AMBER
Version 5

Volume 1

Amber 5 is a collaborative effort of the research groups of Peter Kollman (UCSF), David Case (Scripps), Ken Merz(Penn State), Tom Darden (NIEHS), and Dave Fergunson (Minnesota).

The authors are:

David A. Case (The Scripps Research Institute)
David A. Pearlman (Vertex Pharmaceuticals)
James W. Caldwell (UCSF)
Thomas E. Cheatham III (UCSF,NIH)
Wilson S. Ross (UCSF)
Carlos Simmerling (UCSF)
Tom Darden (NIEHS)
Kenneth M. Merz (Penn State)
Robert V. Stanton (UCSF)
Ailan Cheng (Penn State)
James J. Vincent(Penn State)
Mike Crowley (PSC)
David M. Ferguson (University of Minnesota)
Randall Radmer (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)

Acknowledgements

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, the NIH, and the NSF for Peter Kollman is gratefully acknowledged, as is support from the NIH for David Case. Use of the facilities of the UCSF Computer Graphics Laboratory (Thomas Ferrin, PI) is appreciated. We thank Nelson H.F. Beebe of the University of Utah for permission to include his ‘‘portable namelist’’ code. Wendy Cornell contributed a discussion of charge derivation to the manual and added demos and documentation for the RESP program. We also thank Allison Howard and Valerie Daggett for various helpful discussions and suggestions. The pseudocontact shift code was provided by Ivano Bertini of the University of Florence. Many people helped add features to various codes; these contributions are described in the documentation for the individual programs. Jed Pitera, in particular, made a large contribution to the documentaion for PROFEC in Version 5.

Recommended Citations:

When citing Amber Version 5 in the literature, the following citation should be used:

> D.A. Case, D.A. Pearlman, J.W. Caldwell, T.E. Cheatham III, W.S. Ross, C.L. Simmerling, T.A. Darden, K.M. Merz, R.V. Stanton, A.L. Cheng, J.J. Vincent, M. Crowley, D.M. Ferguson, R.J. Radmer, G.L. Seibel, U.C. Singh, P.K. Weiner and P.A. Kollman (1997), AMBER 5, University of California, San Francisco.

The basic description of the methods incorporated in Amber is 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).

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. The term *amber* is also sometimes used to refer to the empirical force field that is implemented here. It should be recognized however, that the code and force field are separate: several other computer packages have implemented the *amber* force field, and other force fields can be implemented with the *amber* programs. Further, the force field is in the public domain, whereas the codes are distributed under a license agreement.

Amber 5 (1997) represents a significant change from the most recent previous version, *4.1*, which was released in 1995. Briefly, the major differences include:

(1) an updated and parallelized implementation of the particle-mesh Ewald routine, and its incorporation into the free energy module;

(2) "locally-enhanced sampling" (LES) code that allows parts of the system to be present as multiple copies;

(3) an alternate version of Sander (ROAR) that includes the ability to define part of the system as a quantum-mechanical section (QM/MM), and includes alternate integrators;

(4) PROFEC (Pictorial Representation of Free Energy Changes), a set of tools for carrying out and displaying extrapolative free energy changes;

(5) new and parallelized methods for NMR refinement; incorporation of penalites based on pseudocontact shifts.

(6) updates to the functionality and stability of LEaP.

1.1. What to read next.

If you are installing this package or want to redimension the code, see *installation* section of this manual. New users should continue with this *introduction* section, and will also find the *tutorial* section useful. The directories under amber5/demo contain a number of systems that may serve as examples. You should familiarize yourself with the files in the database directory, amber5/dat, by looking over the *database* section of the manual. There is also lots of information, tips, and examples on the Amber Web pages at `http://www.amber.ucsf.edu`; a portion of this is included in the distribution tape: point your browser to *amber5/Web/index.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. Hundreds of people have learned to use Amber; don't be easily discouraged.

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. You first need to understand what information is needed by the energy programs (*gibbs, sander, spasms, 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 all the energy programs need:

(1) Cartesian coordinates for each atom in the system.

(2) "Topology": connectivity, atom names, atom types, residue names, and charges.

(3) Force field: Parameters for all of the bonds, angles, dihedrals, and atom types in the system.

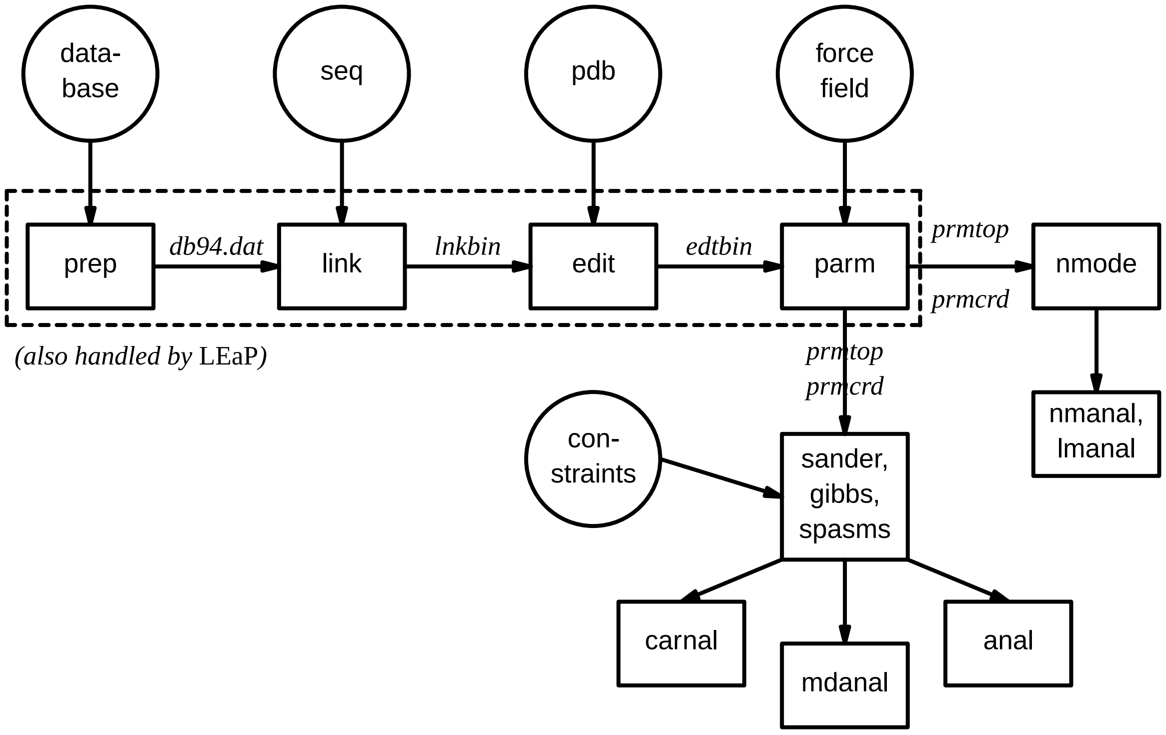
(4) Commands: The user specifies the procedural options and state parameters desired.

This information is provided to the energy programs in three files: One contains the coordinates; the second contains the topology and parameters, and is called the "topology file"; the command or "input" file is the third file. Additional files may needed for special options specified in the command file.

Cartesian coordinates usually come from Xray crystallography, NMR spectroscopy, or model-building. They should be in Brookhaven 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.

Topology comes from the database: The database is found in the *amber5/dat* directory. It is called *db94.dat*, and is described in Chapter 3. It contains topology for the standard amino acids as 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".

The basic *force field* parameters are also found in the *amber5/dat* directory; the *database* section of the manual contains some detailed descriptions of various force field options. This file 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.



Basic information flow in AMBER

1.3. Preparatory programs.

LEaPis 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` outlined below. These latter programs are retained primarily for backward compatibility with older versions of Amber.

PREP

creates or adds to a residue database from the appropriate topology/parameter information. Required for residues not already defined in the standard AMBER database. (As supplied, the standard AMBER database contains definitions for the 20 standard amino acids, nucleic acids with the five standard bases, and a few other units).

LINK

deals only with topology. You tell LINK the residue sequence of your molecule (even if there is only one residue). LINK will extract the topology information for each residue from the standard AMBER database or, optionally, from the residue database files created with PREP. The topology for each residue will be linked together to form the topology for the system. This is written to a binary file (default name = *lnkbin*) that is read by EDIT.

EDIT

deals mainly with coordinates. One of the primary purposes of EDIT is to read PDB coordinates and apply them to the system generated by LINK. Coordinates for atoms that are missing from the PDB file (usually hydrogens) will in most cases be generated automatically by EDIT, using the stored internal coordinates in the link binary file link.bin. EDIT can be used to solvate a molecule in water, to add counterions to nucleic acid systems, or to alter coordinates in specific ways.

PARM

will determine which bonds, angles, dihedrals, and atom types exist in the system, and extract the appropriate parameters for them from the force field file. PARM then writes the final coordinate and topology files needed by all other AMBER programs. used to add simple non-varying internal coordinate restraints to the system, and to create a two-state topology file for use in GIBBS free energy perturbation calculations. PARM outputs two files which are used subsequently: The topology file (default name = *prmtop*); and the coordinate file (default name = *prmcrd*).

PROTONATE

This program will add hydrogens in appropriate locations to peptides and proteins that lack them. It can also check the suitability of protons that are already present, and convert from one naming system to another (*e.g.* from IUPAC-IUB recommendations to Brookhaven format.)

1.4. Energy 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. Structures should usually be minimized before molecular dynamics simulation. The molecular dynamics portion generates configurations of the system by integrating Newtonian equations of motion. MD will sample more configurational space than MIN, and will allow the structure to cross over small potential energy barriers. For complicated systems MD is usually able to locate lower energy conformations than simple energy minimization. Configurations may be saved at regular intervals during the simulation for later analysis.

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 or PARM.

NMODE

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

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.

1.5. 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. Anal can also be used for more sophisticated analyses of energy and structure.

CARNAL
: is a molecular dynamics analysis program. It is used for geometrical measurements, root mean square coordinate fitting, trajectory averaging, and other structural analyses of MD trajectories. CARNAL executes a programming language for filtering, measuring and comparing multiple streams of coordinate files (the language contains 44 keywords and uses 10 punctuation/logical characters). As an example, one can use it to build a trajectory in which the solute is positioned for minimum root mean square fit of residues in the active site and only the first shell of waters is included.

RDPARM
: is a general purpose utility that can examine and modify prmtop files created by LEaP or PARM. It can also process trajectory files created from MD simulations, carrying out superpositions, extractions of coordinates, etc.

NMANAL/LMANAL
: computes atomic fluctuations and various correlation functions from normal modes.

2. Installation of Amber 5

(1) Compile the basic AMBER distribution:

```
a. go to the src directory below this one.

b. create a link to the appropriate Machine file, e.g.
       ln -s -f Machine/Machine.hp  MACHINE   for HP, or
       ln -s -f Machine/Machine.sgi MACHINE   for SGI, etc.

c. ./Makeall >& make.errs &
   look at the make.errs file for programs that didn’t get made;
   loader warnings (especially on SGI -- see the Machine.sgi file)
   can generally be ignored; compiler warnings should be considered,
   but most are innocuous.
```

(2) Compile LEaP:

```
a. go to the leap directory below this one.

b. xmkmf;  make World;  make install.leap;

   [Note: on some Digital Unix machines, the second step above may
    need to be: make "DNETLIB=-ldnet_stub" World ]

c. Set your LEAPROOT environment variable using the leapSetup.sh
   or leapSetup.csh scripts in amber5/Leap (depending on whether
   you use the Boune or C shells).
```

(3) Compile ROAR:

```
a. go to the roar/source directory below this one.

b. edit the Makefile, to uncomment the lines appropriate for your machine.

c. make install
```

(4) Make the database files:

```
a. go to the dat directory below this one.

b. make
```

(5) Make Interface (optional):

```
a. go to the interface directory below this one.

b. source install_int; source install_ambint.
```

(6) Test the basic AMBER distribution

```
a. go to the test directory below this one.

b. ./Run.tests >& tests.errs &

   Examine the tests.errs file: where "possible FAILURE" messages are
   found, go to the indicated directory under amber5/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.

   The "standards" to which your output will be compared are contained
   in the demo dirctory.  The "default" values were run at UCSF on
   an HP-735 workstation.  You should expect to see slighly different
   detailed MD trajectories on other machines, especially in gibbs,
   although the "statistics" (which are the real answers) should be
   very close to our results.  The transition-state theory test may
   give a large number of differences -- see the comments in the output
   of that test to help interpret it.

c. go to the leap/test directory below this one

d. ./Run >& tests.errs &

   Again, examine the tests.errs file for indications of more than
   round-off error differences.

e. There is no automated test for gibbs with PME; if you are interested
   in this option, please look at the files in
   amber5/demo/sodium_gibbs_pme.

f. There is no automated test for addles; if you are interested in this
   option, please look at the files in amber5/demo/addles.
```

2.1. Installing by hand.

If your system is not included among the configuration files supplied, or if you want to alter the existing file or are curious how these files hide machine dependencies, see the file `amber5/src/Machine/0README`. If you are developing or changing the `MACHINE` file, you may want to go more slowly, compiling and testing. You can type `make` or `make install` in any src/ directory to make the program(s) in that directory, e.g. when redimensioning arrays or otherwise modifying the code. `make clean` will cause all .o files to be removed; otherwise they stay around using significant space to conserve recompilation time.

2.2. Testing.

We have installed and tested AMBER 5 on a number of machines, using Cray, IBM, Sun, Hewlett-Packard, DEC, Convex, and Silicon Graphics hardware. 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, and to a lesser extent molecular mechanics, 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.

When comparing the output from two different machines for purposes of verification, it is very important that identical input files be used to generate both sets of output. The validation suite uses matched inputs and outputs in the amber5/demo/ tree, which is set to read-only to help you avoid overwriting them with files created on your machine. Testing takes place in the amber5/test/ tree.

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 to see small divergences in behavior after as little as 1-200 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.

2.3. Memory Requirements.

The AMBER 5 programs as distributed are dimensioned for a fairly large system (about 10K atoms), and you may want to change their dimensions to be more appropriate for the machine you are using if you are running in a tight memory environment. See `src/0README` for information on resizing. Some programs use local scripts called resize.csh; this script uses the stream editor sed, and employs regular expression matching to correctly redimension the code regardless of what its dimensions are currently set to, even if the same parameter has been inadvertently set to different values in different modules. Here we describe dimensioning of sander as an example. Sander can also be compiled the MEM_ALLOC switch set in the MACHINE file. This causes arrays to be allocated dynamically, based on the size of the molecule. If this works on your machine, it can eliminate most of the resizing effort described below.

