using a STATIC id, produces a (triangular) matrix of RMS values, one for each pair of coordinate sets in the STATIC id. The ATOM and RES options additionally cause per-atom and per-residue RMS to be reported. All output is calculated and printed immediately, since it does not depend on reading a serial stream. The matrix format for a 3-coordinate-set STATIC would be:

```
---RMS MATRIX
set0 set1 value0
set0 set2 value1
set1 set2 value2
---
```

NOTE: use of FIT with this option leaves all coordinate sets in the STATIC set with the group center of mass placed at the origin and each successive coordinate set rotated to fit its predecessor.

The final case measures the rms of a second group on a pair of sets that were positioned by a previous RMS FIT statement. See OUTPUT TABLE for instructions on obtaining per-residue and per-atom rms for streams. Any number of any of these types of RMS measurements can be used.

DME dmeid groupid ;
DME dmeid groupid streamid ;
DME dmeid groupid streamid refcrdid ;
DME dmeid groupid staticid;

Using atoms in groupid, measure Distance Matrix Error between one coordinate set and another. DME compares intra-group distances in one conformation to those in another conformation.

$$DME = sqrt(2/(N*(N-1)) * \frac{\Sigma}{atom\ pairs}(distance - refdistance))$$

In proteins, the convention is to use DME for groups defined on C-*alpha* atoms. The first and simplest case above uses groupid to compare the default stream to its first set. The second case compares a named stream (rather than the default) to its first set. The third case specifies both the stream and a reference set for comparison; this reference set could be a static set or another stream. The final example produces a (triangular) matrix of DME values, one for each pair of coordinate sets in the STATIC id. The format for a 3-coordinate-set STATIC would be:

```
---DME MATRIX
set0 set1 value0
set0 set2 value1
set1 set2 value2
---
```

PUCKER pukid NUCLEIC [streamid] [residue_names|residue_numbers] ;
PUCKER pukid number_of_points points ;

Measure pucker using algorithm of D. Cremer and J. A. Pople (JACS 96:6 pp 1354-1358, 1975). For the NUCLEIC options, the Altona and Sundaralingham convention (JACS 94 pp 8205-8212, 1972 or p. 20 of Saenger’s "Principles of Nucleic Acid Structure", Springer-Verlag, 1983) is approximated by adding 90 degrees to the phase angle, the puckers are always ordered according to residue order in the parm file, and the standard atom names (O4’/O1’, C1’, C2’, C3’, C4’) are used to determine the points. (For a comparison of different nucleic acid pucker conventions, see S. C. Harvey and M. Prabhakaran, JACS 108:20, pp 6128-6136, 1986.) In the general case (specifying points explicitly), a ring of N points can be parameterized by N-3 alternating amplitudes and phase angles. Note that the Cremer/Pople algorithm finds a mean plane based on the assumption that successive points in the ring have the same angle between them with respect to the center of geometry, so for kinky rings this may not work. Note that in the averages for angles printed to standard output, carnal section).

PUCKER pukid NUCLEIC;

Measure pucker of all standard residues (’94 force field: G5, G, G3, GN, *etc.*; ’91 force field: GUA, *etc.*) in default stream using standard atom names (O4’/O1’, C1’, C2’, C3’, C4’).

PUCKER pukid NUCLEIC streamid;

Same as above, but uses specific stream rather than default.

PUCKER pukid NUCLEIC GUA;
Measure pucker of all residues named 'GUA' using standard atom names.

PUCKER pukid NUCLEIC 2,4,6,8 ;
Measure pucker of residues 2,4,6, and 8 using standard atom names.

PUCKER pukid 5 O1' 2 C1' 2 C2' 2 C3' 2 C4' 2 ;
Measure pucker of 5 points: O1' (residue 2), C1' (residue 2), etc.

---FOURTH SECTION = "OUTPUT"

OUTPUT

TABLE tbid { column_list } ;

At least one column must be specified. Columns are printed in their order in the list. Column_list may include ids, classes of measurement (e.g. DIST) which print in the order declared, MEAS which prints all scalar measurements, or ALL which prints everything. AXIS ids result in vectors, PLANE ids in normals, and GROUP ids default to center of geometry unless attributes are specified, such as grpid%cmass and grpid%radgyr. RMS ids default to the rms of the group, while rmsid%residues and rmsid%atoms give per-residue and per-atom rms respectively. For per-residue rms, the group must not have any partially-included residues. If either per-residue or per-atom options are used, the statistics are printed in the summary with the residue and atom names.

COORD crdid streamid
[SELECT (-i j k,l m-p q-)] [MOD h]
[AVERAGE] [ATOM n] [EXH2O m [GROUP gid]] [INH2O gid] ;

SELECT (-5 7 8,10 100-105 200-)

Select certain sets from the stream by order. Numbers are separated by spaces or commas, and '-' is used to indicate ranges. In this example, select sets 1 through 5, then sets 7, 8, and 10, then sets 100 through 105, then sets 200 through the end. Note that this option selects files for output only, and does not affect measurements on the stream, as opposed to the STREAM WIN option, which pre-selects sets for all the other commands.

MOD h

Select every h'th set from the stream. Note that this option selects files for output only, and does not affect measurements on the stream, as opposed to the STREAM WIN option, which pre-selects sets for all the other commands.

AVERAGE

Average the coordinates. Not compatible with EXH2O or INH2O, but ok with ATOM. Produces a single set, so PDB or RST format is advised for the corresponding FILES_OUT COORD declaration. Suggested that this be applied to an RMS FIT streamid so that the area of interest has minimal distortion from drift or pressure scaling of the box. Note: The averaged coordinates may be a chemically unrealistic hybrid of different regions of phase space, so visual inspection, energy analysis, and perhaps energy minimization may be in order, depending on the purpose. As a simple example, the methane C-H bond length shortens due to H rotation. Note 2: a measurement on averaged coordinates is not the same as the average of the same measurement over the same trajectory.

ATOM n

Output only the first n atoms. ATOM, GROUP, INH2O and EXH2O are mutually exclusive options.

GROUP id

Output only atoms in the group. ATOM, GROUP, INH2O and EXH2O are mutually exclusive options.

EXH2O m [GROUP gid]

Omit all but m waters from the set, retaining either those closest to the non-waters, or those closest to the atoms specified by GROUP. Distance is measured from water oxygen. Waters are printed in order of closeness to the solute, i.e. the order varies from set to set, so identity-based dynamic graphics smoothing schemes will fail. ATOM, GROUP, INH2O and EXH2O are mutually exclusive options.

INH2O gid

Omit all waters from the set except those with atom type OW in group gid, where gid contains only OWs. This group is intended to be built with the output of a previous pass using DISTRIBUTION MIN which informs the user how close each water came to the area of interest during the run. See example below. ATOM, GROUP, INH2O and EXH2O are mutually exclusive options.

HBOND hbid [DONOR [EXACT] g1] [ACCEPTOR [EXACT] g2]
[DISTANCE x] [ANGLE y] [STATS];

DONOR and ACCEPTOR indicate group ids for searching for the appropriate atoms (note: donor is the heavy atom to which the hydrogen is attached). The default for either is all atoms. A single group id may be given in place of separate definitions. If DONOR and/or ACCEPTOR are specified, the EXACT option forces carnal to use all heavy atoms that may apply, instead of just the ’classic’ ones such as oxygen and nitrogen.

DISTANCE

Cutoff distance in angstroms between the heavy atoms: default is 4.0.

ANGLE

Cutoff H-donor-acceptor angle in degrees (0 is linear): default 1 radian ˜= 60 degrees.

STATS

Directs printing of per-hbond summaries to standard output. The format is:

```
HBOND h1 stats:
  # 19 (ADE  2 N6  )_(ADE  2 HN6A)..(THY  5 O4  ) % 64.400000
       distance    avg: 2.909575  max 2.961379   min 0.000000
       angle(deg)  avg: 7.241544  max 15.219878  min 0.000000
```

where the # refers to the column of file.tab and the ’64.400000’ gives the percentage of occupancy. The other statistics are only for the "occupied" states. The distance is between donor and acceptor atoms.

DISTRIBUTION dsid
{ RAW | min max nboxes [WIN nsets] }
{ measid | DIST group1 [group2 [ALL]] [NORM] } ;

Distribution output can be either RAW (a long line of ascii floating point numbers per coordinate set) or binned and normalized. If the latter, the WINdow option causes the distribution for each nsets supplied by the STREAM to be written, with a ’%’ line to separate each window.

RAW is the recommended option for large data sets, since the proper range and number of bins are hard to guess at: the rdis program can be used on the raw output to quickly try various min/max/nboxes numbers on the raw file. Note, however, that measurements including many terms can generate files larger than the original trajectory, so the RAW option may not be appropriate in such cases. For example, when measuring O-O distributions in a system of N waters, there are 9N numbers per coordinate set, but N(N-1)/2 distances to write out if RAW is used. For N=1000 this amounts to a RAW file 55 times larger than the trajectory.

Either an id for a scalar measurement or a radial distance distribution (DIST) may be specified. In the latter case, one or two groups can be specified. A single group may be given, in which case all intra-group distances are used (this would be appropriate for e.g. water O-O); otherwise, two groups are required. When just two groups are given (without the ALL option), the groups are treated as "solute" and "solvent" respectively: for each "solvent" atom, the distance to the closest "solute" atom is applied. The ALL option includes all group1-group2 distances rather than the "solvent" to closest "solute" atom. Note that when two groups overlap, distances of 0 would be obtained for the atoms that are in both groups, so groups should be disjunct. The MIN option from the FILES_OUT section above is only valid for the plain, two-group mode.

Volumetric NORMalization is optional when radial DISTribution is selected. This only makes sense for measuring the distribution around a single atom or a set of chemically identical atoms, since the normalization is done by dividing the count for each interval by the volume of a spherical shell having radii equal to the shell boundaries. (To normalize the distribution around a functional group, for example, would require calculating the volumes of the non-spherical shells around the group.)

The output format is: "bin_center value smoothed_value integral," where bin_center is the center of the interval of the bin (the first is min + 0.5 * (max-min)/nboxes), the value is the distribution for that bin, and the integral is the cumulative sum at the current bin. The integral is not affected by the NORM option.