In src/sander/sizes.h, you will find the following parameters that might need to be changed:

```
parameter (MAXINT=1300000)
parameter (MAXPR=2500000)
parameter (MAXREA=340000)
parameter (MAXHOL=200000)
parameter (MAXDUP=5000)
```

The actual memory requirements for a particular job can be determined from the output of SANDER or GIBBS. An annotated example follows:

```
1.  R E S O U R C E   U S E:


  Memory Use       Allocated      Used
  Real              340000        19109  <-- minimum MAXREA
  Integer          1300000        29904  <-- minimum MAXINT

  Max Nonbonded Pairs: 1270096 packed  1 to a machine word
                                       ^
                                    "NWDVAR"
  Duplicated   26 dihedrals

  Duplicated   94 dihedrals  <-- minimum MAXDUP
      .
      .
      .

   NB-UPDATE: NPAIRS =  150395  HBPAIR =    2804
```

As is shown above, MAXREA must be at least as large as the number of Real words used. It can be read directly off the output. MAXDUP must be at least as large as the larger of the two 'duplicated' values given, in this case 94. The actual amount of memory controlled by MAXDUP is 10 times its value. MAXINT is slightly more complicated, since it depends on the type of pairlist packing used. The number of NB pair pointers packed in a native integer word, NWDVAR, is printed in the output as shown. It will be 1 or 2 on byte-oriented machines, 1, 2, or 4 on 64 bit machines like Cray or FPS264. The minimum value of MAXINT is determined by the sum of the static integer requirement given in the output. For gibbs, MAXINT also includes the requirement for the pairlist, while in sander the pairlist size is determined by MAXPR. The pairlist requirement is the total number of nonbonded pairs, NPAIRS, divided by NWDVAR. Because the number of pairs may grow or shrink during a run, you should include a safety factor of 5-10% extra for NPAIRS. The algorithm to determine the MAXPR (sander) or MAXINT pairlist component is thus:

(NPAIRS/NWDVAR) * 1.1

More detailed documentation on memory use and packing configuration is found in sander/sizes.h. For gibbs, the variables which define memory allocation are MAXREA, MAXINT, and MAXCHR. MAXREA and MAXINT can be set as described above. MAXCHR allocates character storage, is typically small, and scales linearly with the number of atoms. These parameters are scaled to a single parameter, *memmax*.

3. The database directory.

There are two main types of force field file in the amber5/dat/ directory: residue descriptions for building the topolgy database, and force field files. The residue descriptions include topologies, atom types and charges and have `.in` extensions. The PARM force field files contains parameters mapped to the atom types: mass, Van der Waals, bond, angle, torsional and hydrogen bonding terms. These files have names matching the pattern, `parm*.dat`.

The following files are found in the database directory `amber5/dat/`:

DATABASE AND DATABASE INPUT FILES:

db94.dat	Residue database for the 1994 force field.
	(needs to be created by "make db94.dat")
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.
db4.dat	Residue database for the 1991 force field.
	(needs to be created by "make db4.dat")
uni.in	United atom database input.
unict.in	United atom database input, COO- Amino acids.
unint.in	United atom database input, NH3+ Amino acids.
all.in	All atom database input.
allct.in	All atom database input, COO- Amino acids.
allnt.in	All atom database input, NH3+ Amino acids.
nacl.in	Ion file
opls_uni.in	Normal OPLS residues.
opls_unict.in	OPLS COO- Amino acids.
opls_unint.in	OPLS NH3+ Amino acids.

FORCE FIELD PARAMETER FILES:

parm96.dat	modified version (see below) of 1994 force field
parm94.dat	1994 force field file.
parm91.dat	1991 force field file.
opls_parm.dat	OPLS force field file.

STANDARD PROGRAM INPUTS:

wat216.dat	Cube of 216 TIP4P waters, MC liquid.

```
    nucgen.dat         Nucgen nucleic acid conformations.
```

(1) **1994 parameters.** 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 the small molecules examined to date. The parameters in *parm94.dat* omit the hydrogen bonding terms of earlier force fields.

The main files in the amber5/dat/ directory that users normally need are `db94.dat` and `parm94.dat`. This is an all-atom force field; no united-atom counterpart is provided. 1-4 electrostatic interactions are scaled by 1.2 instead of 2.0; *users must make this adjustment in their input files for sander, gibbs etc. when using this force field.*

Charges for the old AMBER (Weiner *et al.*) force field were derived using the STO-3G basis set. The 6-31G* basis set was used for the new charges because it 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 at length in the Appendix.

Parm96.dat differs from parm94.dat in that the torsions for phi and psi have been modified in response to high level *ab initio* calculations performed by Friesner (JACS ?? and unpublished), in which they showed that the energy difference between conformations were quite different than calculated by Cornell *et al.* (using parm94.dat). To create parm96.dat, common V1 and V2 parameters were used for phi and psi, which were empirically adjusted to reproduce the energy difference between extended and constrained alpha helical energies for the alanine tetrapeptide studied by M.D. Beachy, D. Chasman, R.B. Murphy, T.A. Halgren and R.A. Friesner, *J. Am. Chem. Soc.* **119,** 5908-5920 (1997). 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 Friesner *et al.* This model (parm96.dat) is described by Dixon *et al.* in Vol. 3 of *Computer Simulation of Biomolecular Systems* A. Wilkinson, P. Weiner, W. Van Gunsteren, eds. (Elsevier, 1997, in press).

(2) **1991 parameters.** These parameters may still be useful for vacuum simulations of nucleic acids and proteins using a distance-dependent dielectric. The material in *parm91.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, not its default value of 2.0.

A number of terms in the non-bonded list of parm91.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).

For alkali ions with explicit waters, we have provided the values of Åqvist (J. Phys. Chem., 1990, 94: 8021-8024) which are adjusted for Amber's nonbonded atom pair combining rules to give the same ion-OW potentials, in order to reproduce the first peak of the radial distribution for ion-OW and the relative free energies of solvation in water of the various ions. These are included in the standard (STDA) parameter file. The atom types are:

QC: Cs+	QK: K+	QL: Li+	QN: Na+	QR: Rb+

(3) **OPLS.** The file *opls_parm.dat* is a parameter set appropriate for use with the AMBER/OPLS parameter set as described by Tirado-Rives and Jorgensen: W.L. Jorgensen and J. Tirado-Rives, *J. Am. Chem. Soc.* **1988,** *110*, 1657.

4. An Amber Tutorial.

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 do not use the *interface* programs, and only a 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. A separate introduction to *LEaP* is given in Chapter 5.

Additional tutorial examples are available on the Amber web page, at `http://www.amber.ucsf.edu`; a portion of this page is included in the distribution tape: point your browser to *amber5/Web/index.html.*

4.1. Example 1. Minimization of BPTI in vacuum.

Step 1. Generate some starting coordinates.

The first step is to obtain starting coordinates. We begin with the file *6pti.pdb*, exactly as distributed by the Protein Data Bank and Brookhaven. This file (as with most Brookhaven 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 need to have their residue names changed from CYS to CYX to reflect this. Let's call this modified file *6pti.mod.pdb.* Finally, hydrogen positions are not included, so we run the Amber program *protonate* to provide these:

```
protonate -d amber41/dat/PROTON_INFO < 6pti.mod.pdb > 6pti.H.pdb
```

In other situations, many different programs could be used to generate starting coordinates, but the basic ideas are the same: somehow generate what you want in a "pdb" format, then run the result through *protonate.* We recommend doing the last step even if protons are present, since protonate performs a number of checks on the correctness and naming of hydrogen atoms.

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 in either *tleap* or *xleap*:

```
bpti = loadPdb 6pti.H.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, distance-dependent dielectric
  &cntrl
    maxcyc=200, imin=1, cut=12.0, nsnb=20, idiel=0, scee=2.0, 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 ‘angstroms’ (`cut`) and a distance-dependent dielectric constant (`idiel`). The list of non-bonded pairs will be updated every 20 steps (`nsnb`), and 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 rest of this section documents using older AMBER modules to carry out the equilvalent of *Step 2*, above. These are included for completeness, but we encourage people *not* to continue doing things this way.

Step 2a. Run LINK to establish the topology.

The following script will accomplish this by creating an input file and running LINK with a prep database:

Running link for BPTI

```
cat <<eof >lnkin
 bpti


DU
    0    0    0    0    0
 bpti
P   1    0    1    3    1
ARG 2PRO  ASP  PHE  CYX  LEU  GLU  PRO  PRO  TYR  THR  GLY
PRO  CYX  LYS  ALA  ARG  ILE  ILE  ARG  TYR  PHE  TYR  ASN
ALA  LYS  ALA  GLY  LEU  CYX  GLN  THR  PHE  VAL  TYR  GLY
GLY  CYX  ARG  ALA  LYS  ARG  ASN  ASN  PHE  LYS  SER  ALA
GLU  ASP  CYX  MET  ARG  THR  CYX  GLY  GLY  ALA

    5   55SG  SG      0
   14   38SG  SG      0
   30   51SG  SG      0

QUIT
eof
#
link -i lnkin -o lnkout -p  $AMHOME/dat/db4.dat
/bin/rm lnkin
```

You should interpret the file given above using the input description for *link*. Basically, the first seven lines contain operation flags, many of which are almost always the same. The next four lines give the amino acid sequence, then come lines that establish cross-links (disulfide bonds) between residues 5-55, 14-38 and 30-51. The UNIX `AMHOME` variable should be set to the location of Amber on your system, and your `PATH` variable should allow the program *link* to be found. The above script will create a text output file *lnkout* (which you should read), and a binary file *lnkbin*, which will be used as input to the next step.

This step informs AMBER of the "topology" of the system: what all the atoms are called, what their "types" are (needed to set up a force field), and where all the bonds are. All this information was assembled from the sequence and the information about amino acids that is contained in the `db4.dat` file.

Step 2b. Run EDIT to insert the starting coordinates.

The following script will accomplish this:

Running edit for BPTI

```
cat <<eof > edtin
bpti, 5pti structure
    0    0    0    0
XYZ
OMIT
XRAY
    0    0    0    0    0
QUIT
eof
#
edit -i edtin -o edtout -pi 6pti.H.pdb
/bin/rm edtin
```

The XRAY option reads in a Brookhaven format file and compares the atoms in that file to what Amber expects to see; when it finds matches it inserts the proper coordinates into the system, and it reports errors when it fails to find matches. In this case, all the atoms are present, and no warning messages should be obtained.

The output from the above script will be a text file called *edtout* and a binary file *edtbin*, which will be used in the next step.

Step 2c. Run PARM to connect the force field to the protein.

This is done by this simple script:

Running parm for BPTI

```
cat <<eof >prmin
 name of system
BIN FOR STDA

    0

eof
#
parm -i prmin -o prmout  -f $AMHOME/dat/parm91.dat
/bin/rm prmin
```

In this step, the *parm* program looks through the molecular information in *edtbin* and determines all the types of "parameters" (force constants, bond lengths, non-bonded sizes, etc.) that are necessary to calculate the energy of BPTI. It then searches through the *parm91.dat* file to find the parameters. The program will complain if something is missing, but this is just a standard protein, and everything is in place. The output is a text file *prmout*, which you should read, and data files *prmtop* and *prmcrd* that will be used in the next step. The *prmtop* and *prmcrd* files are ascii files, so can be moved easily from one machine to another. (It is common to run *link*, *edit*, and *parm* on a workstation, then transfer the *prmtop* and *prmcrd* files to a more powerful computer for minimization and dynamics.) The *prmtop* file contains all the information needed to compute the energy of a molecule, and *prmcrd* contains the coordinates (in this case, the starting coordinates.) This division makes sense since minimization or dynamics will change the coordinates but not the make-up of the molecule.

You are now ready to go back to *Step 3*, above.

4.2. Example 2. Peptide with a non-standard residue.

As a second example, suppose you want to minimize an enzyme - substrate complex, and that you have a standard PDB file with coordinates for the enzyme and substrate, which we will call ’model.pdb’. Such a file might come from X-ray crystallography or model building. PDB files generally don’t contain connectivity information, so this must be provided. In addition, each atom of the system must be assigned the appropriate AMBER atom type so the correct force field parameters will be applied. For the amino acid residues of the protein, this connectivity and atom type information already exists, keyed by residue name in the AMBER database, and need not be specified. However, the substrate will require that you input connectivity and atom type information. This is done using the program PREP. The input for prep consists of only one file, in this case subprep.in. Running PREP will give you two output files. One of the files, sub.res, is a residue topology file for your substrate. It will have the same format as the amino acid residues in the standard data- base. The other file, prep.out, is a list of diagnostic information.

Amber programs are usually run through the use of command files. In the following examples, one will see that the control file for each program is named *filename.in*. The output file containing user information and diagnostics is called *filename.out*. The binary topology files that are passed from module to module are called *filename.bin*, where filename is the name of the module that created it. Residue files from prep have names ending in "*.res*", pdb files have names ending in "*.pdb*", and coordinate files created by AMBER are usually named *name.crd*. It is not essential that these naming conventions be adhered to but it will facilitate communication with other AMBER users. The following two files are the command and input files that create the substrate residue file using PREP.

Running PREP

Unix:

```
~amber41/exe/prep -i subprep.in -o prep.out
```

VMS:

```
$ set default [yourdir.tet]
$ assign subprep.in for005
$ assign prep.out for006
$ run [amber41.exe]prep
```

Here is the "subprep.in" file; strip comments before using.

```
0 0 1                !control for database generation
                     !blank card
substrate            !title
sub.res              !name of output file
SUB INT   0          !control parameters - see PREP.DOC
CORRECT OMIT DU  BEG

 1 DUMM DU M 0 0 0 0.   0.  0.  0.
 2 DUMM DU M 1 0 0 1.449  0.  0.  0.
 3 DUMM DU M 2 1 0 1.522 111.1  0.  0.
 4 N  N M 3 2 1 1.335 116.6 180. -0.5200
 5 HN  H E 4 3 2 1.01 119.8  0. 0.2480
 6 CA  CH M 4 3 2 1.449 121.9 180. 0.2270
 7 CB  C2 S 6 4 3 1.525 111.1  60. 0.0390
 8 CG  C2 S 7 6 4 1.525 109.47 180. 0.0530
 9 CD  C2 S 8 7 6 1.525 109.47 180. 0.0480
10 CE  C2 S 9 8 7 1.525 109.47 180. 0.2180
11 NZ  N3 3 10 9 8 1.47 109.47 180. -0.2720
12 HNZ1 H3 E 11 10 9 1.01 109.47 60. 0.3110
13 HNZ2 H3 E 11 10 9 1.01 109.47 180. 0.3110
14 HNZ3 H3 E 11 10 9 1.01 109.47 300. 0.3110
15 C  JJ M 6 4 3 1.522 111.1 180. 0.5260
16 O  O2 E 15 6 4 1.229 120.5  0. -0.5000



IMPROPER
-M  CA  N  H
CA  +M  C  O
CB  CA  N  C

DONE
STOP
```

The next step is to link the appropriate residues from the standard database, along with the residue file you created with PREP into a macromolecule. This is done using the program LINK. Note that if you were only interested in the enzyme and not the substrate, you would start at this point. The third line of *link.in* tells LINK that the topological information for residue "SUB" is in the file sub.res (the "standard" residues are retrieved from the prep database file specified with the ‘−p’ argument). The residues of the enzyme are listed sequentially in the order that they are to be bonded. The substrate residue is put at the end, separated by the spacer residue "***" indicating that it is not covalently attached. Alternatively it could be specified as a separate molecule. After the residue sequence, disulfide crosslinks are input. Any desired covalent attachment can be input as a crosslink. See LINK.DOC for details. Running LINK again produces two files: *link.bin*, the molecular topology file, and *link.out*, which contains diagnostic information.

Running LINK

Unix:

```
˜amber41/exe/link -i link.in -o link.out -l link.bin -p db4.dat
```

VMS:

```
$ SET DEFAULT [YOURDIR.TET]
$ ASSIGN [AMBER.DAT]DB4.DAT FOR001
$ ASSIGN LINK.IN FOR005
$ ASSIGN LINK.OUT FOR006
$ ASSIGN LINK.BIN FOR010
$ RUN [AMBER41.EXE]LINK
```

LINK.IN:

```
TACK’S PROTEIN       !title
                     !blank line
SUB   0sub.res       !filename for residue SUB
                     !blank card
DU                   !symbol for dummy atoms
  0  0  0  0  0      !print controls
tacks protein        !subtitle for first molecule
P  1  0  1  3  1     !control parameters for first molecule
ASP 1SER  CYX  GLU  ALA  ILE  ILE  HIP  GLU  LEU  HID  SER
ARG  HID  PRO  GLY  ASP  PHE  GLY  ALA  ASP  ALA  GLN  GLY
ALA  MET  ASN  LYS  ALA  CYX  GLU  SER  ***  SUB   !residue list
                     !blank card
  3  30SG SG   0     !crosslink info
                     !blank card
QUIT                 !exit control
```

The binary file *link.bin* contains your system, but at this point it lacks the correct atomic coordinates. It does contain the internal coordinates for each residue, but the residues are linked with arbitrary dihedral angles. The file also contains some pseudo atoms called "dummy" atoms. They are there to define the space axes for the internal coordinate system and must be removed. The addition of correct coordinates and removal of dummy atoms is accomplished with the program EDIT. Input for EDIT consists of a small control file, *edit.in*; the topology file from LINK, *link.bin*; and your PDB file. Two files are output: *edit.bin*, the molecular topology file (now with correct coordinates and dummy atoms removed), and *edit.out* containing user information and diagnostics. A look at *edit.out* generally reveals some frightening diagnostics stating that input for some atoms was not found. What this actually means is that the PDB file was missing some atoms present in the database residues, or had some extra atoms not present in the database (sometimes these are the same atoms, with different names). If atoms in the PDB file are missing, edit will add them using the stored internal coordinates of the residues. In the event that this can't be done (notably for the very first atoms in a molecule), the correct orientation of the atoms can be specified on *edit.in* using the "ABC" option of EDIT. Extra atoms in the PDB file are ignored. Some important notes: EDIT expects the residue sequence of the pdb file to match the link input file. If any residues are missing or extra ones are present, the program will stop with an error message. The ordering of atoms within a residue does not matter, nor do the atom sequence numbers, however, all atom records for a given residue should have the same residue sequence number.

Running EDIT

Unix:
```
˜amber41/exe/edit -i edit.in -o edit.out -l link.bin
      -e edit.bin -pi model.pdb -po edit.pdb
```

VMS:
```
$ set default [yourdir.tet]
$ assign edit.in for005
$ assign edit.out for006
$ assign link.bin for010
$ assign edit.bin for012
$ assign model.pdb for015
$ run [amber41.exe]edit
```

EDIT.IN:

```
tacks protein          !title
  0  0  0  0           !print controls
XYZ                    !select xyz option
OMIT                   !xyz input - omit dummy atoms
XRAY                   !select xray option
  0  0  0  0           !xray input
ABC                    !select abc option
  1    0               !abc input
  1  1.01 109.5 60.0  2  5  6   !abc input
  3  1.01 109.5 180.0  2  5  6  !abc input
  4  1.01 109.5 300.0  2  5  6  !abc input
                       !blank card
   QUI                 !terminate abc option
QUIT                   !terminate edit
```

All that remains to be done is add force field parameters to the molecular topology file, and you will be ready to run either molecular mechanics or molecular dynamics. Force field parameters are added with the program PARM. Input for PARM consists of a control file; *parm.in*, a parameter file; *parm91.dat*, and the topology file from EDIT; *edit.bin*. *parm91.dat* is part of the AMBER 4 distribution. Output from PARM consists of the completed topology file, *prmtop*, a coordinate file, *prmcrd*, and the diagnostics file *prmout*. The finished topology and coordinate files can be written either in binary form or formatted form. In general we now use only the formatted form for all files, so they can be used on various machines regardless of the underlying representation of data. It is important to look at parm.out to make sure that all the needed parameters were found in *parm91.dat*. If you are only working with amino acids, nucleic acids, or water, they should all be there. However, it is very easy to construct a molecule for which parameters do not exist in *parm91.dat*. In that event you will have to create some on your own. Often parameters for similar bonding situations can be found in *parm91.dat*, and simply duplicated in that file with the appropriate atom types. An overview of parameter generation is contained in Appendices C and D.

Running PARM

Unix:

```
˜amber41/exe/parm -i parm.in -o prmout
     -e edit.bin -f amber5/dat/parm91.dat
     -c prmcrd -p prmtop
```

VMS:

```
$ SET DEFAULT [YOURDIR.TET]
$ ASSIGN PARM.IN FOR005
$ ASSIGN PARM.OUT FOR006
$ ASSIGN [AMBER41.DAT]PARM91.DAT FOR010
$ ASSIGN PARM.TOP FOR012
$ ASSIGN EDIT.BIN FOR015
$ ASSIGN PARM.CRD FOR018
$ RUN [AMBER41.EXE]PARM
```

PARM.IN:

```
tack helix      !title
BIN FOR STDA    !format controls + nonbon param set name
  0  0  0       !print flags
  1  1  1       !print flags
```

Now you are finally ready to run SANDER, the molecular mechanics/dynamics module. Input for this program consists of a control file, *sander.in*, the topology file from PARM; *prmtop*, and the coordinate file from PARM; *prmcrd*. Output from SANDER consists of the final coordinates, *minmd.crd*, and a record of the molecular mechanical energies of the system as the minimization/dynamics proceeds, *minmd.out*. Output from a dynamics run may optionally include files containing the dynamics trajectory and velocities of all the atoms of the system over the course of the simulation.

SANDER.COM: This file will typically be submitted to a batch queue, or run in the background at reduced priority.

Running SANDER

SANDER.IN:
This file uses the *namelist* style of input.

```
# 500 steps min, constant dielectric
  &cntrl
   imin = 1, maxcyc = 500, nrun = 0, nsnb = 50,
   idiel = 1, cut = 8.0, scee = 2.0,
  &end
```

Unix:

```
sander -i sander.in -o minmd.out -p prmtop
     -c prmcrd -r minmd.crd -inf minmd.inf
```

VMS:

```
$ SET DEFAULT [YOURDIR.TET]
$ ASS SANDER.IN FOR005
$ ASS MINMD.OUT FOR006
$ ASS PARM.BIN FOR020
$ ASS COORD.DAT FOR021
$ ASS COORD.OUT FOR033
$ RUN [AMBER41.EXE]SANDER.EXE
```

4.3. Example 3. A more complicated protein example.

This section works through in some detail setting up a protein simulation in AMBER. The example is for plastocyanin in water, and contains a number of things that experienced AMBER users know how to do, but which may be far from obvious for others. In particular, there are a number of items that go beyond a simple protein. The examples assume you have an environment variable `$amber5` that points to the top of your AMBER distribution.

(1) Plastocyanin contains a metal ion bound to four amino acids, and I also want to modify a methionine residue that is bound to the copper in such a way that it has a different type of sulfur than is found in the standard database.

(2) The Brookhaven crystallographic file (1PLC) contains crystallographic waters, which I might want to keep. Only the oxygen positions are provided, so I will need to try to figure out where to put protons.

(3) Somewhat unusually, this PDB file has proton positions for the protein, which I would like to keep. However, Brookhaven uses proton names that are different than what NMR spectroscopists use, and I would like to be able to use the latter to make easy contact with NMR results.

(4) Using the most probable ionization states of the protein (at neutral pH) results in a protein with a net charge of -8, so I would like to include mobile counterions in the solution to create an overall neutral system.

This will be a lot of work, but it's infinitely easier now in AMBER than it used to be. We will also use this example to show how to set up some constraints, such as might be found in a NMR refinement, and will illustrate how to carry out simulated annealing optimizations.

4.3.1. Make database file for the modified residues.

For plastocyanin, I will define two new types of residues: HIC, which will be a histidine coupled to a copper ion, and which will take the place of HIS 37 in the "real" sequence, and MEM, which is a modified methionine where the sulfur atom is of type "SM" rather than type "S". "SM" is a type I made up, and will use to create special force field parameters for MET 94, which is bonded to the copper ion with a fairly long bond.

Here are the input files for these two residues:

hicu.in

```
    0    0    2

HISTIDINE PLUS
hicu.db4
 HIC  INT     1
 CORR OMIT DU   BEG
   0.00000
   1  DUMM  DU    M    0  -1  -2     0.000     0.000     0.000   0.00000
   2  DUMM  DU    M    1   0  -1     1.449     0.000     0.000   0.00000
   3  DUMM  DU    M    2   1   0     1.522   111.100     0.000   0.00000
   4  N     N     M    3   2   1     1.335   116.600   180.000  -0.34790
   5  H     H     E    4   3   2     1.010   119.800     0.000   0.27470
   6  CA    CT    M    4   3   2     1.449   121.900   180.000  -0.13540
   7  HA    H1    E    6   4   3     1.090   109.500   300.000   0.12120
   8  CB    CT    3    6   4   3     1.525   111.100    60.000  -0.04140
   9  HB2   HC    E    8   6   4     1.090   109.500   300.000   0.08100
  10  HB3   HC    E    8   6   4     1.090   109.500    60.000   0.08100
  11  CG    CC    S    8   6   4     1.510   115.000   180.000  -0.00120
  12  ND1   NB    B   11   8   6     1.390   122.000   180.000  -0.15130
  13  CU    CU    E   12  11   8     2.050   126.000     0.000   0.38660
  14  CE1   CR    B   12  11   8     1.320   108.000   180.000  -0.01700
  15  HE1   H5    E   14  12  11     1.090   120.000   180.000   0.26810
  16  NE2   NA    B   14  12  11     1.310   109.000     0.000  -0.17180
  17  HE2   H     E   16  14  12     1.010   125.000   180.000   0.39110
  18  CD2   CW    S   16  14  12     1.360   110.000     0.000  -0.11410
  19  HD2   H4    E   18  16  14     1.090   120.000   180.000   0.23170
  20  C     C     M    6   4   3     1.522   111.100   180.000   0.73410
  21  O     O     E   20   6   4     1.229   120.500     0.000  -0.58940

LOOP
 CG   CD2

IMPROPER
 -M   CA   N    H
 CA   +M   C    O
 CE1  CD2  NE2  HE2
 CG   NE2  CD2  HD2
 ND1  NE2  CE1  HE1
 ND1  CD2  CG   CB

DONE
STOP
```

mem.in

```
    0    0    2

METHIONINE, with SM atom type for the sulfur
mem.db4
 MEM  INT     1
 CORR OMIT DU   BEG
   0.00000
   1  DUMM  DU    M    0  -1  -2     0.000     0.000     0.000   0.00000
   2  DUMM  DU    M    1   0  -1     1.449     0.000     0.000   0.00000
   3  DUMM  DU    M    2   1   0     1.522   111.100     0.000   0.00000
   4  N     N     M    3   2   1     1.335   116.600   180.000  -0.41570
   5  H     H     E    4   3   2     1.010   119.800     0.000   0.27190
   6  CA    CT    M    4   3   2     1.449   121.900   180.000  -0.02370
   7  HA    H1    E    6   4   3     1.090   109.500   300.000   0.08800
   8  CB    CT    3    6   4   3     1.525   111.100    60.000   0.03420
   9  HB2   HC    E    8   6   4     1.090   109.500   300.000   0.02410
  10  HB3   HC    E    8   6   4     1.090   109.500    60.000   0.02410
  11  CG    CT    3    8   6   4     1.525   109.470   180.000   0.00180
  12  HG2   H1    E   11   8   6     1.090   109.500   300.000   0.04400
  13  HG3   H1    E   11   8   6     1.090   109.500    60.000   0.04400
  14  SD    SM    S   11   8   6     1.810   110.000   180.000  -0.27370
  15  CE    CT    3   14  11   8     1.780   100.000   180.000  -0.05360
  16  HE1   H1    E   15  14  11     1.090   109.500    60.000   0.06840
  17  HE2   H1    E   15  14  11     1.090   109.500   180.000   0.06840
  18  HE3   H1    E   15  14  11     1.090   109.500   300.000   0.06840
  19  C     C     M    6   4   3     1.522   111.100   180.000   0.59730
  20  O     O     E   19   6   4     1.229   120.500     0.000  -0.56790

IMPROPER
 -M   CA   N    H
 CA   +M   C    O

DONE
STOP
```

I made *mem.in* just by copying the relevant portions of the methionine entry from *all_amino94.in* in the database directory, changing the atom type of the sulfur, and adding appropriate first and last lines. Similar things were done for the histidine residue (starting from the library's HIP residue), except that I added a copper atom bonded to ND1. It is a good idea to read these files alongside the PREP documentation.

Then, these files were used as input to PREP:

```
prep -i hicu.in -o hicu.prpout -p hicu.params
prep -i mem.in -o mem.prpout -p mem.params
```

This creates the files *hicu.db4* and *mem.db4*, which describe these modified residues that will be incorporated into our protein.

4.3.2. Do some editing of the PDB file.

Several small changes need to be made to the input PDB file to make it work with Amber:

(1) First, we need to split of the HOH water residues into a separate file, say *watpdb*, and remove them from the main PDB file (call this modified file *1plc.nowat.pdb*). Further, the remarks in this pdb file indicate that waters #183 and #187 are a disordered pair, and should not both be present. So, I arbitrarily choose to delete #187 and to keep #183. For some reason, these two disordered waters were both put in the PDB file, and not assigned as alternate conformers. Note that AMBER by default will also choose only the principal position for disordered side chains, *i.e.* the "A" conformation if there is more than one. But this is done automatically, and does not require editing. If you want to start a simulation from the "B" conformation of a side chain, you need to manually edit the PDB file to remove the "A" conformation and blank-out the alternate conformation flag for the atoms you want AMBER to use. Generally speaking, you have to look carefully at a Brookhaven file before really using it.

(An alternative is to simply strip out the "crystallographic" waters and not use them at all. This is most appropriate if you are planning an MD or free energy simulation that will go through an extensive equilibration period before the "real" simulation begins. One goal of equilibration is to minimize dependence upon the starting conditions, and certainly the individual water molecules will move around a lot during any decent equilibration. At that point, the fact that you went to some trouble to originally place the waters in some nice positions may be irrelevant. Or maybe not; opinions differ on this matter, which is why we try to provide flexible tools in Amber. For this tutorial, we will not use the "crystallographic" waters in our starting structure.)

One final change involves residue names. Brookhaven files do no distinguish between cysteine residue that are involved in bonds to other things (and hence which have no proton on the sulfur atom) and "free" residues that do have such a proton. Molecular mechanics studies need to make this sort of distinction. Since residue 84 in plastocyanin has the sulfur atom bonded to the copper ion, I changed its name from CYS to CYX. Similar comments apply to histidines: molecular mechanics studies need to know (or guess, or assume) whether the histidine has a proton bonded at the ‘delta‘ position (HID), at the ‘epsilon‘ position (HIE), or at both (HIP). This is pretty easy for plastocyanin, since the two histidine side chains are both bound to copper through the ND1 nitrogen. So we initially change both HIS residues to HIE, in order to tell AMBER to put the protons on the NE2 nitrogen. (Note that in many other proteins, it will often be reasonable to assign surface-accessible histidines to be protonated, residue name HIP.)

(2) Next, we need to work on the proton names in the main protein file. Most Brookhaven crystallographic files do not have protons, so the *protonate* program is used to add them. Even here, we want to change the names Brookhaven uses to NMR conventions as described above, so we will still use *protonate*. This program also does sanity and chirality checking, so it is generally a good idea to use it prior to putting any pdb file into Amber. Now run:

csh:
```
( protonate -k -d PROTON_INFO < 1plc.nowat.pdb
    > 1plc.nowat.H.pdb) >& protonate.out
```

sh:
```
protonate -k -d PROTON_INFO < 1plc.nowat.pdb
    > 1plc.nowat.H.pdb 2> protonate.out
```

The *-k* option changes the names but "keeps" the positions of the protons in the input pdb file. As in other examples, you need to use the location of the PROTON_INFO file on your machine in place the the location listed above.

(3) Next, I moved the copper ATOM card from the end of the pdb file into residue 37, changing its residue name to "HIC" and its residue number to 37. I also changed the residue name for the rest of atoms of residue 37 from "HIE" to "HIC", and changed the residue name for residue 92 from "MET" to "MEM" as described above. I call this file *1plc.protein.pdb*.

4.3.3. Set up the system without solvent.

AMBER provides two ways to set up the files needed to carry out minimization or molecule dynamics. The original ("old") way runs the programs *link*, *edit*, and *parm*, each of which needs a separate input file. The "new" way involves running the program *LEaP* (which stands for "link, edit and parm"). Most new users will run *LEaP*, but it won't hurt to skim over the next section, which describes the "old" way, since the ideas required are very similar in the two approaches.

4.3.3.1. (Alternative 1): Create link, edit and parm files.

AMBER gets the sequence information, plus information about how the copper ion is bound to its ligands, from the input files to LINK:

lnkin.nowat