9.4. Examples

9.4.1. Simple coordinate averaging

```
#Simple Coordinate Averaging
FILES_IN
  PARM   p1 ketop;           ! keyword, id, filename
  STREAM s1 kecrd kfcrd;     ! keyword, id, 2 filenames
FILES_OUT
  COORD  c1 /tmp/ke.p PDB;   ! keyword, id, filename, output format
DECLARE
                             ! no declarations for this simple case
OUTPUT
  COORD  c1 s1 AVERAGE;
                             ! keyword, files_out id, files_in id:
                             ! command to average sets
END
```

9.4.2. Simple distance, angle, and torsion measurements

```
# Plain measurements involving points (atoms, centers of mass)
FILES_IN
  PARM p1 prm.top;
  STREAM s1 a1.trj a2.trj a3.trj;
FILES_OUT
  TABLE tab1 meas.tab;
DECLARE
#
#    First, some measurements using atoms only: format is:
#       FUNC_NAME ID atom_specs ;
#    each atom_spec can be either:
#       ATNAME RESNUM
#    or
#       ATNUM
#
  DIST dist1 O1’ 2 O1’ 7;
  ANGLE ang1 2 12 13;
  TORSION tor1 C1 4 C2 4 C3 4 C4 4;
#
#    A special case for torsions:
#
  TORSION tor2 BACKBONE;
#              ˆ find all torsions consisting completely of
#                main chain atoms
#
#    Now some more geometrical stuff: the angle between
#    the normal vectors of two planes:
#
```

```
  PLANE pla1  C1 4 C2 4 C3 4;
  PLANE pla2  C1 5 C2 5 C3 5;
  ANGLE ang2  pla1 pla2;
#
#    Now some measurements using composite points:
#    format is the same, except for atom_specs.
#    First we’ll define a group consisting of the non-waters,
#    and a group consisting of atoms in 3 numerically adjacent
#    residues:
#
  GROUP g1 (SOLUTE);
  GROUP g2 (RES 5,6,7);
#
#    And now to measure the distance between the centers of
#    mass of each group to see how that pair of residues
#    fluctuates from the center:
#
  DIST dist2 g1%cmass g2%cmass ;
#
#    Now let’s define 2 more residue-based groups:
#
  GROUP g3 (RES 21,22,23);
  GROUP g4 (RES 44,45,46);
#
#    And we’ll measure the angular fluctuations of the 3
#    residue-based centers of mass:
  ANGLE ang3 g2%cmass g3%cmass g4%cmass ;

OUTPUT
#    Now direct all the measurements to the table defined above
#    MEAS refers to all scalar measurements; each 1 will be a
#    column in the order defined (dist1 ang1 tor1 tor2 dist2 ang2).
#    Alternatively, the ids could be given explicitly in any order.
#
  TABLE tab1 MEAS;
END
```

9.4.3. RMS deviation

You want a measurement of the minimum RMS deviation of a group of atoms as a measure of how disordered some structures are relative to another structure. Given the best fit on that group of atoms, you also want to know how much another subgroup differs and how much all the atoms outside the fit group differ. You also want to save the fit structures for viewing.

```
#
# RMS example: fit central 8 bases of a G4 DNA quadruplex
#
FILES_IN
  PARM p1 p524.top;
  STREAM s1
    sm4.pdb
    sm9.pdb
    sm17.pdb;
  STATIC ref_set  sm3.rst;
#                 ^^^^^^^ this file will be used as one
#                         reference set for the comparison;
#                         no pdb file around, but that’s ok
FILES_OUT
  TABLE tbl  sm.rms;
#            ^^^^^^ this is a table for the per-set rms values
  COORD crd  fit.p  PDB;
#                   ^^^ save some structure(s) in PDB format
#            ^^^^^ name of file
DECLARE
#
#    Now let’s get down to business.
#    Declare a group of atoms to fit on - all the
#    non-sugars in the central 8 GUAs:
#
  GROUP grp1 (  (ATOM NAME N9 C8 H8 N7 C5 C6 O6 N1 H1
                        C2 N2 HN2A HN2B N3 C4)
              & (RES 4,6, 13,15, 22,24, 31,33)  );
#             ^ boolean for "all of the atom names in
#               the 1st part that occur in the following
#               residue numbers"
#
#    RMS fit the structures in the stream to the reference
#    set using the group of atoms just defined (fitting is
#    mass-weighted). This creates a new thing that can be
#    treated as a stream.
#
  RMS fit1  FIT  grp1  s1 ref_set;
#                         ^^^^^^^ reference structure id -
#                                 if not given, the first
#                                 structure in the stream
#                                 would be used; or this id
#                                 could be for a different
#                                 stream instead of the static
#                                 coord set used here
#                      ^^ stream id - what to fit
#                ^^^ group of atoms to use for fitting
#           ^^^ fit (position) the stream set to minimize rms
#     ^^^ ’fit1’ is the new, streamlike thing
#
#    Specify the group of atoms to measure a secondary
#    deviation - the terminal bases on each strand, w/out
#    the sugars:
#
```

```
  GROUP g2 ((RES 2,8, 11,17, 20,26, 29,35) &
        (ATOM NAME N9 C8 H8 N7 C5 C6 O6 N1 H1
               C2 N2 HN2A HN2B N3 C4));
#
#    Measure the RMS on that group resulting from the fit
#    of the other bases, i.e. between the target structure
#    and the new, fitted structure. Just measuring this
#    time, not creating a new set.
#
  RMS fit2  g2  fit1  ref_set;
#
#    Let’s see what that fit does for _all_ the atoms outside
#    of the fit, not just the end bases.
#    Specify the group of all atoms not in the original group
#    used for the fitting:
#
  GROUP g3 (!grp1);
#
#    Measure the RMS of that group on the fitted structures
#    as before.
#
  RMS fit3  g3  fit1  ref_set;
#
#    Just for fun, create a new fitted set using the 1st group
#    but using the 1st set in the stream as reference.
#
  RMS fit4  FIT  grp1  s1;
#                      ^^ just specifying the stream with no
#                         reference defaults the reference to
#                         the 1st set in the stream
#                ^^^^ use our "central bases" group again
#           ^^^ FIT the thing
#     ^^^^ name of a new, stream-like thing starting with the
#          second crd set in the stream
#
OUTPUT
#
#    Write the RMS values to the table:
#
  TABLE tbl  fit1  fit2  fit3  fit4;
#            ^^^^  ^^^^  ^^^^  ^^^^ output the measured rms
#                                   values as columns in the
#                                   table declared as a file
#                                   above
#       ^^^ to table ’tbl’
#
#    Average the the structures fitted using the 1st group
#    on the reference set and print them to the coordinate
#    file defined above. Perhaps we will then energy min
#    this structure and claim it means something.
#
  COORD c1  fit1  AVERAGE;
#
END
```

9.4.4. Coordinate selection: waters

You want coordinate dump of solute with selected waters. Not the closest waters at each step: you *know_exactly* which waters you want: the same ones in every set, maybe so that when you smooth the trajectory, the Nth water won’t change its identity at each step. This is a 2-pass procedure: first you need to get a list of the waters: you need the atom number of the OW (atom type) in each water. If you want those waters to be all those that came within a given distance of the solute, have I got an option for you. The thing to have is, for each water, the closest it came to whatever you want to call the solute. DISTRIBUTION MIN will give you a list of atom number, distance pairs that you can sort to generate the list of favored waters you need. With this list of OW atom numbers, you can define a group which, in another pass, can be used to filter waters.

```
# use of INH2O with DISTRIBUTION MIN - 1st pass
FILES_IN
  PARM p1 prm.top;
  STREAM s1 a1.trj a2.trj a3.trj;
FILES_OUT
  DISTRIBUTION d1 file MIN;
#                       ˆthis is the key: creates "file.min"
DECLARE
  GROUP g1 (SOLUTE);
  GROUP g2 (ATOM TYPE OW);
#                     ˆˆ have to use OW
OUTPUT
  DISTRIBUTION d1 0.0 10.0 10 DIST g1 g2;
#                 ˆˆˆˆˆˆˆˆˆˆˆ             you’ll also get the
#                                           net curve; RAW ok
#                             ˆˆˆˆ        this is the 2nd key
#                                   ˆˆˆˆ  order is solute then
#                                           solvent
END
```

--Now the critical step: filtering the water. First we use ’awk’ to see what waters we want to keep, e.g.:

```
awk ’$2 < 3.0 {print $1}’ file.min | wc -l
```

This tells you how many water oxygens came within the cutoff (3.0 in this example). Choose a cutoff such that the resulting list of type OW atoms is the right size for you and:

```
awk ’$2 < 3.0 {print $1}’ file.min > temp
```

Now you need to take the list of OWs in the temp file and include it in a GROUP ATOM statement as in the following example in order to select the waters into a coordinate dump.

```
# use of INH2O with DISTRIBUTION MIN - 2nd pass
FILES_IN
  PARM p1 prm.top;
  STREAM s1 a1.trj a2.trj a3.trj;
FILES_OUT
  COORD c1 filtered.trj;
DECLARE
  GROUP g1 (ATOM 2,11,26,29);
#                 ^^^^^^^^^^ the type OW atom numbers of your choice
OUTPUT
  COORD c1 INH2O g1;
END
```