```
PLASTOCYANIN

HIC      2hicu_all.db4
MEM      2met.db4

DU
    0    0    0    0    0
Plastocyanin (poplar)
P   1    0    1    3    1
ILE 2ASP  VAL  LEU  LEU  GLY  ALA  ASP  ASP  GLY  SER  LEU  ALA  PHE  VAL  PRO
SER  GLU  PHE  SER  ILE  SER  PRO  GLY  GLU  LYS  ILE  VAL  PHE  LYS  ASN  ASN
ALA  GLY  PHE  PRO  HIC  ASN  ILE  VAL  PHE  ASP  GLU  ASP  SER  ILE  PRO  SER
GLY  VAL  ASP  ALA  SER  LYS  ILE  SER  MET  SER  GLU  GLU  ASP  LEU  LEU  ASN
ALA  LYS  GLY  GLU  THR  PHE  GLU  VAL  ALA  LEU  SER  ASN  LYS  GLY  GLU  TYR
SER  PHE  TYR  CYX  SER  PRO  HIE  GLN  GLY  ALA  GLY  MEM  VAL  GLY  LYS  VAL
THR  VAL  ASN

   37   87CU  ND1     0
   37   84CU  SG      0
   37   92CU  SD      0

QUIT
```

Again, it is a good idea to compare this input to the descriptions in the manual. Note the the copper atom is already bonded to the ND1 atom of HIS37, and that crosslink commands to used to add three other ligands to it. Then run:

```
link -i lnkin.nowat -o lnkout.nowat
     -p /afs/psc/packages/amber/amber41/dat/db94.dat
```

where you must substitute the location of *db94.dat* on your machine for the file listed above. Study the output file *lnkout.nowat* to see if it looks like everything worked OK.

Next create a standard input for for edit:

edtin.nowat

```
poplar plastocyanin
    0    0    0    0
XYZ
OMIT
XRAY
    0    0    0    0    0
QUIT
```

and run:

```
edit -i edtin.nowat -o edtout.nowat -pi 1plc.protein.pdb
```

and look carefully at the output file. It is very common to find warning messages at this point, and they need to be cleared up, usually by minor re-editing of the input PDB file.

Finally, create a standard input file for *parm*:

prmin

```
  standard parm using parm94.dat
BIN FOR MOD4
    0    0    0
    1
```

We also need to make modifications to the AMBER force field to recognize the copper atom and the modified methionine residue. The best way to do this is not to edit the main force field file, but to create a *frcmod* file with changes specific to each project. Here is the one I created for plastocyanin:

frcmod.pcy

```
#  modifications to force field for poplar plastocyanin
MASS
SM 32.06
CU 65.36

BOND
NB-CU    70.000   2.05000       kludge by JRS
CU-S     70.000   2.10000       kludge by JRS
CU-SM    70.000   2.90000       for pcy
CT-SM   222.000   1.81000       met(aa)

ANGLE
CU-NB-CV   50.000     126.700    JRS estimate
CU-NB-CR   50.000     126.700    JRS estimate
CU-NB-CP   50.000     126.700    JRS estimate
CU-NB-CC   50.000     126.700    JRS estimate
CU-SM-CT   50.000     120.000    JRS estimate
CU-S -CT   50.000     120.000    JRS estimate
CU-S -C2   50.000     120.000    JRS estimate
CU-S -C3   50.000     120.000    JRS estimate
NB-CU-NB   10.000     110.000    dac estimate
NB-CU-SM   10.000     110.000    dac estimate
NB-CU-S    10.000     110.000    dac estimate
SM-CU-S    10.000     110.000    dac estimate
CU-SM-CT   50.000     120.000    JRS estimate
CT-CT-SM   50.000     114.700     met(aa)
HC-CT-SM   35.000     109.500
H1-CT-SM   35.000     109.500
CT-SM-CT   62.000      98.900    MET(OL)

DIHE
X -NB-CU-X    1       0.000     180.000       3.000
X -CU-SM-X    1       0.000     180.000       3.000
X -CU-S -X    1       0.000     180.000       3.000
X -CT-SM-X    3       1.000       0.000       3.000

NONBON
  CU       2.20      0.200
  SM       2.00      0.200
```

Crating a *frcmod* file is a bit of an art, since one is often faced with unknown parameters (such as force constants from copper to its ligands in the above example.) In this tutorial, I am just going over the mechanics of running AMBER, and make no claims about the validity or appropriateness of the above parameters. There is a big literature on parameter estimation, and users are encouraged to consult this when faced with unusual chemical environments.

Now, run *parm* with the above inputs:

```
parm -i prmin -o prmout.nowat
    -f $amber5/dat/parm94.dat
    -m frcmod.pcy -p prmtop.nowat -c prmcrd.nowat
```

Again, you need to put in the location of the *parm94.dat* file on your machine. Study the *prmout.nowat* file to see if it looks like things went OK. It might be typical to find "missing parameters" at this point, which means that the *frcmod* file does not contain all of the parameters you need; the missing ones will be identified in *parm* output.

You might also create a pdb file at this point with the new coordinates:

```
ambpdb -p prmtop.nowat -wrap < prmcrd.nowat > 1plc.protein.amber.pdb
```

The *-wrap* flag will make the output proton names more like those Brookhaven uses. Leave this flag off if you want the names in the output PDB file to be the internal AMBER proton names.

4.3.3.2. (Alternative 2): Use LEaP to set up the required files.

It is simpler to carry out the above procedure in *LEaP*. Let's start by setting up an input file:

leap.in

```
PARM94 = loadamberparams frcmod.pcy
loadAmberPrep mem.lib
loadAmberPrep hicu.lib
plc = loadPdb 1plc.protein.pdb
bond plc.87.ND1 plc.37.CU
bond plc.84.SG plc.37.CU
bond plc.92.SD plc.37.CU
saveAmberParm plc prmtop.nowat prmcrd.nowat
```

By default, *LEaP* will read in the standard AMBER 94 force field libraries. The first line in the file above merges in the extra material from the *frcmod.pcy* file described above. The next two lines load in the files describing the modified residues HIC and MEM. The a molecule, named "plc" is created by reading in the appropriate pdb file. (*LEaP* will often complain at this point if it finds something it doesn't understand or doesn't like; a typical task would be to modify the pdb file and try again.) Next, the three "cross-links" that connect residues 84, 87 and 92 to the copper atom are added via the `bond` command. Finally, the `saveAmberParm` command is used to create the required output files. Details of all of these commands can be found in the *LEaP* manual.

The file above is read into *LEaP* as follows; ">" is the prompt that *LEaP* provides:

```
tleap
> source leap.in
> quit
```

4.3.4. Run a simple minimization.

We start with a very simple minimization with restraints to keep the protein from moving too far. In the examples below, do not include the comments in parenthesis in your actual files.

min1.in

```
Minimization with Cartesian restraints
  &cntrl
    imin=1, maxcyc=200,           (invoke minimization)
    scee=1.2, idiel=0, cut=12.0,  (force field options)
    nsnb=20,                      (update non-bonded list)
    ntpr=5,                       (print frequency)
    ntr=1,                        (turn on cartesian restraints)
  &end
Group input for restrained atoms
 1.0                              (force constant for restraint)
RES 1 99                          (all atoms in residues 1-99)
END
END
```

To run this file, use the following command:

```
sander -i min1.in -c prmcrd.nowat -p prmtop.nowat -ref prmcrd.nowat
    -o min1.out -r min1.xyz
```

4.3.5. Run simulated annealing optimization.

At this point, one often would want to carry out a more robust optimization, using a dynamical simulated annealing protocol. One also might add some sort of constraints at this point. In this example, we will illustrate how NMR-derived constraints might be incorporated; constraints from other sources of information would be handled in a similar fashion.

simul_anneal.in

```
 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)
  &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)

simul_anneal.in (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 restraint file referenced above (*RST.f*) would ordinarily hold hundreds to thousands of constraints based on experimental information. The constraints would keep the protein near its native conformation even though we have heated to a very high temperature. Examples of such constraint files are given in the SANDER section of the Users' Manual. Since a refinement like this can take a long time to run, we have not actually included files for it in the tutorial directory: the example given above can serve as a guide for real calculations that you might want to carry out.

4.3.6. Setup the system with counterions in a box of water.

Next, we turn from "vacuum" simulations to consider how to set up and carry out molecular dynamics simulations in a box of solvent water, using periodic boundary conditions. This example is typical of many molecular dynamics simulations begin carried out with AMBER.

4.3.6.1. Work on positioning counterions.

The question of how or whether to include solvent counterions in protein and DNA simulations is a difficult one. Generally speaking, DNA simulations have often used counterions and many existing protein simulations have not. In terms just of "mechanics" and not science, AMBER will suggest counterion positions for you, by using the *cion* program:

csh:

```
    ( cion -elstat -p prmtop.nowat < 1plc.protein.amber.pdb > cion.pdb)
        >& cion.out
```

sh:

```
    cion -elstat -p prmtop.nowat < 1plc.protein.amber.pdb > cion.pdb
        2> cion.out
```

This routine places counterions at the most favorable electrostatic positions, until it achieves a neutral overall system. Note, however, that this may end up corresponding to a fairly high salt concentration, and may not be at all what you want. At this stage, the user's judgment is required, which is why a lot of this stuff is not yet automated. This tutorial can't go over all of the pros and cons of various choices, and in any event, different users will have different preferences. Let's suppose that you choose a few sodium counterions to add to the simulation. For this example, we will choose the first eight counterions, in order to neutralize the -8 charge on the protein itself. Then, you need to run LINK again, using an input something like the following:

lnkin.cion

```
PLASTOCYANIN

HIC      2hicu.db4
MEM      2mem.db4

DU
    0    0    0    0    0
Plastocyanin (poplar)
P   1    0    1    3    1
ILE 2ASP  VAL  LEU  LEU  GLY  ALA  ASP  ASP  GLY  SER  LEU  ALA  PHE  VAL  PRO
SER  GLU  PHE  SER  ILE  SER  PRO  GLY  GLU  LYS  ILE  VAL  PHE  LYS  ASN  ASN
ALA  GLY  PHE  PRO  HIC  ASN  ILE  VAL  PHE  ASP  GLU  ASP  SER  ILE  PRO  SER
GLY  VAL  ASP  ALA  SER  LYS  ILE  SER  MET  SER  GLU  GLU  ASP  LEU  LEU  ASN
ALA  LYS  GLY  GLU  THR  PHE  GLU  VAL  ALA  LEU  SER  ASN  LYS  GLY  GLU  TYR
SER  PHE  TYR  CYX  SER  PRO  HIE  GLN  GLY  ALA  GLY  MEM  VAL  GLY  LYS  VAL
THR  VAL  ASN

   37   87CU  ND1     0
   37   84CU  SG      0
   37   92CU  SD      0

Eight sodium counterions:
P   0    0    1    3    0
CIP 2***  CIP  ***  CIP  ***  CIP  ***  CIP  ***  CIP  ***  CIP  ***  CIP

QUIT
```

Note the use of the "***" residue to indicate that the counterions are not chemically bonded to each other. Now run:

```
rm lnkbin edtbin
link -i lnkin.cion -o lnkout.cion
   -p $amber5/dat/db94.dat
```

where you must again substitute the location of *db94.dat* on your machine for the file listed above.

Study the output file *lnkout.cion* to see if it looks like everything worked OK.

4.3.6.2. Run EDIT and PARM to add a box of waters around the system.

Actually, the hard part is mostly done. At this point we would manually add the top eight counterions from the file *cion.pdb* to the bottom of the file *1plc.protein.pdb*, (call this new file *1plc.cion.pdb*) and then would run edit to read in the PDB file that has counterions, and to add a box of water molecules around the solute. Here is a sample input file for that:

edtin.wat

```
poplar plastocyanin
    0    0    0    0
XYZ
OMIT
XRAY
    0    0    0    0    0
BOX
HW   OW       4
  0.417     2.8       2.3
  12.0      12.0        12.0
QUIT
```

The edit command is:

```
edit -i edtin.wat -o edtout.wat -pi 1plc.cion.pdb
    -b $amber5/dat/wat216.dat
```

This creates a box with a minimum of 10 Å between the protein and the edge of the box. There end up being 3336 water molecules surrounding the protein. This is not a large value, since with counterions you need to be sure that there is enough room for them to move around if they need to.

Running PARM with counterions and water is no different than without, so at this point you need to repeat the PARM step outlined above, except for changing the names of the output files:

```
parm -i prmin -o prmout.wat
    -f $amber5/dat/parm94.dat
    -m frcmod.pcy -p prmtop.wat -c prmcrd.wat
```

If everything went well, you will have a parameter file (*prmtop.wat*), and a coordinate file (*prmcrd.wat*). These would be used to start an equilibration procedure, followed by an MD or free energy simulation.

4.3.6.3. Run solvated molecular dynamics simulation.

We will first run a simple minimization in water to remove initial bad contacts:

min2.in

```
 molecular dynamics run
&cntrl
   imin=1,
   scee=1.2, idiel=1, cut=9.0,
   ntb=1, ntc=2, nsnb=25,
   maxcyc=500, ntpr=25,
&end
```

Here is the command to run:

```
sander -O -i min2.in -c prmcrd.wat -p prmtop.wat
    -o min2.out -r min2.xyz &
```

(This is run in background, since it will take a few minutes to run; Next, try out a short molecular dynamics run; actual "production" computations would include many, many more MD steps than is given here.

md1.in

```
molecular dynamics run
 &cntrl
    imin=0, irest=0, ntx=1, tempi=0.,       (no restart MD)
    scee=1.2, idiel=1, cut=9.0,             (force field options)
    ntt=1, temp0=300.0, tautp=0.2,          (temperature control)
    ntp=2, taup=0.2,                        (pressure control)
    ntb=2, ntc=4, ntf=2, nsnb=25,           (SHAKE, periodic bc.)
    nstlim=500,                             (run for 0.0005 nsec)
    ntwe=100, ntwx=100, ntpr=25,            (output frequency)
 &end
```

Here we will start from the output of the minimization step to carry out the dynamics:

```
sander -O -i md1.in -c min2.xyz -p prmtop.wat
    -o md1.out -r md1.xyz -x md1.crd -e md1.ene &
```

The output will include the *md1.out* file giving information about the progress of the trajectory, along with *md1.crd* and *md1.ene* files that contain the coordinates and energy information at every 100th step, respectively. Many of the analysis programs in AMBER can use these sorts of files.

LEaP

5. LEaP

5.1. Preface

LEaP is the generic name given to the programs *tLEaP* and *xLEaP*, which are more commonly known as *tleap* and *xleap*. 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 AMBER software modules it replaces: prep, link, edit, and parm. Thus, *LEaP* can be used to prepare input for the AMBER and SPASMS molecular mechanics programs. However, the utility of *LEaP* is not limited to this task.

The *LEaP* User's Manual is intended to provide a comprehensive description of the programs *tLEaP* and *xLEaP* from a user's perspective. After reading the manual, the user ought to be able to run either the *tLEaP* or *xLEaP* program to obtain meaningful results. Also, the user should be able to understand the methods used by *LEaP* during the process of constructing molecules and generating input for other computational chemistry programs.

There are very few prerequisites for understanding *LEaP* or this manual. We have tried to write the manual so a person with a minimal background in chemistry and/or computers can understand the material. The material is frequently presented in more than one format to aid in comprehension.

It was our goal to write a user manual that provided a thorough and comprehensive presentation of the *LEaP* program. The information in this manual includes:

• examination of the concepts involved in running *LEaP*

• the use of *LEaP* in creating input for the AMBER and SPASMS programs

• discussion and comparison of the terminal and X-windows versions

• explanation of all chemistry-related algorithms used in *LEaP*

• descriptions and examples of each command

• several practical examples of running *LEaP* to build specific molecules.

Researchers using *LEaP* should use the following citation when referencing the program: Christian E. A. F. Schafmeister, Wilson S. Ross and Vladimir Romanovski, *LEaP*, University of California, San Francisco (1995). *LEaP* was developed in Peter A. Kollman's laboratory at the University of California−San Francisco. Its development was supported, in part, by the Defense Advanced Research Projects Agency under contract N00014-86-K-0757 administered by the Office of Naval Research (Robert Langridge, PI.). The software and documentation are provided to users pursuant to a license agreement containing restrictions on its disclosure, duplication, and use.

It is our intention to provide regular updates to this manual. However, in order to do so, we need input from users of *LEaP*. We would appreciate receiving reports regarding mistakes or omissions in the manual. Furthermore, if the user finds parts of the manual difficult to understand or poorly written, we would appreciate being made aware of the problems. Users can direct electronic mail to `amber-request@cgl.ucsf.edu`.

We gratefully acknowledge the Computer Graphics Laboratory (*cgl*: Robert Langridge, PI) for providing code that was used as the foundation for the pdb routines in *LEaP*. Financial support from the Defense Advanced Research Projects Agency under contract N00014-86-K-0757 administered by the Office of Naval Research (Robert Langridge, PI.) is appreciated. We wish to thank W. L. Jorgensen for providing the Monte Carlo equilibrated box of TIP3P water which is included in the *LEaP* distribution.

5.2. Introduction

LEaP is the general name for a program that performs as a workbench for computational chemists. In its current version, *LEaP* allows researchers to prepare input for the molecular modeling programs AMBER and SPASMS. Thus, the program replaces the prep, link, edit, and parm programs that are distributed with AMBER. It was felt that a replacement was needed that had all such functionality within a single program, that incorporated a simple, consistent user interface, and that would incorporate graphics to allow users to see the systems they were preparing.

The inspiration to write a program to replace the preparation modules of AMBER was first provided by Erik-Robert Evensen while he was at UCSF. Such a program, MainLine, was also written by him. *LEaP* was conceived and written by Christian E. A. F. Schafmeister, working in the laboratory of Peter A. Kollman at the University of California−San Francisco. David A. Rivkin contributed some features and Wilson S. Ross and Vladimir Romanovski have assisted extensively to prepare the program for distribution.

Both *tLEaP* and *xLEaP* are written in vanilla Kernighan & Ritchie (KR) C. *tLEaP* and *xLEaP* share much of the same code and there are approximately 90,000 lines in the *LEaP* distribution software files. About 10,000 of those lines are in C header files and 75,000 in C and yacc source code files. *xLEaP* contains about 3,000 additional lines of code in resource files to utilize the X-windows interface. The *LEaP* source directory contains approximately 200 files. About 48% are source files, 40% are header files, and 12% are X-windows resource files. *tLEaP* does not support graphics and therefore, it will run on any teletype terminal attached to any machine that supports a KR C compiler. *xLEaP* is meant to run on any machine that supports KR C and X-windows (Version 11 Revision 4 and latter versions).

xLEaP 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
Add bond, angle, and torsion restraints to molecules
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.
```

LEaP supports the following input and output file formats:

(1) Object File Format (OFF): This is a general file format developed for *LEaP* but applicable to any scientific computing problem. *LEaP* is capable of both reading and writing these files.

(2) AMBER PREP input files: The *LEaP* program can read and write these files. Note that when reading, it ignores improper torsions in these files (used to keep things like amino groups planar), instead generating its own impropers from connectivity and whatever force field information is available when AMBER PARM output files are written. When writing PREP files, *LEaP* adds *all* possible improper torsions, leaving it to the user to decide which are needed. (See Appendix C for a description of improper torsions.)

(3) AMBER PARM formatted parameter set files (parm.dat): The formatted parameter set files have been used as an input file for the AMBER program PARM. These files can be read by *LEaP*.

(4) AMBER PARM output files: The AMBER PARM output files are coordinate and topology files that are used as input to the programs AMBER and SPASMS. The *LEaP* program can generate these files (prmtop and prmcrd).

(5) PDB files: *LEaP* is capable of reading and generating these files.

5.3. Concepts

In order to effectively use LEAP it is necessary to understand the philosophy behind the program. The philosophy which guides LEAP is developed in this section. This is done by exploring the concepts of LEAP commands, variables, and objects. Once the user understands how commands, variables, and objects are defined and employed within LEAP, they will have also learned the principles necessary to use the program effectively. In addition to exploring these concepts, this section also addresses the use of external files and libraries with the program.

5.3.1. 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 commas. 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. In the following manual sections, we discuss variables and objects. It is important that the user be able to differentiate between variables and objects in order to properly use the LEaP command line interface.

5.3.2. 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. Each variable is associated with one object. Variables may also be assigned to the object represented by some other variable. For example, suppose that you wanted to create a RESIDUE called AIB (aminoisobutyric acid) using LEaP. Since this amino acid differs from ALA by only one substituent, you might decide to start with the ALA UNIT loaded from the "lib/all_amino94.lib" file and edit this to add one methyl group to the ALA RESIDUE. To implement this idea, you might enter the command line:

```
AIB = ALA
```

After executing the above statement, `AIB` and `ALA` will "point to" the same (single) object. This means more than simply saying `AIB` and `ALA` have the same contents. At this point, there will only be one UNIT object; both `AIB` and `ALA` will represent that one object. If the contents of the object are changed at some later time (such as editing AIB to add a methyl substituent), then the change will be seen in both `AIB` and `ALA`. Clearly, this would *not* be a good idea for this specific example. Instead, in this case one could use the LEaP `copy` command to create a duplicate of the `ALA` object. The strategy of creating an equivalent relationship between objects is largely used to prevent unnecessary duplication of objects, some of which can be *very* large. **NOTE:** equivalencing for residues, effective only when loading PDB files, is also achieved by the `addResidueNameMap` command, and the use of alternate names ("AIB = ALA") may be discontinued in future.

5.3.3. 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.

5.3.3.1. NUMBERs

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

5.3.3.2. 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. STRINGS may also be represented by prefixing a sequence of characters by a dollar sign, where the delimiter is a comma, a space, a semicolon, or a list open or close character ({ or }). If a string does not contain a comma, a space, a semicolon, or a list open or close character and there is no variable defined that is the same as that string, then it is not necessary to put the string in quotes or prefix it by a dollar sign. Double quote characters within the STRING may be represented by a pair of double quotes. Example strings are:

```
"Hello there"
"String with a "" (quote) character"
"Strings contain letters and numbers:1231232"
$noQuotes
$343noQuotesAgain
noQuotesOrDollarSign
```

5.3.3.3. 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" $anotherString }
{ 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.

5.3.3.4. 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.`

5.3.3.5. 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 that 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.

Both the AMBER and SPASMS software packages support a type of calculation know 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.

5.3.3.6. RESIDUEs

RESIDUEs are complex objects that contain ATOMs. RESIDUEs are collections of ATOMs that 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.

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.

When residues are read from AMBER PREP input files, LEAP creates a RESIDUE object for each residue read and defines the first main chain atom of the AMBER residue to be the `connect0` ATOM of the RESIDUE. The last main chain atom of the AMBER residue becomes the `connect1` ATOM of the RESIDUE. Any other atoms that would be used for cross links must be explicitly defined as connect ATOMs using the `set` command. The scripts in "leap/lib/" show how this is done for the standard force field residues.

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 RESIDUEss `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.

`connect3`

This is an ATOM property which defines an ATOM that can be used in making links to other RESIDUEs.

`connect4`

This is an ATOM property which defines an ATOM that can be used in making links to other RESIDUEs.

`connect5`

This is an ATOM property which defines an ATOM that can be used in making links to other RESIDUEs.

`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 `"sac-charide"`. 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 `"sol-vent"`. 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.

5.3.3.7. 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 or SPASMS. 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.

UNITs also contain information about restraints. *Users are encouraged to avoid applying such restraints in LEAP, and instead to use the more robust ones available in the simulation programs. Restraints are supported in LEAP only for backward compatibility.*

Restraints can be modified using the LEAP commands: `addBondRestraint`, `addAngleRestraint`, `addTorsionRestraint`, and `removeRestraint`. Restraints are additional energy terms that can be placed between two, three or four ATOMs. There are three kinds of restraints: bond, angle, and torsion restraints. Bond restraints can be created between any two ATOMs, and they are defined by the two ATOMs, an equilibrium distance, and a force constant. Angle restraints are defined by three ATOMs, an equilibrium angle, and a force constant. Torsion restraints are defined by four ATOMs, an equilibrium torsion angle, a force constant and a multiplicity. See Appendix C, Parameter Development, for more details.

5.3.3.8. 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 filed parameters, such as the van der Waals well depth, have been included in "lib/parm94.dat".

5.4. 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. The file is used to customize the operation of the LEAP program. For example, in the "leaprc" file included in the distribution, commands have been added to load the standard AMBER force field parameter library and the TIP3P water residue, the main chain and terminating amino acid residue UNITs, and the nucleic acid residue UNIT libraries. The file also sets up PDB Name maps and creates aliases. The user can copy and customize the file included in the distribution. LEAP will look for this file in the user's current directory during start-up.

5.4.1. 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.

5.4.2. Log File

The command line interface allows the user to specify a log file that is used to log all input and output within the command line environment. The log file is named using the `logFile` command. The file has two purposes: to allow the user to see a complete record of operations performed by LEAP, and to help recover from (and recreate) program crashes. Output from LEAP commands is written to the log file at a verbosity level of `2` regardless of the verbosity level set by the user using the `verbosity` command. Each line in the log file that was typed in by the user begins with the two characters "`> `" (a greater-than sign followed by a space). This allows the user to extract the commands typed into LEAP from the log file to create a script file that can be executed using the `source` command. This provides a type of insurance against program crashes by allowing the user to regenerate their interactive sessions. An example of a command that works on UNIX systems and that will create a script to reenact a LEAP session is:

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

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

5.5. Using LEaP

In the next two sections, we describe how to use the tleap and xleap user interfaces. Strategies for using LEAP in research is discussed in a subsequent section: "Using LEAP With AMBER".

5.6. tLEaP

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). The user can employ the following terminal keys to edit or enter commands in the tleap interface:

`BACKSPACE`
The use of this key will cause the last character on the line to be deleted.

`uparrow/downarrow`
Go backwards and forwards in the command "history" (tleap only).

`RETURN/ENTER`
Accept the current line. If the current line is the last line of a block then accept the block and execute it. (See the manual section describing the "Universe Editor Command Window".)

`control-c`
Cancel the current block or terminate the current process.

`control-u`
Clear the current line.

An edited tleap session is shown below:

```
unix 1% tleap

Welcome to LEaP!
leaprc is /home/euler/ross/Leap/cmd/leaprc
Log file: ./leap.log
Loading parameters: /home/euler/ross/Leap/lib/parm94.dat
Loading library: /home/euler/ross/Leap/lib/water.lib
Loading library: /home/euler/ross/Leap/lib/ions94.lib
Loading library: /home/euler/ross/Leap/lib/all_nucleic94.lib
Loading library: /home/euler/ross/Leap/lib/all_amino94.lib
Loading library: /home/euler/ross/Leap/lib/all_aminoct94.lib
Loading library: /home/euler/ross/Leap/lib/all_aminont94.lib
```