9.4.5. Radial distance distributions

```
# DISTRIBUTION EXAMPLE: ONE GROUP to itself (OW-OW)
FILES_IN
  PARM p1 prmtop;
  STREAM s1 crd;
FILES_OUT
  DISTRIBUTION d1 xdb ;
DECLARE
  GROUP g1 (ATOM TYPE OW);
OUTPUT
  DISTRIBUTION d1 RAW DIST g1 ;
#                          ˆ group id from DECLARE GROUP
#                     ˆ distance macro
#                 ˆ dump all distances to file for use w/ rdis
#                   program (i.e. don’t bin measurements or
#                   output bins)
#              ˆ id
#       For each ’solvent’ group atom, the nearest ’solute’ atom
#       is found and binned if it satisfies the min, max criterion.
END
```

```
# DISTRIBUTION EXAMPLE: TWO GROUPS using closest atom in 1st to
#                       each of 2nd
FILES_IN
  PARM p1 prmtop;
  STREAM s1 crd;
FILES_OUT
  DISTRIBUTION d1 xdb ;
DECLARE
  GROUP g1 (RES NAME ADE);
  GROUP g2 (RES NAME THY);
```

```
OUTPUT
  DISTRIBUTION d1 RAW DIST g1 g2 ;
#                              ˆ 2nd group is the ’solvent’
#                           ˆ 1st group is the ’solute’
#                      ˆ distance macro
#                  ˆ dump all distances to file (don’t bin)
#               ˆ id
#       For each ’solvent’ group atom, the nearest ’solute’ atom
#       is found and binned if it satisfies the min, max criterion.
END

# DISTRIBUTION EXAMPLE: TWO GROUPS using all inter-group pairs
FILES_IN
  PARM p1 prmtop;
  STREAM s1 crd;
FILES_OUT
  DISTRIBUTION d1 xdb ;
DECLARE
  GROUP g1 (RES NAME ADE);
  GROUP g2 (RES NAME THY);
OUTPUT
  DISTRIBUTION d1 RAW DIST g1 g2 ALL;
#                                 ˆ consider all group1-group2
#                                   interactions
#                              ˆ 2nd group id from DECLARE GROUP
#                           ˆ 1st group id from DECLARE GROUP
#                      ˆ distance macro
#                  ˆ dump all distances to file (don’t bin)
#               ˆ id
END
```

9.4.6. Hbond examples

You want a occupancies for all possible hbonds at each step. This file will consist of a line for each coordinate set in the stream with a ’0’ or ’1’ followed by a space for each possible hbond. You also want the percentage occupancy of each hbond over the run, and the average distance and angle when occupied. And while you’re at it, you want to print the distance and angle of each possible hbond.

You also want to specify the maximum distance and angle that qualify an hbond. The percentages and averages are written to the main output at the end of the run.

```
FILES_IN
        PARM p1 hbtop;
        STREAM s1 hbmd;
FILES_OUT
        HBOND h1 xhb TABLE LIST;
#                           ˆ write a list of distances/angles
#                             to "xhb.lis"
#                     ˆ write occupancies to "xhb.tab"
#                 ˆ use "xhb" as the basis for filenames
#
#
DECLARE
OUTPUT
        HBOND h1 DISTANCE 3.3 ANGLE 20.0 STATS;
#                                         ˆ print the averages
#                                           of the occupied cases
#                              ˆ limit the angle;
#                                default 1 radian ˜= 60 degrees
#                 ˆ limit the distance between heavy atoms;
#                   default 4 Angstroms
END
```

Perhaps you want to specify the donor and acceptor groups, if only to limit the number of columns in the table. This time, we'll also just use the default criteria for hbonds.

```
# HBOND ANALYSIS EXAMPLE: USING GROUPS FOR DONOR/ACCEPTOR
FILES_IN
  PARM p1 hbtop;
  STREAM s1 hbmd;
FILES_OUT
  HBOND h1 xhb TABLE LIST;
DECLARE
DECLARE
  GROUP g1 (ATOM TYPE N2 NA);
  GROUP g2 (ATOM TYPE NC O );
OUTPUT
  HBOND h1 DONOR g1 ACCEPTOR g2 STATS;
END
```

10. 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 */DelPhi* program [95], 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 term [48]. The surface area is computed with Paul Beroza's *molsurf* program, which based on analytical ideas primarily developed by Mike Connolly [96]. 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 [97]. You may also wish to refer to a review summarizing many of the initial applications of this model [98], as well as to papers describing more recent applications [99-103].

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).

10.1. 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 labelled ".out", and the most useful summaries are in the "statistics.out" files. These give averages and standard deviations for various quantities, using the following labelling 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 *amber7/src/mm_pbsa/Examples* directory shows examples of running a "Stability" calcultion (*i.e.* estimating the free energy of one species), a "Binding" calculation (estimating Δ*G* for A + B → 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.

10.2. 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 using delphi (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
################################################################################
```

```
@DELPHI
#
# Delphi parameters (this section is only relevant if PB = 1 above)
#
#   The first group of the following parameters are passed to delphi.
#   Additional parameters of delphi (e.g. SALT 0.10) may be added here.
#   For further details see the delphi documentation.
#
#   FOCUS - If FOCUS > 0, subsequent (multiple) PERFIL and SCALE parameters are
#     used for multiple delphi calculations using the focussing technique.
#     The # of _focussing_ delphi calculations thus equals the value of FOCUS.
#   INDI - Dielectric constant for the molecule.
#   EXDI - Dielectric constant for the surrounding solvent.
#   PERFIL - Percentage of the lattice that the largest linear dimension of the
#            molecule will fill.
#   SCALE - Lattice spacing in no. of grids per Angstrom.
#   LINIT - No. of iterations with linear PB equation.
#   BNDCON - Type of boundary condition.
#   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.
#
FOCUS                 0
INDI                  1.0
EXDI                  80.0
PERFIL                80.0
SCALE                 2
LINIT                 1000
BNDCON                4
CHARGE                ./my_amber94_delphi.crg
SIZE                  ./my_parse_delphi.siz
#
SURFTEN               0.00542
SURFOFF               0.092
#
#################################################################################
@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), Onufriev's GB (2),
#             Jayaram's et al. GB (3) or Jayaram's et al. MGB (4).
#   SALTCON - Concentration (in M) of 1-1 mobile counterions in solution.
#   EXTDIEL - Dielectricity constant for the surrounding solvent.
```

```
#
#   SURFTEN / SURFOFF - Values used to compute the nonpolar contribution Gnp to
#                   the desolvation according to Gnp = SURFTEN * SASA + SURFOFF.
#
IGB                     4
SALTCON                 0.00
EXTDIEL                 80.0
#
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.
#   RADII - Name of the radii file.
#
PROBE                   1.4
RADII                   ./atmtypenumbers
#
#################################################################################
@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
#         trajecotry 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 preceeded 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
```

```
#
##################################################################################
```

10.3. 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)*

11. Profec

11.1. Introduction

PROFEC (Pictorial Representation Of Free Energy Changes) is a set of software tools for carrying out and displaying extrapolative free energy calculations. Specifically, the PROFEC software suite calculates the free energy for inserting a specified test particle at a grid of points near the ligand of interest. A weighted electrostatic potential is also calculated for each position on this grid. These two (van der Waals and electrostatic) grids can be visualized and overlaid on a three-dimensional structure of the ligand or ligand-receptor complex to suggest positions where modifications to the ligand would improve binding. The main strengths of PROFEC are its unbiased evaluation of possible modifications and the ability to explicitly include the effect of solvation in the analysis. The main weakness is its inability to deal with modifications at multiple sites or modifications that induce large confomational changes.

PROFEC consists of three programs: makeGrid and makeDiffGrid generate and manipulate the "free energy" grids, while the Field program allows interactive visualization and manipulation of the grid data. A detailed description of the PROFEC algorithms and a demonstration of its application to the benzamidine-trypsin complex can be found in R.J. Radmer and P.A. Kollman [104].

11.2. makeGrid

makeGrid takes four inputs: a control file, an AMBER topology file (using the "old" format), an AMBER trajectory, and a reference structure in AMBER restart format. The program then calculates the test particle grids (van der Waals and electrostatic) for a particle and location specified in the control file. These grids are calculated based on the input topology and trajectory, and projected onto the specified coordinates relative to the provided restart file. makeGrid is designed as a standard Unix program; the input trajectory should be piped to the makeGrid command and other inputs specified on the command line:

```
cat <traj> | makeGrid <in> <top> <crd> <ljp> <vdw> <esp> <obj> <sav>
```

<in>	Control file for makeGrid. The control file consists of a standard AMBER-style namelist followed by atom specifications, described in detail below. The namelist specifies control variables for the run (cutoff, grid dimensions and spacing) as well as indicating the atoms that define the grid center and axes. The subsequent atom specifications indicate which parts of the system to include in the free energy calculation. It is common in a protein-ligand complex to include everything _except_ the ligand in the calculation. This way the grid reflects only the influence of the protein and is not biased by nearby atoms of the ligand.
<top>	AMBER format topology file for makeGrid. This should be the correct topology for both the input trajectory and coordinate files.
<crd>	Reference coordinates in AMBER restart format. Once the grid is calculated, it will be rotated and translated to fit the appropriate atoms of this particular coordinate set.
<ljp>	Specifies location of lennard-jones particle(s) to be added to all calculations. The intent of this feature is to allow the user to calculate test partical grids for the case

where a lennard-jones particle is also added at the specified locations. Can be use-full for modeling a simple modification and calculating a new test partical grid without running an additional simulation. This should be used with caution, as the results are subject to more noise then a standard calculation.

<vdw> van der Waals "cost field"; the vdw file is a scalar field of the free energy required to insert the specified test particle on a grid of points relative to the ligand or atoms of interest, based on the coordinates and parameters supplied in the <traj> and <top> files. The grid is specified relative to the coordinates of the specified atoms as found in the <crd> file. See the Field section below for format information.

<esp> weighted electrostatic potential field; the esp file is a scalar field in a format similar to the vdw file. However, it contains the average electrostatic potential at each grid point. This is not a true electrostatic potential -- instead, it is weighted by the probability that the test particle would occupy this location (boltzman factor of the the vdw grid). This weighting avoids the infinite electrostatic contributions that result from overlaps of the test charge with real atoms. Again, format information is supplied with the description of the Field delegate, below.

<obj> indicates the location of lennard-jones particles added using the ljp file (above). This is intended for use with MIDAS but is not critical.

<sav> detailed save file; this file contains all of the makeGrid output data in a format suitable for more detailed analysis. In particular, the sav file contains the data necessary to concatenate the output of makeGrid runs on two different trajectories of the same topology.

11.3. makeGrid input format

Here is a sample input file:

```
 &cntrl
  Rprobe = 1.9080, Eprobe = 0.1094,
  iAtomO = 3877,    iAtomX = 3879,    iAtomY = 3883,
  nGridX = 15,   nGridY = 15,   nGridZ = 15,
  sGridX = 0.50, sGridY = 0.50, sGridZ = 0.50,
  Xtrans = 0.0,  Ytrans = 0.0,  Ztrans = 0.0,
  Temp   = 300.0,  cutoff = 8.0,
 &end
include all
exclude residueName DMT
```

The namelist is delimited by the " &cntrl" and " &end" lines. It defines the input variables for the run:

Rprobe, Eprobe
van der Waals parameters for the test particle. Rprobe is the R* value (in Angstroms) and Eprobe the epsilon (kcal/mol).

iAtomO Origin atom for the grid. In the absence of other input, the grid will be centered on this atom.

iAtomX Atom to define the x-coordinate of the grid.

iAtomY Atom to define the y-coordinate of the grid. iAtomO, iAtomX, and iAtomY are used to define a constant frame of reference for the grid. The z- axis is generated by the right hand rule. Typically all three are ligand atoms near the region of interest. Also, these atoms should not move much relative to one another, since they are assumed to be a relatively constant frame of reference.

nGridX, nGridY, nGridZ

number of grid points in each dimension; since the grid point (0,0,0) is centered on the origin (iAtomO), this must be an odd number.

sGridX, sGridY, sGridZ

grid spacing in each dimension, in Angstrom. A grid spacing of 0.5 or 0.25 Angstrom is typical.

Xtrans, Ytrans, Ztrans

translation of the grid away from iAtomO, in Angstroms. This can be used to center the grid at a position other than iAtomO (anywhere in space).

Temp Temperature used for the Boltzmann weighting and free energy calculation, in Kelvin. 300K is typical.

cutoff the cutoff (in Angstroms) used to truncate calculation of the van der Waals and electrostatic interactions. Note that this does _not_ have to be the cutoff used to generate the input trajectory.

The include and exclude commands specify the atoms to consider in the test particle insertion and electrostatics calculations. Typically all atoms are included, then the ligand of interest is specifically excluded. The commands are:

```
include/exclude all
                    residueName <name>
                    residueNumber <res>
                    residueRange <res1> <res2>
                    atomNumber <atom>
                    atomRange <atom1> <atom2>
```

where <name>, <res>, <atom>, etc. refer to residue names and residue or atom numbers as specified in the input topology <top>

11.4. makeDiffGrid

PROFEC is typically used to suggest changes to a ligand that might improve its binding to a given receptor. Since the free energy of binding involves both the bound and free states of the ligand, PROFEC can be used to compare the effects of ligand modification (test particle insertion) in two different states. For example, if we want to compare our reference ligand (A) with a modified ligand A’, we use the following thermodynamic cycle:

```
ligand A (protein) ---dG1 (PROFEC)---> ligand A’ (protein)
       |                                       |
    dGbind (A)                             dGbind (A’)
       |                                       |
               V                                      V
        ligand A (water)   ---dG2 (PROFEC)---> ligand A′ (water)
```

We want to know how the modification A -> A’ affects the binding free energies. That is, we wish to calculate ddGbind = dGbind (A) - dGbind (A’). By the thermodynamic cycle, we know that ddGbind = dG1 - dG2, so we can use PROFEC to calculate the value of interest, ddGbind.

This is done by generating grids from two different trajectories; first, grids are calculated for the protein-ligand complex. Second, grids are generated for a simulation of the free ligand (in water). Both of these grids must be defined relative to the _same_ set of atoms on the ligand in each trajectory.

Finally, the program makeDiffGrid can be used to generate a third grid file that is the difference of two input grids. For calculating the free energy of binding, one would use

```
makeDiffGrid <protein.vdw> <solvent.vdw> <output.vdw>
makeDiffGrid <protein.esp> <solvent.esp> <output.esp>
```

In the <output.vdw> grid, negative values indicate grid positions where adding the test particle favors the protein-ligand complex at the expense of the water-ligand interaction. These are the locations where the ligand might be modified to increase the binding free energy. The <output.esp> file, however, gives a more qualitative indication of how charge should be distributed to favor binding; negatively charged groups should be added in regions of positive electrostatic potential, and vice versa.

For other applications of makeGrid and makeDiffGrid, please consult the Radmer and Kollman paper cited above.

11.5. Field

Field is a delegate for the MidasPlus graphics program that overlays the grids created by makeGrid or makeDiffGrid on top of a three-dimensional molecular structure. Field permits interactive contouring and coloring of the van der Waals and electrostatic grids, allowing the user to visualize areas of the molecule where modifications might be favorable or unfavorable. Field is invoked like any other MidasPlus delegate; from within MidasPlus, load the structure of interest (the reference structure used in makeGrid), and then start the Field delegate (refer to the MidasPlus manual for details on the delegate function).

Once the Field delegate is started, the following commands may be used:

```
read contour <vdw-filename>
read color <esp-filename>
contour <vdw-value>
color <esp-value1> <color1> <esp-value2> <color2>
erase
slice (No longer suppored)
```

If you do not have access to the MidasPlus program, or wish to use another visualization package, the exact format of the vdw and esp grid files is supplied below.

11.6. Data formats

```
vdw, esp format:

## header and comments
##
## input, output files
##
## control variables from <in>
##
## included, excluded atoms
##
## atom type information from <top>
##
##
          21          21          21    # number of Grid spaces (3I12)
        .500        .500        .500    # size of Grid spacing  (3F12.3)
    -.109328     .993533    -.030654    # rotation matrix       (3F12.6)
     .900871     .112071     .419371    # rotation matrix       (3F12.6)
     .420094     .018234    -.907297    # rotation matrix       (3F12.6)
   51.064120   30.547364   24.991392    # translation vector    (3F12.6)
  .92329E+01  .54755E+01  .54026E+01  .57455E+01  .14020E+02  .34699E+02
grid data points continue in (6E12.5) format, ZYX slices without indexing
```

12. 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.

12.1. 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. There are accompanying programs *nmanal* (normal mode analysis) and *lmanal* (Langevin mode analysis) that use the output of these programs to compute molecular fluctuations and time correlation functions. *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 inplemented the non-additive capabilities.

The second derivative routines are based on expressions used in the Consistent Force Field programs [105]. 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 [106], as described by Nguyen and Case [107]. 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 [108], and were implemented in the Amber environment by Kottalam and Case [109].

12.2. 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 File 5, 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.

12.3. Files

```
nmdin  : control input for the run
nmdout : standard output file for print and error messages
prmtop : parameter file as output by the AMBER program parm
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
```

12.4. 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)
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 unformattted form; 1 to use the formatted option (default).
ntx	0 if the input coordinates are to be read in unformattted form; 1 to use the formatted option (default).
ntxo	0 if the output (restart) coordinates are to be written out in unformattted form; 1 to use the formatted option (default).

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) , 0.00d0). (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 of format.

12.5. quasih

NAME

quasih − compute quasiharmonic frequencies and directions from a trajectory

SYNOPSIS

quasih -natom # [-f # -m <mass> -x <xyz>} [-novec | -v <vecs>] [-first #] [-last #] [-nmode #]

DESCRIPTION

quasih reads in a trajectory file, computes the matrix of cartesian fluctuations, diagonalizes this matrix, and computes the quasiharmonic frequencies and directions [110]. The input coordinates (on STDIN) should first have been run through *ptraj* to superimpose on a common coordinate frame.

OPTIONS

The number of atoms, *natom*, is a required parameter. Parameters *first* and *last* refer to snapshot numbers in the input file: only parts of the trajectory between these two snapshot values will be used to compute the flucutation matrix. The *nmode* parameter (required) indicates how many eigenvectors will be computed; the default compiled-in maximum is 50. The *-f* flag specifies type of input, and is 2 for a standard Amber trajectory, and 5 for a trajectory in the "binpos" format.

FILES

The *mass* file is required for input: it contains floating point numbers (10F8.2 format) giving atomic masses, in the same order as the atoms in the input stream. The script *pdb_to_mass* can generally be used to construct this file.

Output files are *xyz*, holding the average coordinates, and *vecs*, holding the eigenvectors. By default, the *Makefile* defines a symbol FORMATTED, which makes these files be human-readable. For large systems, you can recompile without this symbol to save on output disk space. Information about the frequencies, and thermodynamic values calculated from them, are sent to STDOUT.

12.6. nmanal

Usage:

```
      nmanal [-O] -i nmdin -o nmdout -p prmtop -v vecs -r rvecs
```

−O	Overwrite output files if they exist.

This is a general routine to do vibrational analysis by projecting cartesian normal mode eigenvectors (generated by the nmode program) onto internal coordinates or onto "rigid groups." For each internal coordinate, the program will determine the projection of each normal mode onto that coordinate, and the fraction of the total potential energy change along the normal mode that is contributed by that internal coordinate (the "potential energy distribution" for each mode.) You can also sum over all modes to obtain the rms thermal fluctuations for any particular internal coordinate.

The program can also compute the rms thermal fluctuations of atoms in a cartesian coordinate frame, and will compute time correlation functions and fluctuations of internal coordinates.

The original code was written by D. T. Nguyen and D. A. Case at U. C. Davis, 1985. The RMS analysis added by Barbara Rudolph. Capabilities for time correlation functions were added by J. Kottalam at Scripps Clinic. Responsibility for the final versions of the codes (and for any bugs) rests with Dave Case.

Files used in the program:

```
nmdin  : control input for the run, described below.
nmdout : standard output file for print and error messages
prmtop : formatted parameter file
vecs   : formatted file containing output normal mode
         frequencies and eigenvectors. This file is generated
         by the program nmode.
rvecs  : formatted file containing the reference eigenvectors
         and associated frequencies
```

Input found on nmdin:

Card 1: Title of the run

Cards 2: namelist /data/, which contains the following parameters:

ntrun	Values of *ntrun* from −1 to 3 are used to analyze modes in terms of internal coordinates (if *ipro = 1*), to compute thermal fluctuations in internal coordinates (if *ifluc = 1*), or to compute time correlation and cross-correlation functions (if *ntrun = -1*). Options 4 and 5 are present only for historical reasons (although the code might be a good starting point for some interesting calculations), and options 6 to 8 carry out some specialized tasks: *(default=1)*

	=-1 project eigenvectors onto those internal coordinates read in on subsequent cards (labelled "3c", below). = 0 project eigenvectors onto bonds only = 1 project eigenvectors onto all internal coordinates = 2 " " " angles and dihedral angles = 3 " " " dihedral angles only = 4 project eigenvectors onto "dynamics groups" = 5 project eigenvectors of the system('system vectors') onto reference eigenvectors. (This is a fairly specialized option, and you will probably have to look at the code to see what you really get. It has rarely been used.) = 6 just calculate rms fluctuations in cartesian coordinates. = 7 compute dipole-dipole correlation functions. In this case, prmtop is not read, and the &data namelist must be followed by cards that have two integers per card (free format), giving the atom numbers for each pair for which the correlation functions are desired. See subroutine "corf" for details of the calculational procedure. = 8 project MD snapshots onto normal mode directions
nvect	= number of eigenvectors in file vecs to be read in (default=50)
ivform	= 0 if the input vectors are in unformatted form = 1 for input vectors in formatted form (default)
ieff	= 0 use true frequencies (default) = 1 use effective frequencies (not implemented!)
pcut	cutoff value for printing out projections: print will occur if the estimated contribution of an internal coordinate to the total potential energy distribution along this mode exceed pcut. (default = 0.02)
ibelly	= 0 (default) no belly = 1 belly calculation
ibeg	first eigenvector to be analyzed (default = 7)
iend	last eigenvector to be analyzed (default = 50)
ifluc	= 0 don't do the rms internal crds. fluctuation(default) = 1 do the rms internal crds. fluctuation for the internal coordinates selected by the "ntrun" variable
ipro	= 0 don't print out the projections onto internal crds. = 1 print out the projections onto int. crds (default)
bose	true. if quantum (Bose) statistics are to be used in populating the modes; .false. (default) if classical (Boltzmann) statistics are to be used.
natom	number of atoms; only needed if ntrun=7 or 8
ihsful	= 0 if dipole-dipole correlations do not include contributions from distance fluctuations =1 (default) if both distance and angle fluctuations are to be included in dipole-dipole correlations.
tmax	maximum value for time correlation functions if *ntrun* = 7. Default is 0.0, i.e. no time correlations will be carried out.
tintvl	interval for time correlation functions (default = 1.0).

The following are only used if ntrun=8:

first	first snapshot from MD simulation be be projected onto normal mode directions, when *ntrun* = 8. Default = 1.
last	last snapshot to be projected. Default = 9999.
iskip	every *iksip*-th snapshot will be projected. Default = 1.

The following are only used if ntrun=5:

nrgrp	number of rigid groups (default = 0)
nrvec	number of reference eigenvectors to be read in from file "rvecs" (only if ntrun=5; default=0)
nrat	number of atoms in reference system (default=0)
iat	first atom number of the part of the system to be excised and compared to reference system (only if ntrun = 5; default=1)
jat	last atom number of the part of the system to be excised (only if ntrun = 5; default = natom)
imov	flag to rotate/translate eigenvectors of system and reference to principal axes. This essentially decouples translation and rotation of the excised part of the system as a rigid body from the rest of the vibrational motion (only if ntrun=5, default=0).

Cards 3a group cards for the parts of the molecule that move, only if ibelly.ne.0. See group documentation for format. This card is not needed when ibelly = 0.

Cards 3b group cards for subdividing the molecule into rigid groups (if ntrun=4). See group documentation for format. Each rigid group will have its own set of cards 3b.

Cards 3c *(if ntrun .eq. -1)* Quantities for which time correlation functions are to be calculated. These quantities are of the form of internal coordinates. All input is free format, which means that you must enter all numbers -- blanks are ignored.

```
TYPE, IAT, JAT, KAT, LAT
  INTNAME = identifier for this internal coordinate (character*8)
  IAT, JAT, KAT and LAT  are atom numbers.  Set LAT to
    zero for bonds and angle, KAT to zero for bonds.  Repeat this card
    for as many internal coordinates as you are interested in, up the
    the value of MAXINT specified in the "sizes.h" header file.
```

Input is terminated when the end-of-file is reached.

12.7. lmanal

Usage:

```
lmanal [-O] -i lmdin -o lmdout -c inpcrd -l lmode
```

−O	Overwrite output files if they exist.

This program will compute time correlation functions from Langevin modes. Note that since the time-independent aspects of the molecular normal mode description are independent of solvent viscosity, all of the equal-time correlations (such as rms fluctuations in cartesian or internal coordinates) will be the same as for the vacuum calculation. Hence the companion program *nmanal* should be used to compute those.

Input description for the *lmdin* file:

```
namelist   default    meaning
&data
 ntrun       1        'type of run' flag
                      1: correlation function calculated is
                         for the deviation of the length of
                         the vector from the reference value
                         in the minimum energy structure.
                         i.e., <dr(t)dr(0)> / <dr(0)dr(0)>
                      2: for the orientation of the vector
                         i.e., <P2[r(t).r(0)]>
                      3: the frequency distribution is plotted
                         i.e., the imaginary parts of the
                         eigenvalues are counted at every interval
                         of 10 wavenumbers. Other input parameters
                         are irrelevant.
 kup         1
 lup         2        atom numbers defining a position vector
 nvect      12        number of langevin modes to be used
 tf        2.0        final time for correlation functions
                      i.e., t ranges from 0.0 to tf picoseconds
 np       1000        number of points at which to calculate
                      correlation function between t = 0.0 and
                      t = tf ps.
bose      .false.    .true. if quantum (Bose) statistics are to be
                      used in populating the modes; =.false. (default)
                      if classical (Boltzmann) statistics are to be
                      used.
&end
```

The input file *inpcrd* is a standard Amber coordinate file; the file *lmode* is that created by the *nmode* program with ntrun = 5. Output files are CORF (if ntrun = 1 or 2) and DENS, CUMU (if ntrun = 3.) All of the output files (except *lmdout*) are input files for the <plot79> package, which will create plots of the correlation functions. You should(?) have little trouble converting them to some other plotting package in order to see the correlation functions.

Sample input file for nmode with ntrun=1

```
get vibrational modes for staph nuclease ternary complex
&data
    ntrun = 1,            do vibrational calculation
    cut=10.0,             cutoff; use same value in minimization
    idiel=0,              distance-dependent dielectric
    nvect=6753,           write out all 3*N modes...
    ivform=0,             ...in unformatted form...
    ilevel=0,             ...with no removal of trans. & rotation
    drms = 0.0001,        will complain if rms gradient is not
                            less than this
&end
```

Sample input file for nmanal with ntrun=7

```
#
 Get N-H S**2 values from quasi-harmonic modes
 &data
  ntrun=7,                compute dipole-dipole correlation fns.
  nvect=4496,             this many modes in input file
  ibeg=1, iend=4496,      use all of the modes for the calculation
  ivform=0                unformatted modes files
  natom=1529,             molecule has this many atoms
  ihsful=0,               do no include distance fluctuations
 &end
3 4                       atom numbers for N and H of residue 1
11 12
20 21
28 29
38 39
```

13. 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. [111-113] 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     flag  fortran            purpose
name            unit
---------------------------------------------------------------------
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 -q)
```

```
  -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.

14. Miscellaneous

14.1. 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 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.

-------------------------------------------------------------
```

14.2. 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 #]
```

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.

-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*.

-*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.

14.3. 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.

14.4. 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 [-p <prmtop-file>] [-w <water-position-file>] < input-pdb-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. If the -*w* flag is set, the program reads water oxygen positions from the file *water-position-file*, and uses these as well to help fix hydrogen positions.

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.

14.5. 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...".

14.6. 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*, *intfile* 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.

14.7. 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 errorsˆ2 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
```

15. Anal

Usage:

```
anal [-O] -i analin -o analout -p prmtop -c inpcrd [-ref refc ]
```

−O Overwrite output files if they exist.

15.1. Introduction

This is the energy analysis module of AMBER. Its purpose is to do energetic analyses of individual structures. The key function of this program is decomposition of the energy among different groups of atoms in order to find the interaction energies between different parts of the system. The program puts those atoms which are not in explicitly defined groups into a separate group. In the case of a belly or partial minimization the unfrozen part of the system can be defined as the desired groups and the frozen part will be automatically taken by the program as an additional group.

15.2. Files

```
analin    5     Input control data for the analysis run
analout   6     Output results
prmtop   20     Input Molecular topology file from PARM
inpcrd   21     Input Coordinates to be analyzed
refc     24     Input positional constraint coordinates
```