```
     > desc ALA
       : :
     > quit
     unix 2%
```

In the first line, the user starts the execution of tleap. Several lines of information may then be displayed: the user is first greeted, and if the LEAP script file ("leaprc") contained any commands to load libraries, the execution of these load commands is confirmed. The user may enter commands at the tleap command line prompt ("`> `"). In the above edited example, the user entered two commands into tleap: "`desc ALA`" and "`quit`". After quitting the program, the user is returned to the Unix shell command line prompt.

5.7. xLEaP

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.

5.7.1. 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. Within the command window, the user can cut and paste text using the mouse buttons:

`LEFTBUTTON`
  Begin selection.

`RIGHTBUTTON`
  Continue selection.

`MIDDLEBUTTON`
  Append the selection to the last input line.

The command window has a scroll bar that allows the user to scroll back and review messages that have scrolled off the top of the window.

Because some commands within LEAP can be very long, the command window allows users to spread input for single commands over multiple lines. A block is defined as one or more lines of input that contain a single command and all of its arguments. LEAP uses two criteria to determine whether a line terminates a block.

- If the last character in a line is a backslash then the block is continued on the next line.

- If the block has any open lists, meaning if there are more "{" (open lists) than "}" (close lists), then the block will continue to grow in lines until all lists are closed. This is to facilitate entry of LEAP commands that use long lists as arguments (e.g., the `zMatrix` command). Note: from this it follows that, if typing many carriage returns seems to have no effect, an extra open left-hand brace is the probable cause.

5.7.1.1. Universe Editor Menu Bar

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.

5.7.2. 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.

5.7.2.1. Unit Editor Menu Bar

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

`Unit` pulldown

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` pulldown

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` pulldown

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 "()".

"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.

5.7.2.2. 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 options to complete any partial selection of rings, residues or molecules. menu options within the Selection menu.

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.

`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`

Choosing `Draw` is equivalent to choosing the default "Elements" atom in the next array of buttons; the initial default is carbon. While in the draw mode, the 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.

5.7.2.3. 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.

5.7.2.4. 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.

5.7.3. 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 Atom Properties Editor has three parts: the Menu Bar, Status Window, and Table Window.

5.7.3.1. Atom Properties Menu Bar

The Menu Bar has two pulldowns: `Table` and `Operations`.

`Table`

"Save" − save current table. This transfers the properties in the table to the ATOMs. The table is first checked, and if something is incorrect, a warning box pops up with the option to cancel the save.

"Save and quit" − save current table (as in "Save") and quit.

"Close table" − the table information is discarded and not applied to the ATOMs. If the table has been changed, a warning box pops up with the option to cancel the close.

`Operations`

"Find..." − pops up a dialog box to enter a string to search for. All fields in the table are searched, and if the string matches the contents of a field, that cell is highlighted.

"Find next" − find next instance of current search string.

"Check table" − check each field in the table for errors. Error information is printed in the Status Window. The first error in the table is highlighted.

"Go to next error" − highlight the next error from the last "Check table" command.

5.7.3.2. Atom Properties Status Window

The Status Window is used to display messages from the Atom Properties Editor.

5.7.3.3. Atom Properties Table Window

The Table Window contains the properties of the selected ATOMs in the UNIT. The properties can be edited by clicking the mouse left-button in a cell (clicking again to position the cursor within an existing string), and typing/backspacing to modify the contents. Also, a selection can be made by dragging the cursor with the left-button down, then pasted into other cells by positioning the cursor over the target cell and clicking the middle button.

The ATOM properties are:

`Name`

This is a case-sensitive STRING. 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 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`

This property is the monopole atomic charge centered on the ATOM. See the Appendices for more information on deriving these charges. The value can be any floating point number (e.g., 3.0, -1.0E-2).

`Element`

The atomic element provides a simpler description of the atom that 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. The element name is automatically generated when drawing new atoms.

`Perturb`

This is a flag that indicates whether to consider the ensuing perturbation information when building a perturbation topology file using the `saveAmberParmPert` command. The setting is "true" if the ATOM is to be perturbed, nothing or "false" otherwise. Unless one is setting up a perturbation, this aprt of the table can be ignored.

`Perturbed name`

This property is a case-sensitive STRING. The property is a unique identifier for an ATOM in its final state during a Free Energy Perturbation calculation. LEAP fills in the unperturbed name as a default. The perturbed name has no effect on calculations and is mainly useful as a reminder of what was intended.

`Perturbed type`

This property is the force field atom type of the perturbed species in a Free Energy Perturbation calculation. The atom type is used to assign force field parameters. The type can be any string from the force field.

`Perturbed charge`

This property is the monopole charge on an ATOM at the completion of a Free Energy Perturbation calculation. The value can be any floating point number (e.g., -1.0E-2, 3.0).

5.7.4. 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.

5.7.4.1. Parmset Table Editor Menu Bar

The Menu Bar has three pulldowns: `Table`, `Edit` and `Operations`.

`Table`

"Save" − save current table. This transfers the parameter definitions in the table to the PARMSET. The table is first checked, and if something is incorrect, a warning box pops up with the option to cancel the save.

"Save and quit" − save current table (as in "Save") and close the Parmset Editor.

"Close table" − the table information is discarded and not applied to the PARMSET. If the table has been changed, a warning box pops up with the option to cancel the close.

`Edit`

"Add row at end" − create a new, empty row at the end of the table.

"Insert row before selection" − insert a new, empty row before the row containing the currently-selected cell.

"Delete row" − delete row containing currently-selected cell.

`Operations`

"Find..." − pops up a dialog box to enter a string to search for. All fields in the table are searched, and if the string matches the contents of a field, that cell is highlighted.

"Find again" − find next instance of current search string.

"Check table" − check each field in the table for errors. Error information is printed in the Status Window. The first error in the table is highlighted.

"Go to next error" − highlight the next error from the last "Check table" command.

5.7.4.2. Parmset Table Editor Status Window

The Status Window is used to display messages from the Parmset Editor.

5.7.4.3. Parmset Table Window

The Table Window contains the parameters in a table format. The cell that is currently being edited is highlighted; clicking mouse left-button over a cell selects it.

5.8. Using LEaP With AMBER

This section focuses on concepts necessary to employ LEAP to produce input files for the AMBER molecular mechanics programs. First we discuss loading force field parameter files (PARMSETs) and residue libraries, and then the general strategy for using LEAP to create the coordinates and parameters necessary to run AMBER. The user should refer to the "Examples" section and the tutorials in under the "Web" part of the AMBER source tree for detailed analyses of utilizing LEAP for specific tasks.

There are two AMBER force fields, Weiner *et al.* 1984, 1986 and Cornell *et al.* 1995. The associated files happen to be named "91" and "94" respectively. Each force field consists of residue definitions in one or more files, and an accompanying parameter file, or PARMSET. The residue definitions and PARMSET must be compatible. The newer Cornell *et al.* force field is loaded by the default `leaprc` file. A `$LEAPROOT/cmd/leaprc.ff91` file is provided for loading the Weiner *et al.* force field; to use it, one would copy this file to `leaprc` in the current directory. You never want to have both force fields (or parts of both) loaded at once, because they use incompatible and overlapping atom type definitions.

5.8.1. PARMSETs

The mechanism for loading PARMSETs is the `loadAmberParams` command. The Weiner *et al.* PARMSET file for LEAP is `parm91X.dat`; the Cornell *et al.* one is `parm94.dat`. The default `leaprc` loads Cornell *et al.* as UNIT "parm94," while `leaprc.ff91` loads Weiner *et al.* as "parm91." The `list` command will show the PARMSET(s) that have been loaded along with all other objects currently existing in LEAP.

In addition to one of the standard PARMSETs, one may need to load extra parameters, either for residues not contained in these basic force fields, or in order to override standard parameters. This is done by making a `frcmod` file in a normal text editor and loading that as well (using `loadAmberParams`). The most recently-loaded parameters take precedence when there is a duplicate definition of *e.g.* an atom type, so a frcmod file can be used to alter standard parameters without the risk of modifying the standard file and losing track of what was changed.

NOTE that the nonbonded parameters in PARMSETs must be of the form $r*$ and ε. LEAP will not read nonbonded parameters of the form "A & C". Also, for every atom type "mass" definition in the parameter set, there must also exist a nonbonded parameter for that atom type. LEAP ignores polarizability values from AMBER PARM parameter sets.

When LEAP needs force field parameters for commands like `saveAmberParm` or `saveAmberParmPert`, it searches through the list of PARMSETs that are currently loaded from the most recently loaded to the oldest. As soon as LEAP finds a parameter that matches the configuration of atom types for which it is looking, it stops searching the PARMSET list.

Parameters for torsional terms are found by searching each loaded PARMSET in turn. Specific torsional parameters take precedence over general ones (*i.e.* parameters having wild-card atoms). Only an exact match will prevent searching all the PARMSETs.

5.8.2. UNIT Libraries

A set of residue libraries is provided for each force field; the file names include "91" for Weiner *et al.* and "94" for Cornell *et al.*. The latter ones are loaded by the default `leaprc` file. One can see a list of all the loaded residues using the `list` command. In this section, we list the UNITs found in the libraries and their names and aliases. When a UNIT is distributed with LEAP, all of the force field parameters needed to define the UNIT within AMBER are also found in the PARMSETs.

5.8.2.1. Amino Acid Residues

The following amino acid UNITs and their aliases are defined in the LEAP libraries.

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
Cysteine	CYS
Cystine, S--S crosslink	CYX
Glutamic acid	GLU
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

The UNIT/RESIDUE names and aliases listed above correspond to the AMBER all atom force field. (The Weiner *et al.* united atom force field has not been adapted for LEAP.) For each of the amino acids found in the LEAP libraries, there has been created an n-terminal and a c-terminal analog. The n-terminal amino acid UNIT/RESIDUE names and aliases are prefaced by the letter N (e.g. NALA) and the c-terminal amino acids by the letter C (e.g. CALA}. If the user models a peptide or protein within LEAP, they may choose one of three ways 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.

It should be noted that by convention amino acid sequences are written starting with the n-terminus. This same convention is used in LEAP, dictated by the atom order in the residue libraries.

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 `crossLink` 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.

5.8.2.2. 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."

5.8.2.3. Miscellaneous Residues

Miscellaneous Residue	unit/residue name
TIP3P water molecule	WAT, HOH, IP3
Periodic box of TIP3P water	WATBOX216
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 "water.lib". The default `leaprc` file assigns the variables HOH and IP3 to the WAT UNIT found in the OFF library file. Thus, if a PDB file is read and that file contains either the residue name HOH or IP3, the WAT UNIT will be substituted. (Note that PDB residue names are restricted to 3 characters.)

A periodic box of 216 waters (WATBOX216) is provided in the file "water.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.

5.8.3. Building a Molecule For Molecular Mechanics

Note that there are some web examples of using LEAP for various practical tasks under "PSC Tutorials" on http://www.amber.ucsf.edu/amber/.

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

- Any needed UNIT or PARMSET objects must be loaded;
- The molecule must be constructed within LEAP;
- The user must output topology and coordinate files from LEAP to use in AMBER.

5.8.3.1. Loading Objects

Before start-up, LEAP contains no objects. In the default configuration, standard PARMSET and UNIT residue libraries for the Cornell *et al.* force field are loaded by the default `leaprc` file in $LEAPROOT/cmd/. This file can be consulted or copied as a template for constructing a useful work environment.

The `saveOff` command is used to save constructed UNITs to libraries, and frcmod-style PARMSETs are constructed using a normal text editor. Both may be loaded to prepare for a session using `loadOff` and dAmberParams respectively.

Objects are loaded into LEAP either by the user typing in load commands interactively, or by placing appropriate load commands within a "leaprc" start-up file in the working directory.

5.8.3.2. Constructing the Molecule

There are several different methods of constructing molecules or UNITs within LEAP. If the user has an AMBER PREP file, the structure may be read in using the `loadAmberPrep` command. PDB files are read into LEAP using the `loadPdb` or `loadPdbUsingSeq` commands. It is also possible to construct a molecule structure manually using the `zMatrix` command or (most commonly) the xleap `edit` command. The user may use any combination of these methods to make molecules. Once a UNIT is created, it can be stored in an OFF library for subsequent use. Thus, if a user is building a polypeptide which includes one novel amino acid, they would load the OFF library of standard amino acids and create the novel amino acid residue through one of the abovementioned methods. This nonstandard amino acid residue UNIT could then be stored and reloaded at the beginning of future sessions.

5.8.3.3. Z-matrix Input

Let us examine several methods of constructing a water molecule within LEAP. One such method would be to build a water UNIT, which we will call WAT, by utilizing a Z-matrix input for structure. Note that this method is probably only convenient if one already has such a matrix; normally it is easier to draw the new residue usinthe Unit Editor.

In the following example, presented as if it were a special leaprc, the user constructs WAT by creating ATOMs, then a RESIDUE, and finally the WAT UNIT. A structure is applied to the UNIT by using internal coordinates given by a Z-matrix. In this illustration, the user does not define any `head` or `tail` atoms for the RESIDUE or any `connect` atoms for the UNIT. This is because WAT is not a residue in the chemical sense; the WAT UNIT will never be used as a substituent or monomer. Once the WAT UNIT is created, a topology file and a coordinate file are generated for molecular mechanics calculations. In this and subsequent illustrations, all input command lines are prefaced by the characters "`> `". The program output found in these listings is not prefaced by the characters.

```
> #
> #  Constructing the water molecule will be done through
> #  a build-up procedure.  First, ATOMs are created.  The
> #  names, types, and charges of the ATOMs are also set.
> #  We then define the elements that are associated with
> #  each ATOM variable:
> #
> o  = createAtom O  OW -0.834
> h1 = createAtom H1 HW  0.417
> h2 = createAtom H2 HW  0.417
> set o  element O
> set h1 element H
> set h2 element H
> #
> #  A new residue ("WAT") is created and the
> #  variable "waterResidue" represents
> #  this object.  Each of the ATOMs are then
> #  added to the RESIDUE and bonds are next placed
> #  between the ATOMs.  The bond orders are, by
> #  default, single:
> #
> waterResidue = createResidue WAT
> add waterResidue o
> add waterResidue h1
> add waterResidue h2
> bond h1 o
> bond h2 o
> #
> #  A new UNIT ("WAT") is created and "waterResidue" is placed
> #  within the UNIT. At this point, the RESIDUE topology
> #  is known.  A Z-matrix is read in so that the structure
> #  of the RESIDUE can be determined:
> #
> WAT = createUnit WAT
> add WAT waterResidue
> zMatrix WAT {
>      { H1 O 0.9572 }
>      { H2 O H1 0.9572 104.52 }
> }
> #
> #  The "WAT" UNIT is saved in an OFF library:
> #
> saveOff WAT examples.lib
Saving WAT.
Building topology.
Building atom parameters.
```

5.8.3.4. PDB File Input

Another method of constructing a molecule is to use a PDB file. This time, rather than first building the molecule atom-by-atom and adding bonds to create a template, we just load the PDB file and begin work on that.

```
> #
> #  Load the file; since it’s an unknown residue,
> #  there are lots of messages..
> #
> WAT = loadpdb Wat.pdb
Loading PDB file: ./Wat.pdb
Unknown residue: WAT   number: 0   type: Terminal/last
  -no luck
Creating new UNIT for residue: WAT sequence: 1
Created a new atom named: O within residue: .R<WAT 1>
Created a new atom named: H1 within residue: .R<WAT 1>
Created a new atom named: H2 within residue: .R<WAT 1>
  total atoms in file: 3
  The file contained 3 atoms not in residue templates
> #
> #  Add bonds
> #
> bondbydistance WAT
> #
> #  Make it solvent
> #
> set WAT.1 restype solvent
> #
> #  Set atom attributes (easier to do in the
> #  Atom Properties Editor)
> #
> set WAT.1.O  type OW
> set WAT.1.H1 type HW
> set WAT.1.H2 type HW
> set WAT.1.O  charge -0.834
> set WAT.1.H1 charge  0.417
> set WAT.1.H1 charge  0.417
> #
> #  The "WAT" UNIT is saved in an OFF library:
> #
> saveOff WAT examples.lib
Saving WAT.
Building topology.
Building atom parameters.
```

The user may want to model a molecule for which a PDB file exists and a LEAP UNIT has already been created and stored in an OFF library. In this case, it is only necessary to load a PARMSET, the UNIT, and PDB file into LEAP. It is important to replace the coordinates of the UNIT with those of the PDB file in order to ensure that the molecular structure assumes the conformation of interest to the user. To understand this last point, consider the construction of a protein. When the UNITs in LEAP are joined together to form the protein sequence, the resulting structure is linear. Replacing the Cartesian coordinates of the UNITs will allow the proper tertiary protein structure to be modeled. The following example illustrates this procedure:

```
> #
> #  Load a LEaP OFF library that contains the
> #  water UNIT:
> #
> loadOff examples.lib
> #
> #  Load a PDB file to obtain correct Cartesian
> #  coordinates:
> #
> wat = loadPdb Wat.pdb
Loading PDB file: ./Wat.pdb
  total atoms in file: 3
```

5.8.3.5. AMBER PREP Input

If an AMBER PREP file exists for the water molecule it can be used to create the water UNIT within LEAP. When the PREP file is loaded into LEAP, 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.

```
> #
> #  Load AMBER PREP file for water:
> #
> loadAmberPrep Wat.in
Loaded UNIT: 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
```

5.8.3.6. UNIT Editor Input

We have illustrated several methods of constructing a water molecule within LEAP. By far, the most convenient method is the one which we will now discuss. If the user runs the xleap program, the molecule can be created quite easily graphically within the Unit Editor.

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. Note that this WAT residue does not correspond to the TIP3P WAT residue that is loaded from `water.lib` since it lacks an H-H bond (used for keeping the molecule rigid with SHAKE).

```
> #
> #  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
```

5.8.3.7. Generating Molecular Mechanics Input Files

Once the UNIT is constructed, it should be examined using the `check` command. The UNIT may also be augmented in many ways, including adding counterions, restraints, or solvents.

Finally, the user needs to obtain topology and coordinate files. These files are used as input for AMBER. These two files are created by the `saveAmberParm` command. If the user constructed a UNIT to be used in a Free Energy Perturbation calculation, then the `saveAmberParmPert` command should be used instead.

5.9. 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.

5.9.1. 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.
```

5.9.2. 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.

5.9.3. 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. Otherwise, the specified numbers of *numIon1 numIon2* are added in alternating order. 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 angstroms beyond. A distance-dependent dielectric is used for speed.

5.9.4. 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.