15.3. Input description

```
- 1 -       TITLE

           FORMAT(20A4)

TITLE       Title for identification.
```

```
- 2 -      NTX , NTXO , NRC , NRCX , NGRPX , KFORM,    FORMAT(6I)

NTX        Format of coordinates.

 = 1  Formatted (inpcrd, unit 21)
 = 2  Unformatted (inpcrd, unit 21)
```

```
NTXO       Read but not used

NRC        Option to read position constraints.

 = 0  no constraints
 = 1  constrained minimization

      The atoms to be constrained are read as groups with
      different harmonic force constants for each group.
      Consult the section on GROUP in the Appendices for
      group specification format.
      When using positional constraints, the constrained
      groups are given first in the group input followed
      by the groups for energy analysis.

NRCX       Format of constraint coordinates.   The constraint
      coordinate file has the same organization as the structure
      coordinates.

 = 0  Formatted (refc, unit 24)
 = 1  Binary (refc, Unit 24)

NGRPX      Maximum number of groups that the structure can be
      divided into for analysis. Note that if any atoms are not
      explicitly included in a group, they will automatically
      be put in an additional group.

 = 0  Default = 70
 = N  Structure may be partitioned into N different groups

KFORM      The Flag for the type of Topology File
 = 0  Binary    (prmtop, unit 20)
 = 1  Formatted (prmtop, unit 20)
```

```
- 3 -      NTB , BOX(1) , BOX(2) , BOX(3) , BETA

          FORMAT(I,4F)

NTB        Flag for periodic boundary conditions.
      ( not yet operational )

 =-n  Periodicity is applied.  Box is truncated octahedron
      (BETA = 90)
 = 0  No periodicity is applied
 =+n  Periodicity is applied.  Box is rectangular or monoclinic
      depending on the value of BETA

BOX(1..3)  Lengths of the edges of the periodic box
```

```
BETA       Angle between the X- and Z- axes of the box in degrees.
      the Y- axis is assumed to be orthogonal to the other
      axes. ( 0 < BETA < 180 )
```

```
- 4 -      NTF , NTID , NTN , NTNB , NSNB , IDIEL,   FORMAT(6I)

NTF        Flag for force evaluation.

 = 1  complete interaction is calculated

 = 2  bond interactions involving H-atoms omitted

 = 3  all the bond interactions are omitted

 = 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

NTID       Flag for improper dihedral contribution (read but not used).

NTN        Read but not used

      Note:   non-bonded interactions are now always calculated
      using a residue based cutoff.  The nb pairs are stored as
      residue pairs.  This uses substantially less memory than
      the atom pairlist in the minimizer.

NTNB       Read but not used

NSNB       Read but not used

IDIEL      Flag for the type of dielectric function to be used in
      calculating the electrostatic energy.

 = 0  distance dependent dielectric function
 = 1  constant dielectric function
```

```
- 5 -      CUT , SCNB , SCEE , DIELC,     FORMAT(4F)
```

```
CUT        The cutoff distance for the non-bonded pairs.


SCNB       1-4 vdw interactions are divided by SCNB.
      if SCNB .le. 0.0 then SCNB = 2.0

SCEE       1-4 electrostatic interactions are divided by SCEE
      if SCEE .le. 0.0 then SCEE = 2.0

DIELC      Dielectric multiplicative constant for the electrostatic
      interactions.  If DIELC .le. 0.0 then DIELC = 1.0.  DIELC
      and IDIEL are coupled.  For example to obtain a dielectric
      ’constant’ of 4rij set DIELC=4 and IDIEL=0.
```

```
- 6 -      Printout of energies beyond specified values.  You
      must use the ENERGY keyword to obtain output.

      IMAX , EMAX(I) , I = 1, 9,       FORMAT(I,9F)

IMAX       Flag for printing the energy contributions.

 = 0  no printing
 = 1  energy contributions will be printed

EMAX(1)    All the bonds whose energy contribution is greater
      than EMAX(1) will be printed.

EMAX(2)    All the angles whose energy contribution is greater
      than EMAX(2) will be printed.

EMAX(3)    All the dihedrals whose energy contribution is
      greater than EMAX(3) will be printed.

EMAX(4)    All the 1-4 vdw whose energy contribution is greater
      than EMAX(4) will be printed.

EMAX(5)    All the 1-4 eel whose energy contribution is greater
      than EMAX(5) will be printed.

EMAX(6)    All the vdw nb pairs whose energy contribution is
      greater than EMAX(6) will be printed.

EMAX(7)    All the eel nb-pairs with absolute value of energy
      greater than EMAX(7) will be printed.

EMAX(8)    All the H-bond pairs whose energy contribution is
      greater than EMAX(8) will be printed.
```

EMAX(9)	All the constrained atoms whose energy contribution is greater than EMAX(9) will be printed.

- 7 - The control for doing the desired options. The only currently-supported control word is ENERGY.

```
IOPT,      FORMAT(A)
```

IOPT	The control word for the option.
’ENERGY’	Energy decomposition into groups
’STOP’	Control to terminate the run

ENERGY

Parts of the molecule for which interaction energies are to be calculated are entered in GROUP format. See the section on GROUP in the Appendices for details. Groups are read sequentially in any order. Each group is terminated by an "END" card.

The ENERGY option is terminated by another "END" card.

16. Appendices

16.1. 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, 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,
 &end
```

Note the initial space before the "&"s. 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). A comma (or the terminal &END) must follow each constant; end-of-line does NOT constitute a valid constant separator.

The NAMELIST name &name must ALWAYS begin in column 2 of an input record. The ampersand sign in &name and &end may optionally be replaced by a dollar sign, and the word "end" after the dollar sign may optionally be omitted. Column 1 of all input records is ignored, and only columns 2..80 are examined (in some implementations, a non-blank in the first column comments out the whole line). The terminal &end may occur anywhere in the input character stream (ignoring column 1 of course); it need not begin in column 2.

Letter case is ignored in all character comparisons, but case is preserved in string constants. An exception is that the namelist name itself must appear in lower case, e.g. *&cntrl*, not *&CNTRL*. 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.

Scalar variables may NOT be subscripted, and must be followed by 0 or 1 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.

16.2. 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 simuation), 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 satisify 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 satisify 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.

16.3. Appendix C: 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 Appendix D.

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 [13]. 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:

```
          R                          R
          |                          |
         :S=O                      O=S=O
          |                          |
          R′                         R′
```

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"):

```
          R
           \
            C=O
           /
          R′
```

or to those of carboxyl or phosphate oxygens (type "O2" in both examples)

```
                                      R
                                      |
         O2                          OS
        /                             |
  R - C                         O2 - P - O2
        \                             |
         O2                          OS
                                      |
                                      R′
```

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+ [114]. Another source of ion "size" information is crystallographic studies of ion complexes [115]. 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 [23]. 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 [21] (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 [116]. 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 progessively 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 [117,118]. 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 [119]. Cannon gives an example of extending the Weiner *et al.* force field to guanosine triphosphate and analogs [120]. 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:

$$\text{Etors} = (\text{PK} / \text{IDIVF}) * (1 + \cos(\text{PN} * \text{phi} - \text{PHASE}))$$

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 bonds (X-CT-CT-X), or to make it explicit torsion restricted to HC substituents (HC-CT-CT-HC).

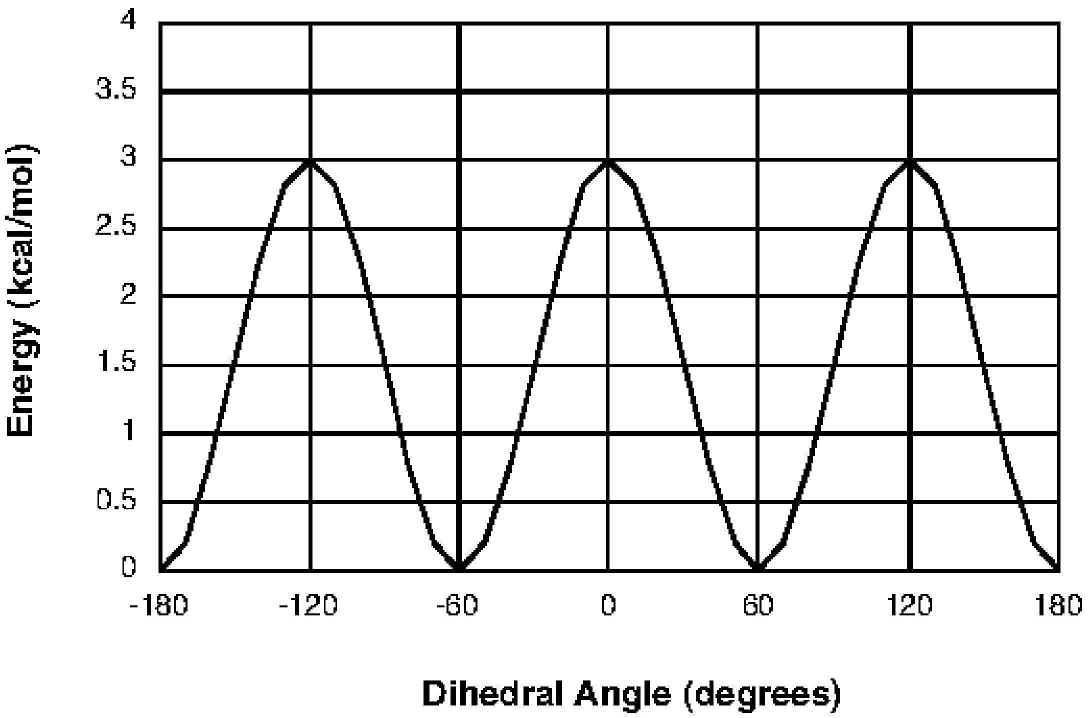


Figure 1

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:
```