5.9.5. 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.*

5.9.6. 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 }
          :  :
> }
```

5.9.7. 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.

The leaprc file that is found in the LEAP distribution tape creates the following aliases:

```
q    quit
exit    quit
e    edit
a    alias
?    help
l    loadOff
lp   loadPdb
so   saveOff
sap     saveAmberParm
sp   savePdb
ap   loadAmberParams
lap     loadAmberPrep
ai   addIons
s    source
```

The following line is an example of this command:

```
> alias q quit
```

5.9.8. alignAxes

```
alignAxes unit

      UNIT   unit
```

Translate the geometric center of *unit to the origin and align the principle axes of the ATOMs within unit* along the coordinate axes. This is done by calculating the moment of inertia of the UNIT using the identical mass for each ATOM, and then diagonalizing the resulting matrix and aligning the eigenvectors along the coordinate axes. This command modifies the coordinates of the UNIT.

The following example illustrates the alignAxes command. For the purposes of this manual, the CA ATOM of the all_amino94.lib UNIT GLY is described. The user should note the change in the CA Cartesian coordinates after alignment.

```
> desc GLY.GLY.CA
ATOM
Name:    CA
Type:    CT
Charge:  0.035
```

```
Element: C
Atom flags: 20000|posfxd- posblt- posdrn- sel- pert-
notdisp- tchd- posknwn+ int- nmin- nbld-
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.
> alignAxes GLY
> desc GLY.GLY.CA
ATOM
Name:    CA
Type:    CT
Charge:  0.035
Element: C
Atom flags: 20000|posfxd- posblt- posdrn- sel- pert-
notdisp- tchd- posknwn+ int- nmin- nbld-
Atom position: 0.124598, 0.610835, 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.
```

5.9.9. 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. (See the *add* command for an example of the *bond command.)*

5.9.10. 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
```

5.9.11. 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
```

5.9.12. 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
```

5.9.13. 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. 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.
```

5.9.14. clearPdbAtomMap

Clear the atom Name Map (see the `addPdbAtomMap` command).

5.9.15. clearPdbResMap

Clear the residue Name Map (see the `addPdbResMap` command).

5.9.16. clearVariables

```
clearVariables [ list ]
```

```
LIST   list
```

This command removes variables from LEAP. If the list argument is provided then only the variables in the LIST will be removed. If no argument is provided then all variables will be removed. Example:

```
> addPath /disk/howard/LeapTests/Ethane
/disk/howard/LeapTests/Ethane added to file search path.
> loadOff ETH.lib
Loading library: ETH.lib
Loading: ETH
> list
ETH
> desc ETH
UNIT name: ETH
```

```
Head atom: null
Tail atom: null
Contents:
R<ETH 1>
> clearVariables { ETH }
> list
> desc ETH
"ETH"
```

5.9.17. 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>
```

5.9.18. 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.

5.9.19. 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.)

5.9.20. createParmset

```
variable = createParmset  name
```

```
PARMSET variable
STRING  name
```

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

```
> newparms = createParmset pertParms
```

5.9.21. 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.)

5.9.22. 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.)

5.9.23. crossLink

```
crossLink res1 conn1 res2 conn2 [ order ]
```

```
RESIDUE res1
STRING  connect1
RESIDUE res2
STRING  connect2
STRING  order
```

Create a bond between ATOMs at the connection point specified by conn1 and conn2. The user may also specify the bond order of the crosslink through the order option. By specifying "-", "=", "#", or ":" as the optional argument, order, the user can specify a single, double, triple, or aromatic bond, respectively. The arguments conn1 and conn2 can have the following values:

Name	Alternative Names
connect0	nend firstend
connect1	cend lastend
connect2	send disulphide
connect3	-
connect4	-
connect5	-

Example:

```
> disulfide = sequence {
>    ALA CYX ALA ALA ALA ALA CYX ALA
> }
> crosslink disulfide.2 connect2 disulfide.7 connect2
> desc disulfide.2.8
ATOM
Name:    SG
```

```
Type:    S
Charge:  0.824
Element: S
Atom flags: 20000|posfxd- posblt- posdrn- sel- pert-
notdisp- tchd- posknwn+ int- nmin- nbld-
Atom position: 9.992386, 3.049907, -1.366045
Atom velocity: 0.000000, 0.000000, 0.000000
  Bonded to .R<CYX 2>.A<CB 5> by a single bond.
  Bonded to .R<CYX 2>.A<LP1 9> by a single bond.
  Bonded to .R<CYX 2>.A<LP2 10> by a single bond.
  Bonded to .R<CYX 7>.A<SG 8> by a single bond.
```

5.9.24. debugOff

```
debugOff filename

      STRING filename
```

This command is a system debugging function. It turns off debugging messages from the source file filename. The default for all filenames is OFF. Command example:

```
> debugoff /disk/howard/debug
Messages will be displayed from the files:
------
```

5.9.25. debugOn

```
debugOn filename

      STRING filename
```

This command is a system debugging function. It turns on debugging messages from the source file filename. The default for all filenames is OFF. Example:

```
> debugon /disk/howard/debug
Messages will be displayed from the files:
/disk/howard/debug
------
```

5.9.26. debugStatus

```
debugStatus [memory]
```

```
      STRING memory
```

This command is a system debugging command. It displays various messages that describe LEAP's usage of system resources. The optional string, memory, can have the following values:

```
      testMemoryOn    Turn memory testing on
      testMemoryOff   Turn memory testing off
```

Command example:

```
      > debugstatus
      Current memory usage: 0 bytes
      Memory testing on = FALSE
```

5.9.27. 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
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.
```

5.9.28. deSelect

```
deSelect object

        CONT   object
```

Clears the select flag on all ATOMs within object. (see the *select* command for additional information.)

```
        > deSelect ALA.1.5
```

5.9.29. displayPdbAtomMap

Display the atom Name Map (see the *addPdbAtomMap* command).

5.9.30. displayPdbResMap

Display the residue Name Map (see the *addPdbResMap* command).

5.9.31. 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. In tleap this command prints an error message.

5.9.32. 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.)

5.9.33. 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. This command is illustrated with the following example:

```
> help quit
    quit
Quit LEaP.
```

5.9.34. 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.

5.9.35. 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
```

5.9.36. listOff

```
listOff library
```

```
STRING library
```

List the UNITs/PARMSETs that are stored within the library. Command example:

```
> listOff all_amino94.lib
Index of library: all_amino94.lib
ALA
ARG
ASH
ASN
ASP
CYM
CYS
CYX
GLN
GLU
GLY
HID
HIE
HIP
ILE
LEU
LYS
MET
PHE
PRO
SER
THR
TRP
TYR
VAL
```

5.9.37. 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".

The LEAP distribution contains "old" and "new" AMBER force field parameters in files "parm91X.dat" and "parm94.dat". One could build OFF libraries using the commands shown below; at some point this may become a standard conversion, but since it is easier to maintain the parameters in the AMBER format, this procedure is not used in the default setup. "parm91X.dat" is used instead of the "parm91.dat" in the AMBER dat/ tree because this file has corrections for LEAP 's method of applying improper torsions.

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

5.9.38. 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
```

5.9.39. 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
```

5.9.40. 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 *crossLink* command.

5.9.41. 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".

5.9.42. 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
```

5.9.43. matchVariables

```
variable = matchVariables  string
```

```
LIST   variable
STRING string
```

The matchVariables command is used to create a LIST of variables with names that match string. The argument string can contain the wildcard characters "?" and "*" to match a specific character or multiple characters, respectively.

```
> cTerminal = matchVariables C???
> desc cTerminal
List size=24
CALA: CARG: CASN: CASP: CCYS: CCYX: CGLN:
CGLU: CGLY: CHID: CHIE: CHIP:
CHIS: CILE: CLEU: CLYS: CMET: CPHE: CPRO:
CSER: CTHR: CTRP: CTYR: CVAL:
--End of list
```

5.9.44. 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
```

5.9.45. quit

Quit the LEAP program.

5.9.46. 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>
```

5.9.47. removeBond

```
removeBond atom1 atom2
```

```
ATOM   atom1
ATOM   atom2
```

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

```
> dipeptide = sequence { ALA GLY }
Sequence: ALA
Sequence: GLY
Joining ALA - GLY
> desc dipeptide.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<O 10>
```

```
A<C 9>
A<HB3 8>
A<HB2 7>
A<HB1 6>
A<CB 5>
A<HA 4>
A<CA 3>
A<HN 2>
A<N 1>
> desc dipeptide.1.9
ATOM
Name:    C
Type:    C
Charge:  0.616
Element: C
Atom flags: 20000|posfxd- posblt- posdrn- sel- pert-
notdisp- tchd- posknwn+ int- nmin- nbld-
Atom position: 5.485541, 2.705207, -0.000004
Atom velocity: 0.000000, 0.000000, 0.000000
  Bonded to .R<ALA 1>.A<CA 3> by a single bond.
  Bonded to .R<ALA 1>.A<O 10> by a single bond.
  Bonded to .R<GLY 2>.A<N 1> by a single bond.
> removeBond dipeptide.1.9 dipeptide.2.1
> desc dipeptide.1.9
ATOM
Name:    C
Type:    C
Charge:  0.616
Element: C
Atom flags: 20000|posfxd- posblt- posdrn- sel- pert-
notdisp- tchd- posknwn+ int- nmin- nbld-
Atom position: 5.485541, 2.705207, -0.000004
Atom velocity: 0.000000, 0.000000, 0.000000
  Bonded to .R<ALA 1>.A<CA 3> by a single bond.
  Bonded to .R<ALA 1>.A<O 10> by a single bond.
```

5.9.48. removeOffLibEntry

```
removeOffLibEntry library entry
```

```
STRING library
STRING entry
```

Remove the entry from the library.

```
> loadOff ETH.lib
Loading library: ETH.lib
```

```
Loading: ETH
> listOff ETH.lib
Index of library: ETH.lib
ETH
> removeOffLibEntry ETH.lib ETH
ETH was removed.
> listOff ETH.lib
Index of library: ETH.lib
```

5.9.49. removeRestraint

```
removeRestraint unit a b [ c [ d ] ]
```

```
UNIT   unit
ATOM   a
ATOM   b
ATOM   c
ATOM   d
```

Remove a restraint bond, angle, or torsion from unit, depending on the number of ATOMs specified. (see the *restrainBond* command for an example of this command).

5.9.50. restrainAngle

```
restrainAngle unit a b c force angle
```

```
UNIT   unit
ATOM   a
ATOM   b
ATOM   c
NUMBER force
NUMBER angle
```

Add a restraint angle to unit. A restraint is used to constrain atoms during molecular mechanics calculations. It causes the energy of the unit to increase as the deviation from the restraining internal coordinate increases. The restraint is between atoms a, b, and c. The force constant of the restraint is force (in kcal-mol$ˆ{-1}$-deg$ˆ{-1}$) and the equilibrium angle is angle (in degrees). (see the *restrainBond* command for an example of this command).

5.9.51. restrainBond

```
restrainBond unit a b force length
```

```
UNIT   unit
ATOM   a
ATOM   b
NUMBER force
NUMBER length
```

Add a restraint bond to unit. A restraint is used to constrain atoms during molecular mechanics calculations. It causes the energy of the unit to increase as the deviation from the restraining internal coordinate increases. The restraint is between atoms a and b and has a force constant of force (in kcal/mol-Å^2) and an equilibrium distance of length (in Å).

```
> 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>
> restrainBond GLY GLY.1.1  GLY.1.7 50 2.8
> restrainAngle GLY GLY.1.1 GLY.1.3 GLY.1.6 50 110
> restrainTorsion GLY GLY.1.2 GLY.1.1 GLY.1.3
> GLY.1.6 50 0 2
> desc GLY
UNIT name: GLY
Head atom: .R<GLY 1>.A<N 1>
Tail atom: .R<GLY 1>.A<C 6>
Restraint BOND: .R<GLY 1>.A<N 1> - .R<GLY 1>.A<O 7>
Kr=50.000000  R0=2.800000
Restraint ANGLE: .R<GLY 1>.A<N 1> - .R<GLY 1>.A<CA 3> -
Restraint TORSION: .R<GLY 1>.A<HN 2> - .R<GLY 1>.A<N 1> -
Kt=50.000000  T0=0.000000  N=2.000000
Contents:
R<GLY 1>
> removeRestraint GLY GLY.1.2 GLY.1.1 GLY.1.3 GLY.1.6
Removing restraint.
> removeRestraint GLY GLY.1.1 GLY.1.3 GLY.1.6
Removing restraint.
> removeRestraint GLY GLY.1.1  GLY.1.7
Removing restraint.
> 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>
```

In the above example, we illustrate several commands for adding and removing restraints to the "lib/all_amino94.lib" UNIT GLY. (Don’t try this at home or you might alter the "lib/all_amino94.lib". In general, we would not suggest modifying any standard libraries, rather, one should create a new UNIT for practice.)

5.9.52. restrainTorsion

```
restrainTorsion unit a b c d force phi multiplicity
```

```
UNIT   unit
ATOM   a
ATOM   b
ATOM   c
ATOM   d
NUMBER force
NUMBER phi
NUMBER multiplicity
```

Add a restraint torsion to unit. A restraint is used to constrain atoms during molecular mechanics calculations. It causes the energy of the unit to increase as the deviation from the restraining internal coordinate increases. The restraint is between atoms a, b, c, and d, and has a force constant of *force* (in kcal/mol-deg^2), an equilibrium torsion angle of ϕ (in degrees), and a periodicity of *multiplicity* (see the *restrainBond* command for an example of this command).

5.9.53. 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.
```

5.9.54. 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/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 gibbs and SPASMS 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.
```

5.9.55. 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.)

5.9.56. 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
```

5.9.57. select

```
select object
```

```
CONT   object
```

Sets the select flag on all ATOMs within object (see the *deSelect* command). In the following example, the side chain of the amino acid ALA has been selected. The center and charge of the selected atoms is compared with those of the entire ALA RESIDUE. We have also described one of the ATOMs in ALA, ALA.1.5, before and after selection in order that the user could see the output when a select flag is set.

```
> center ALA
The center is at: 4.04, 2.80, 0.49
> charge ALA
Total unperturbed charge: 0.00
Total perturbed charge:   0.00
> 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>
> desc ALA.1.5
ATOM
Name:    CB
```

```
Type:    CT
Charge:  -0.098
Element: C
Atom flags: 20000|posfxd- posblt- posdrn- sel- pert-
notdisp- tchd- posknwn+ int- nmin- nbld-
Atom position: 3.576965, 3.653838, 1.232143
Atom velocity: 0.000000, 0.000000, 0.000000
  Bonded to .R<ALA 1>.A<CA 3> by a single bond.
  Bonded to .R<ALA 1>.A<HB1 6> by a single bond.
  Bonded to .R<ALA 1>.A<HB2 7> by a single bond.
  Bonded to .R<ALA 1>.A<HB3 8> by a single bond.
> select ALA.1.5
> select ALA.1.6
> select ALA.1.7
> select ALA.1.8
> desc ALA.1.5
ATOM
Name:    CB
Type:    CT
Charge:  -0.098
Element: C
Atom flags: 20001|posfxd- posblt- posdrn- sel+ pert-
notdisp- tchd- posknwn+ int- nmin- nbld-
Atom position: 3.576965, 3.653838, 1.232143
Atom velocity: 0.000000, 0.000000, 0.000000
  Bonded to .R<ALA 1>.A<CA 3> by a single bond.
  Bonded to .R<ALA 1>.A<HB1 6> by a single bond.
  Bonded to .R<ALA 1>.A<HB2 7> by a single bond.
  Bonded to .R<ALA 1>.A<HB3 8> by a single bond.
> groupSelectedAtoms ALA sideChain
Added 4 atoms.
> center ALA@sideChain
The center is at: 3.51, 3.80, 1.45
> charge ALA@sideChain
Total unperturbed charge: 0.02
Total perturbed charge:   0.02
```

5.9.58. 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>
```

5.9.59. set

```
set container parameter object
```

```
CONT   container
STRING parameter
object object
```

This command sets various parameters associated with container. Please see the "Concepts" section for an extended discussion of the UNIT, RESIDUE, and ATOM parameters that may be altered for each type of object. (See the *add* command for an example of the *set* command.) The following parameters can be set within LEAP:

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.

connect3

This is an ATOM property which defines an ATOM that can be used in making links to other RESIDUEs.

connect4

This is an ATOM property which defines an ATOM that can be used in making links to other RESIDUEs.

connect5

This is an ATOM property which defines an ATOM that can be used in making links to other RESIDUEs.

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.

5.9.60. setBox

```
setBox unit

     UNIT   unit
```

The *setBox* command adds a periodic box to the UNIT, turning it into a periodic system for the simulation programs. The unit is first centered on the origin, using Van der Waals shells of the unit's atoms along each axis.

5.9.61. solvateBox

```
solvateBox solute solvent buffer [ closeness ]

     UNIT   solute
     UNIT   solvent
     object buffer
     NUMBER closeness
```

The *solvateBox* command creates a solvent box around the solute UNIT. The solute UNIT is modified by the addition of solvent RESIDUEs.

The normal choice for a TIP3 _solvent_ UNIT is WATBOX216. Note that constant pressure equilibration is 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.

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. 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 water.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.)

5.9.62. 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 UNIT, 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.
```

5.9.63. 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.