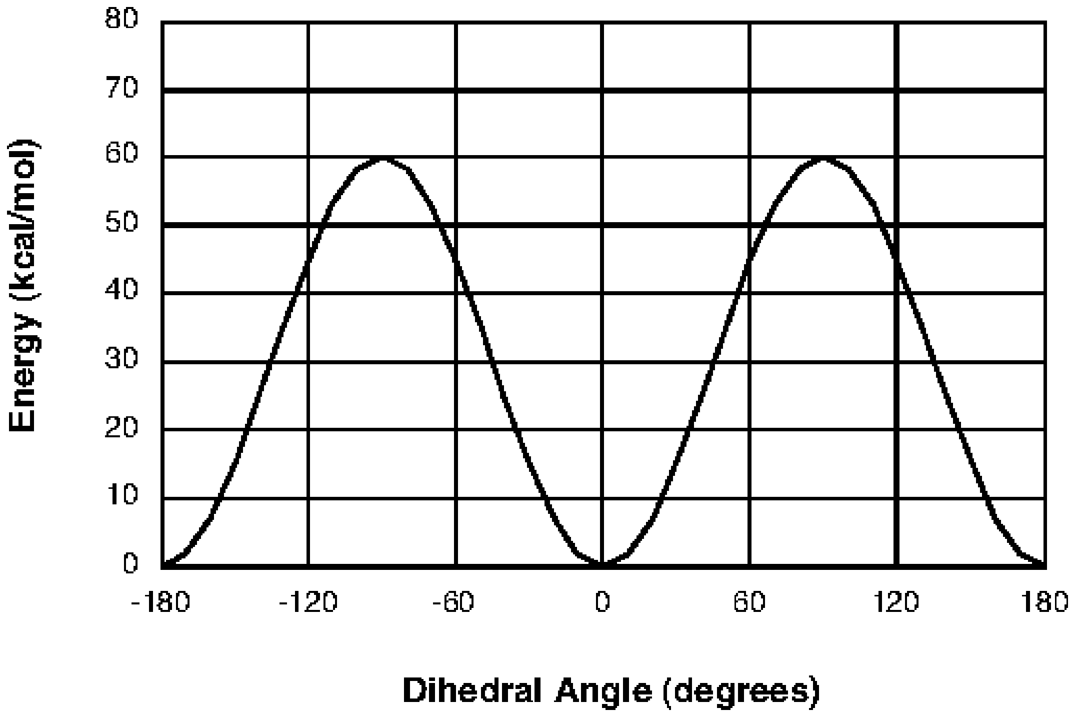


Figure 2

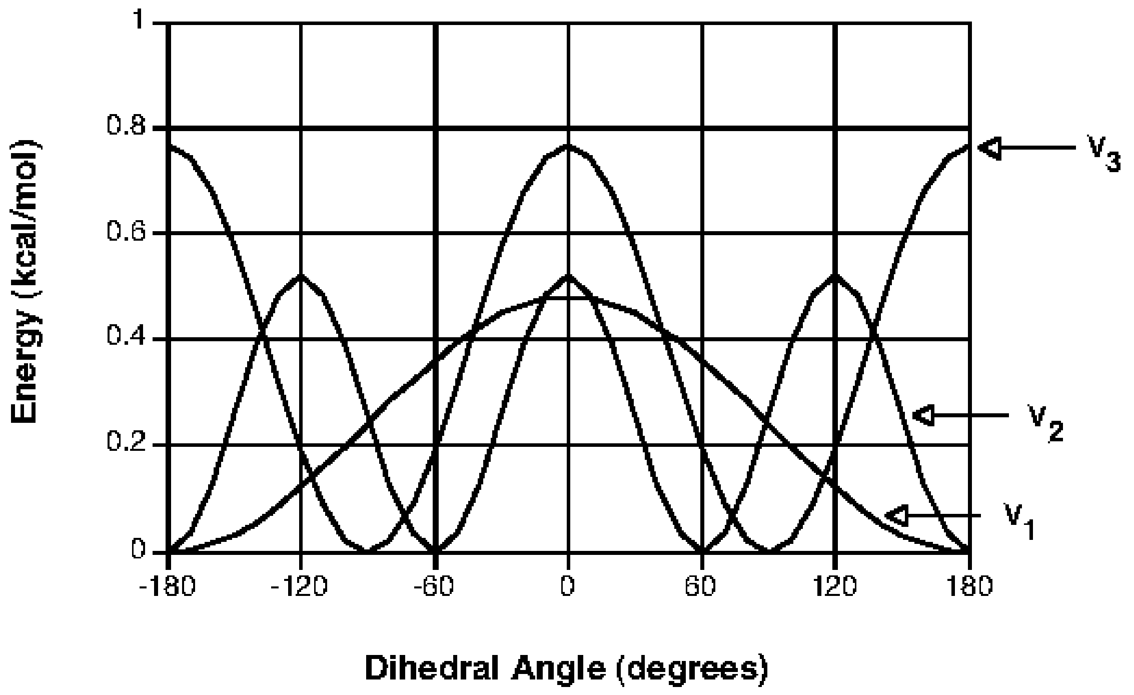
```
IDIVF = 4
PK    = 60.0 kcal/mol / 2.0 = 30.0 kcal/mol
```

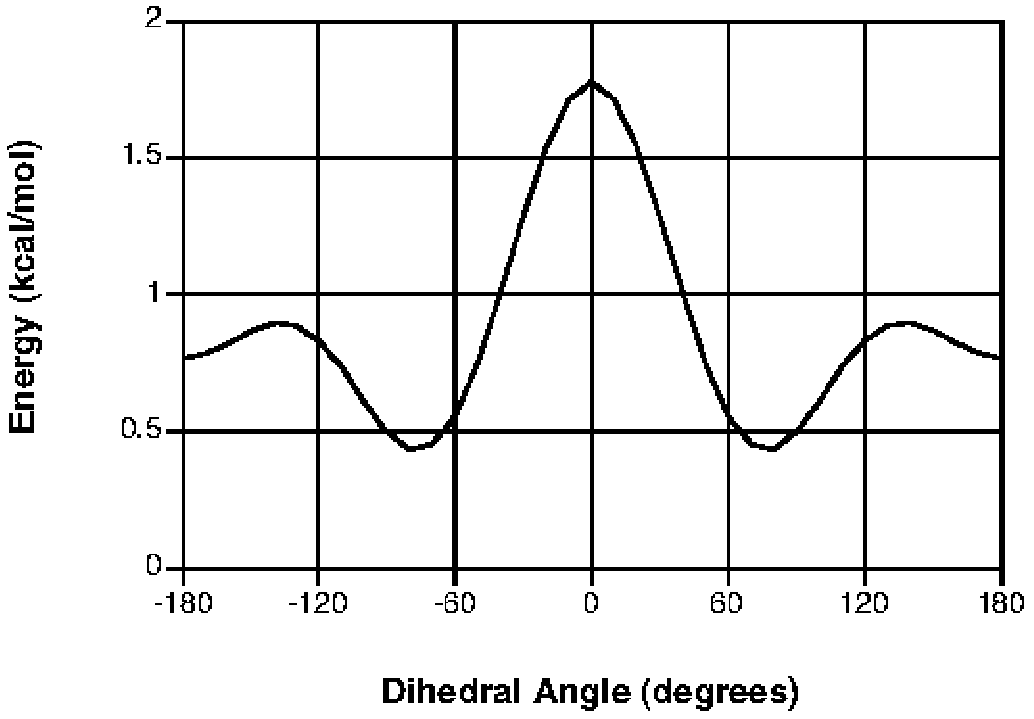
Finally, we examine a hypothetical molecule ZH2C-CH2Z, where Z represents an electronegative functional group. Let us imagine that we either have experimental data on the relative conformational energies or we have simulated the rotational potential of this molecule with a series of quantum mechanical 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





Figures 3 and 4

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 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:

```
       J
       |
       K
      / \
     I   L
```

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 they 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.

16.4. Appendix D: 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. [113] 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 [111,112]. 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 [121]. 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 [122] 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.

16.5. Appendix E: 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  : currently not used
  NNB    : number of excluded atoms
  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 pertrubed 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)
```

16.6. Appendix F: 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

Note: in AMBER 4.1 if the ewald option is turned on, the box angles
will also be written out in the same format.
```

```
GIBBS will print extra information.
```

16.7. Appendix G: trajectory (coordinates or velocity) 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
```

17. 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. S. Harvey & J.A. McCammon. *Dynamics of Proteins and Nucleic Acids.* Cambridge: Cambridge University Press, (1987).
3. M.P. Allen & D.J. Tildesley. *Computer Simulation of Liquids.* Oxford: Clarendon Press, (1987).
4. D. Frenkel & B. Smit. *Understanding Molecular Simulation: From Algorithms to Applications.* San Diego: Academic Press, (1996).
5. W.F. van Gunsteren, P.K. Weiner & A.J. Wilkinson, eds.. *Computer Simulations of Biomolecular Systems, Vol. 2..* Leiden: ESCOM Science Publishers, (1993).
6. W.F. van Gunsteren, P.K. Weiner & A.J. Wilkinson, eds.. *Computer Simulations of Biomolecular Systems, Vol. 3..* Leiden: ESCOM Science Publishers, (1997).
7. L.R. Pratt & G. Hummer, eds.. *Simulation and Theory of Electrostatic Interactions in Solution.* Melville, NY: American Institute of Physics, (1999).
8. J.J. Vincent & K.M. Merz, Jr.. A highly portable parallel implementation of AMBER4 using the message passing interface standard. *J. Computat. Chem.* **16,** 1420-1427 (1995).
9. J. Wang, P. Cieplak & P.A. Kollman. How well does a restrained electrostatic potential (RESP) model perform in calcluating conformational energies of organic and biological molecules?. *J. Comput. Chem.* **21,** 1049-1074 (2000).
10. 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. Computat. Chem.* **22,** 1048-1057 (2001).
11. R.W. Dixon & P.A. Kollman. Advancing Beyond the Atom-Centered Model in Additive and Nonadditive Molecular Mechanics.. *J. Computat. Chem.* **18,** 1632-1646 (1997).
12. 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).
13. 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).
14. 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.

15. M.D. Beachy & R.A. Friesner. *J. Am. Chem. Soc.* **119,** 5908-5920 (1997).

16. 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 termperatures. *Prot. Sci.* **8,** 1292-1304 (1999).

17. 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).

18. 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).

19. 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).

20. 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. Computat. Chem.* **7,** 230-252 (1986).

21. J. Åqvist. Ion-water interaction potentials derived from free energy perturbation simulations. *J. Phys. Chem.* **94,** 8021-8024 (1990).

22. T. Darden, D. Pearlman & L.G. Pedersen. Ionic charging free energies: Spherical versus periodic boundary conditions. *J. Chem. Phys.* **109,** 10921-10935 (1998).

23. 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).

24. W.L. Jorgensen & J.D. Madura. *Mol. Phys.* **56,** 1381 (1985).

25. 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).

26. 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).

27. H.J.C. Berendsen, J.R. Grigera & T.P. Straatsma. The missing term in effective pair potentials. *J. Phys. Chem.* **91,** 6269-6271 (1987).

28. 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).

29. 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).

30. 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).

31. 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. Computat. Chem.* **21,** 132-146 (2000).

32. J. Wang & P.A. Kollman. Automatic Parameterization of Force Field by Systematic Search and Genetic Algorithms. *J. Computat. Chem.* **22,** 1219-1228 (2001).

33. T.A. Halgren. Merck Molecular Force Field (MMFF94). Part I-V. *J. Comput. Chem.* **17,** 490-641 (1996).

34. 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).

35. 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).

36. 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.

37. 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).

38. 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).

39. 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).

40. W.C. Still, A. Tempczyk, R.C. Hawley & T. Hendrickson. Semianalytical treatement of solvation for molecular mechanics and dynamics. *J. Am. Chem. Soc.* **112,** 6127-6129 (1990).

41. 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).

42. M. Schaefer, H.W.T. Van Vlijmen & M. Karplus. Electrostatic contributions to molecular free energies in solution.. *Adv. Protein Chem.* **51,** 1-57 (1998).

43. D. Bashford & D.A. Case. Generalized Born Models of Macromolecular Solvation Effects. *Annu. Rev. Phys. Chem.* **51,** 129-152 (2000).

44. A. Bondi. *J. Chem. Phys.* **64,** 441 (1964).

45. 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).

46. A. Onufriev, D. Bashford & D.A. Case. Modification of the Generalized Born Model Suitable for Macromolecules. *J. Phys. Chem. B* **104,** 3712-3720 (2000).

47. J. Weiser, P.S. Shenkin & W.C. Still. Approximate Atomic Surfaces from Linear Combinations of Pairwise Overlaps (LCPO). *J. Computat. Chem.* **20,** 217-230 (1999).

48. D. Sitkoff, K.A. Sharp & B. Honig. Accurate calculation of hydration free energies using macroscopic solvent models. *J. Phys. Chem.* **98,** 1978-1988 (1994).

49. 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).

50. T. Morishita. Fluctuation formulas in molecular-dynamics simulations with the weak coupling heat bath. *J. Chem. Phys.* **113,** 2976 (2000).

51. 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. Computat. Phys.* **23,** 327-341 (1977).

52. S. Miyamoto & P.A. Kollman. SETTLE: An analytical version of the SHAKE and RATTLE algorithm for rigid water models. *J. Computat. Chem.* **13,** 952-962 (1992).

53. G. Hummer & A. Szabo. Calculation of free-energy differences from computer simulations of initial and final states. *J. Chem. Phys.* **105,** 2004-2010 (1996).

54. 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).

55. A. Kalk & H.J.C. Berendsen. Proton magnetic relaxation and spin diffusion in proteins. *J. Magn. Reson.* **24,** 343-366 (1976).

56. E.T. Olejniczak & M.A. Weiss. Are methyl groups relaxation sinks in small proteins?. *J. Magn. Reson.* **86,** 148-155 (1990).

57. K.J. Cross & P.E. Wright. Calibration of ring-current models for the heme ring. *J. Magn. Reson.* **64,** 220-231 (1985).

58. K. Ösapay & D.A. Case. A new analysis of proton chemical shifts in proteins. *J. Am. Chem. Soc.* **113,** 9436-9444 (1991).

59. D.A. Case. Calibration of ring-current effects in proteins and nucleic acids. *J. Biomol. NMR* **6,** 341-346 (1995).

60. 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).

61. D.A. Case. Calculations of NMR dipolar coupling strengths in model peptides. *J. Biomol. NMR* **15,** 95-102 (1999).

62. 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).

63. 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).

64. 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.

65. 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).

66. R. Brüschweiler & D.A. Case. Characterization of biomolecular structure and dynamics by NMR cross-relaxation. *Prog. NMR Spectr.* **26,** 27-58 (1994).

67. D.A. Case. The use of chemical shifts and their anisotropies in biomolecular structure determination. *Curr. Opin. Struct. Biol.* **8,** 624-630 (1998).

68. D.S. Wishart & D.A. Case. Use of chemical shifts in macromolecular structure determination.. *Meth. Enzymol.* **338,** 3-34 (2001).

69. A.E. Torda, R.M. Scheek & W.F. VanGunsteren. Time-dependent distance restraints in molecular dynamics simulations. *Chem. Phys. Lett.* **157,** 289-294 (1989).

70. 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).

71. D.A. Pearlman. How well to time-averaged J-coupling restraints work?. *J. Biomol. NMR* **4,** 279-299 (1994).

72. 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).

73. R. Elber & M. Karplus. Enchanced 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).

74. A. Roitberg & R. Elber. *J. Chem. Phys.* **95,** 9277 (1991).

75. 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).

76. 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).

77. C. Simmerling, J.L. Miller & P.A. Kollman. Combined locally enchanced 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).

78. 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: Applicatin to the Small Protein CMTI-1. *J. Am. Chem. Soc.* **122,** 8392-8402 (2000).

79. A. Miranker & M. Karplus. Functionality maps of binding sites: A multiple copy simultaneous search method. *Proteins: Str. Funct. Gen.* **11,** 29-34 (1991).

80. J.E. Straub & M. Karplus. *J. Chem. Phys.* **94,** 6737 (1991).

81. A. Ulitsky & R. Elber. *J. Chem. Phys.* **98,** 3380 (1993).

82. D.L. Beveridge & F.M. DiCapua. Free energy simulation via molecular simulations: Applications to chemical and biomolecular systems. *Annu. Rev. Biophys. Biophys. Chem.* **18,** 431-492 (1989).

83. P. Kollman. Free energy calculations: Applications to chemical and biochemical phenomena. *Chem. Rev.* **93,** 2395-2417 (1993).

84. D.A. Pearlman & B.G. Rao. Free energy calculations: Methods and applications. In *Encyclopedia of Computational Chemistry*, P. von R. Schleyer, N.L. Allinger, T. Clark, J. Gasteiger, P.A. Kollman & H.F. Schaefer, III, Ed. Chichester: John Wiley, (1998). pp. 1036-1061.

85. D.A. Pearlman & P.A. Kollman. A New Method for Carrying Out Free Energy Perturbation Calculations: Dynamically Modified Windows. *J. Chem. PHys.* **90,** 2460-2470 (1989).

86. D.A. Pearlman & P.A. Kollman. The overlooked bond-stretching contribution in free energy perturbation calculations. *J. Chem. Phys.* **94,** 4532-4545 (1991).

87. D.A. Pearlman. Determining the contributions of constraints in free energy calculations: Development, characterization, and recommendations. *J. Chem. Phys.* **98,** 8946-8957 (1993).

88. D.A. Pearlman. Free energy derivatives: A new method for probing the convergence problem in free energy calculations. *J. Computat. Chem.* **15,** 105-123 (1994).

89. D.A. Pearlman. A comparison of alternative approaches to free energy calculations. *J. Phys. Chem.* **98,** 1487-1493 (1994).

90. C. Chipot, P.A. Kollman & D.A. Pearlman. Alternative approaches to potential of mean force calculations: free energy pertubation versus thermodynamics integration. Case study of some reoresentative nonpolar interactions. *J. Comput. Chem* **17,** 1112-1131 (1996).

91. R.J. Radmer & P.A. Kollman. Free energy calculation methods: A theoretical and empirical comparison of numerical errors and a new method for qualitative estimates of free energy changes. *J. Computat. Chem.* **18,** 902-919 (1997).

92. G. Hummer. Fast-growth thermodynamic integration: Error and efficiency analysis. *J. Chem. Phys.* **114,** 7330-7337 (2001).

93. S.H. Fleischman & C.L. Brooks, III. Thermodynamic calculations on biological systems: Solution properties of alochols and alkanes. *J. Chem. Phys.* **87,** 221-234 (1988).

94. H.-A. Yu & M. Karplus. A thermodynamic analysis of solvation. *J. Chem. Phys.* **89,** 2366-2379 (1988).

95. B. Honig & A. Nicholls. Classical electrostatics in biology and chemistry. *Science* **268,** 1144-1149 (1995).

96. M.L. Connolly. Analytical molecular surface calculation. *J. Appl. Cryst.* **16,** 548-558 (1983).

97. 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).

98. 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).

99. 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).

100. 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).

101. 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).

102. 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).

103. S. Huo, I. Massova & P.A. Kollman. Computational Alanine Scanning of the 1:1 Human Growth Hormone-Receptor Complex. *J. Computat. Chem.* **23,** 15-27 (2002).

104. R. Radmer & P. Kollman. The application of three approximate free energy calculations methods to structure based ligand design: Trypsin and its complex with inhibitors. *J.*

Comput.-Aided Mol. Design **12,** 215-228 (1998).

105. S.R. Niketic & K. Rasmussen. In *The Consistent Force Field: A Documentation*, New York: Springer-Verlag, (1977).
106. C. Cerjan & W.H. Miller. *J. Chem. Phys.* **75,** 2800 (1981).
107. D.T. Nguyen & D.A. Case. On finding stationary states on large-molecule potential energy surfaces. *J. Phys. Chem.* **89,** 4020-4026 (1985).
108. G. Lamm & A. Szabo. Langevin modes of macromolecules. *J. Chem. Phys.* **85,** 7334-7348 (1986).
109. J. Kottalam & D.A. Case. Langevin modes of macromolecules: application to crambin and DNA hexamers. *Biopolymers* **29,** 1409-1421 (1990).
110. R.M. Levy, M. Karplus, J. Kushick & D. Perahia. Evaluation of the configurational entropy for proteins: Application to molecular dynamics simulations of an α-helix. *Macromolecules* **17,** 1370-1374 (1984).
111. 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).
112. 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).
113. P. Cieplak, W.D. Cornell, C. Bayly & P.A. Kollman. Application to the multimolecule and multiconformational RESP methodology to biopolymers: Charge derivation for DNA, RNA and proteins. *J. Computat. Chem.* **16,** 1357-1377 (1995).
114. 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).
115. A. Vedani & D.W. Huhta. *J. Am. Chem. Soc.* **112,** 4759-4767 (1990).
116. D.L. Veenstra, D.M. Ferguson & P.A. Kollman. *J. Computat. Chem.* **13,** 971-978 (1992).
117. F.H. Allen, O. Kennard, D.G. Watson, L. Brammer, A.G. Orpen & R. Taylor. *J. Chem. Soc. Perkin Trans. II* S1-S19 (1987).
118. 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).
119. A.J. Hopfinger & R.A. Pearlstein. *J. Compuat. Chem.* **5,** 486 (1985).
120. J.F. Cannon. *J. Computat. Chem.* **14,** 995-1005 (1993).
121. 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. Computat. Chem.* **16,** 1483-1506 (1995).
122. 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).

Index

This index is designed to help locate information for particular variable names. The Table of Contents should be used to identify subject areas.