5.9.64. 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.
```

5.9.65. 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. The text within the source file is formatted exactly like the text the user types into LEAP. If a file named "paths.cmd" contains the following lines:

```
addPath /disk/howard/LeapTests
addPath /disk/howard/LeapTests/Cholesterol
addPath /disk/howard/LeapTests/Ethane
addPath /disk/howard/LeapTests/Solvents
addPath /disk/howard/LeapTests/Trp
```

"source"ing it will produce the output listing shown below.

```
> source paths.x
>> addPath /disk/howard/LeapTests
/disk/howard/LeapTests added to file search path.
>> addPath /disk/howard/LeapTests/Cholesterol
/disk/howard/LeapTests/Cholesterol added to file search path.
>> addPath /disk/howard/LeapTests/Ethane
/disk/howard/LeapTests/Ethane added to file search path.
>> addPath /disk/howard/LeapTests/Solvents
/disk/howard/LeapTests/Solvents added to file search path.
>> addPath /disk/howard/LeapTests/Trp
/disk/howard/LeapTests/Trp added to file search path.
```

5.9.66. 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  }
         }
```

5.9.67. 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 }
```

5.9.68. 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
```

5.9.69. 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.)

5.10. Examples

Note: see also the "PSC tutorial" materials under the "Web/index.html" page in the source tree.

In this section, we provide examples that illustrate how to use LEAP under a variety of conditions and for several different molecules. For each example, we first enumerate the principles that are demonstrated then print the log file. The log files have been edited in order to provide comments and explanations. The user may wish to recreate these examples and can do so by running the example log files found in the LEAP "demos" directory. (Refer to the "Log File" section of the "Concepts" section.)

5.10.1. A Simple Steroid: Cholesterol

In this example, the steroid cholesterol is created within xleap. The example illustrates:

- creating a LEAP UNIT using the Unit Editor
- using the `Build` and `Relax Selection` commands
- modeling a cyclic molecule
- creating LEAP OFF libraries.

This example is mainly a discussion of how to use the Unit Editor `Build` and `Relax Selection` commands to create a complex cyclic molecule. There are two points to mention that the user needs to know in order to create such a structure in the Unit Editor. First, since one builds molecules on a two-dimensional surface in xleap, structures that are minimized with the `Relax Selection` command may be planar unless they are first altered in the `Manipulate mode` or generated using the `Build` command. Second, the `Build` command frequently improperly generates fused cyclic structures. In the following example, the cholesterol molecule is drawn in the order: ring A, B, C, D, and alkyl chain. The LEAP structure was saved in an OFF library ("chol.lib") at each stage of refinement. The user is encouraged to look at the library entries if they have difficulty constructing the molecule. The user should note that this version of LEAP does a less than ideal job of generating a molecule such as cholesterol since vdw and electrostatic force field terms are not used in the Unit Editor minimizer.

```
> #
> #   First, execute the commands in the distribution
> #   "leaprc" file.  Because of space considerations,
> #   these commands have been deleted from this example.
> #   Next, create a new UNIT and edit it in the Unit
> #   Editor.  Choose the "Draw" button and the drawing
> #   element of carbon.  Draw a six-membered ring (ring
> #   A).  Select the menu item "Add H & Build".  Erase
> #   two hydrogens and draw the ring A hydroxyl oxygen
> #   and methyl carbon atoms.  Select the menu item
> #   "Add H & Build".  Save the "ch1" UNIT in an OFF
> #   library:
> #
> edit ch1
> saveOff ch1 chol.lib
> #
> #   Make a copy of the "ch1" UNIT so we can leave
> #   that UNIT as an illustration.  Edit the copy,
> #   "ch2", in the Unit Editor in order to add ring
> #   B.  First, remove the three hydrogens at the
> #   ring fusion carbons.  Choose the "Draw" button
> #   and the drawing element of carbon.  Draw a
> #   six-membered ring (ring B).  Select the ring B
> #   carbons not associated with the ring A junction.
> #   Select the menu items "Relax selected atoms" and
> #   "Mark built".  Save the UNIT in an OFF file:
> #
> ch2 = copy ch1
> edit ch2
> saveOff ch2 chol.lib
> #
> #   Make a copy of the "ch2" UNIT so we can leave that
> #   UNIT as an illustration.  Edit the copy,
> #   "ch3", in the Unit Editor in order to add ring
> #   C.  Choose the "Draw" button and the drawing
> #   element of carbon.  Draw a six-membered
> #   ring (ring C).  Select the ring C
> #   carbons not associated with the ring B junction.
> #   Select the menu items "Relax selected atoms" and
> #   "Mark built".  Save the UNIT in an OFF file:
> #
> ch3 = copy ch2
> edit ch3
> saveOff ch3 chol.lib
> #
> #   Make a copy of the "ch3" UNIT so we can leave that
> #   UNIT as an illustration.  Edit the copy,
> #   "ch4", in the Unit Editor in order to add ring
> #   D.  Choose the "Draw" button and the drawing
> #   element of carbon.  Draw a five-membered
> #   ring (ring D).  Select the ring D
> #   carbons not associated with the ring C junction.
> #   Select the menu items "Relax selected atoms" and
> #   "Mark built".  Save the UNIT in an OFF file:
> #
> ch4 = copy ch3
> edit ch4
> saveOff ch4 chol.lib
> #
> #   Add the alkyl group to ring D.  Select the
> #   menu items "Relax selected atoms" and
> #   "Mark built".  Save the UNIT in an OFF file:
> #
> ch5 = copy ch4
> edit ch5
> saveOff ch5 chol.lib
> #
> #   Add all remaining hydrogens to the molecule.
> #   Select the menu items "Relax selected atoms"
> #   and "Mark built".  Save the UNIT in an OFF file:
> #
> ch6 = copy ch5
> edit ch6
> saveOff ch6 chol.lib
> #
> #   Quit the xLEaP program:
> #
> quit
```

5.10.2. An Ion-Molecule Complex: 18-Crown-6 and Potassium Cation

In this example, the crown ether/ion complex 18-crown-6/K^+ is created within xleap and then solvated with methanol. The example illustrates:

- creating a LEAP UNIT starting from AMBER PREP input files
- modeling a cyclic molecule
- loading a PDB file into a LEAP UNIT
- building a box of solvent molecules
- solvating a molecule
- converting an AMBER parameter file to a LEAP PARMSET
- creating LEAP OFF libraries
- creating topology and coordinate files for AMBER or LEAP.

```
> #
> #   First, execute the commands in the distribution
> #   "leaprc" file.  Because of space considerations,
> #   these commands have been deleted from this example.
> #   Next, 3 UNITs will be created from AMBER PREP
> #   files:
> #
> loadAmberPrep cra.in
> loadAmberPrep crb.in
> loadAmberPrep crc.in
> #
> #   A PDB file is loaded into the UNIT "cr":
> #
> cr = loadPdb crown.pdb
> #
> #   The "cr" UNIT is edited in the xLEaP X-Windows
> #   Unit Editor and a bond is added between RESIDUEs
> #   CRA(1) and CRB(6).  This bond could also have been
> #   created with the command line editor using the command:
> #   "crossLink cr.1 lastend cr.6 firstend".
> #   The same editor is then used to create
> #   a potassium ion and assign atom properties to
> #   that ion.  The ion is placed at the center of the
> #   crown ether.  Finally, a methanol UNIT ("meoh") is created
> #   in the Unit Editor.  Atom Properties are defined
> #   for the methanol UNIT and the ATOMs in this UNIT
> #   are minimized using the Unit Editor "Relax Selection"
> #   menu item:
> #
> edit ion
> edit cr
> edit meoh
> #
> #   We will then create methanol solvent box.
> #   In order to do so, it is first
> #   necessary to align the principle axes of the
> #   "meoh" UNIT along the Cartesian coordinate axes
> #   with its geometric center at ( 0, 0, 0 ).
> #   The bounding box is defined to be a 4 angstrom
> #   cube:
> #
> alignAxes meoh
> set meoh box 4
> set meoh restype solvent
> #
> #   Parameters will be needed for molecular mechanics.
> #   The parameters had previously been used in the
> #   AMBER modules and therefore an AMBER parameter
> #   file is used to load the 18-crown-6 parameters.
> #   We now will solvate the crown with methanol
> #   molecules.  The original crown/ion UNIT will be
> #   copied so the "cr" UNIT will not be altered by solvent:
> #
> cr_parms = loadAmberParams cr_parms.mod
> cr-tem = copy cr
> solvateBox cr-tem meoh 10
> #
> #   The parameters, and all of the UNITs that have
> #   been created within xLEaP, are then saved as
> #   OFF library files.  Finally, topology and coordinate
> #   files are created for use in AMBER or SPASMS:
> #
> saveOff cr_parms crown.lib
> saveOff meoh crown.lib
> saveOff cr crown.lib
> saveOff ion crown.lib
> saveOff cr-tem crown.lib
> saveAmberParm cr-tem cr-meoh.top cr-meoh.crd
> #
> #   Quit the xLEaP program:
> #
> quit
```

5.10.3. Free Energy Perturbation: Guanine To Adenine

One type of calculation frequently performed in molecular mechanics is that of Free Energy Perturbation. During Free Energy Perturbation, the potential energy function of one molecular species is slowly perturbed to that of a second molecular species through a perturbation parameter called $lambda$. This calculation allows one to determine the relative Gibbs or Helmholtz free energy differences between the two species. In this example, LEAP is used to produce AMBER/SPASMS input files for the perturbation of Guanine to Adenine, *in vacuo* and in water, using the 1991 force field residues. This perturbation involves the a) replacement of GUA ((C_{(6)} = O_{(6)})) into ADE ((C_{(6)} - NH_{2})), b) replacement of GUA ((N_{(1)} - H)) into ADE ((N_{1})), and c) replacement of GUA ((C_{(2)} - N_{(2)}H_{(2)})) into ADE ((C_{2} - H)).

This example illustrates:

- creating Free Energy Perturbation input files within LEAP
- loading a AMBER PREP RESIDUE into LEAP
- converting an AMBER parameter file to a LEAP PARMSET
- creating LEAP OFF libraries
- creating topology and coordinate files for AMBER or LEAP.

```
> #
> #   First, execute the commands in the distribution
> #   "leaprc" file.  Because of space considerations,
> #   these commands have been deleted from this example.
> #   An AMBER PREP residue for Guanine (GUB) is loaded
> #   into LEaP:
> #
> loadAmberPrep guan.in
> #
> #   An AMBER format parameter set file is loaded and placed
> #   in "parmatgc.dat" file.  The resulting PARMSET is saved
> #   as an OFF file:
> #
> newparm = loadAmberParams parmatgc.dat
> saveOff newparm newparm.lib
> #
> #   The GUB UNIT is edited in the Unit Editor to
> #   prepare the perturbation properties.  The user
> #   should examine the GUB UNIT in "gub.lib",
> #   using the Unit Editor and Atom Properties
> #   Editor, to see the changes that are made for
> #   perturbation.  The UNIT is saved as an OFF file
> #   and coordinate and topology files are prepared
> #   for AMBER and SPASMS :
> #
> edit GUB
> saveOff GUB gub_leap.lib
> saveAmberParm GUB gub_vacuo.top gub_vacuo.crd
> #
> #   The GUB UNIT is then solvated with a box of
> #   TIP3P waters.
> #   The GUB UNIT is saved in an OFF library file.
> #   Topology and coordinate files for AMBER or
> #   SPASMS perturbation calculations (TIP3P
> #   solvated) are created:
> #
> loadOff water.lib
> solvateBox GUB WATERBOX216 10
> saveAmberParm GUB gub_water.top gub_water.crd
> #
> #   Quit the program
> #
> quit
```

5.10.4. Creating Polynucleotides: B DNA

A double helix of dna is built using LEAP in this example. Two examples of adding counterions to the dna are illustrated. In one example, both counterions and tip3p water are added. The second example is an in vacuo model and the counterions have a large vdw radius in order to simulate a water shell surrounding them.

The Cartesian coordinates for the dna were obtained from the PDB file "1bd1". This file also contained coordinates for X-ray waters of crystallization and two triethylammonium ions. Neither of these species are included in this representation. Only one nucleotide strand is found in the PDB file since the symmetry operations necessary to generate the second strand are given. These operations are used with the LEAP `transform` command in order to create the double helix molecule.

This example illustrates:

- creating a dna molecule within LEAP
- loading a PDB file into a LEAP UNI
- using PDB symmetry operations to build a molecule
- using the `addIons` command to place counterions around a molecule
- solvate a molecule using the LEAP `solvateBox` command
- creating LEAP OFF libraries
- creating topology and coordinate files for AMBER or LEAP.

```
#
dnaStrandA = loadPdb 1bd1.pdb
#
#   A second strand is generated by making a
#   copy of the first strand and then applying
#   the symmetry operations found in the PDB
#   file to the latter strand.  The DNA double helix
#   is generated by combining the two strands.
#   The combine command does not link the UNITs
#   in the LIST argument with bonds:
#
dnaStrandB = copy dnaStrandA
transform dnaStrandB {
{ -1  0  0  }
{  0  1  0  }
{  0  0 -1  }
 }
dnaDoubleHelix = combine { dnaStrandA dnaStrandB }
#
#   Two complexes are now created.  The first is
#   the "in vacuo" association of a DNA double
#   strand with ions.  Since this complex will not be
#   solvated, the ions have large VDW radii in order to
#   simulate a shell of water surrounding them.  The
#   second complex associates DNA, ions, and a box of
#   TIP3P waters.  The ions (Na+) in the later case
#   have VDW radii of "standard" size.   In both
#   complexes, enough counterions are added to
#   neutralize the DNA:
#
loadOff ions.lib
dnaDHBigIons = copy dnaDoubleHelix
addIons dnaDHBigIons IB 0
#
dnaDHIons = copy dnaDoubleHelix
solvateBox dnaDHIons WATBOX216 9
addIons dnaDHIons Na+ 0
#
```

```
#   The DNA double helix UNITs are saved in
#   an OFF library file.  Topology and coordinate
#   files for the double helix molecules with associated
#   counterions/water are created for use in AMBER:
#
saveOff dnaDoubleHelix dna.lib
saveOff dnaDHBigIons dna.lib
saveOff dnaDHIons dna.lib
saveAmberParm dnaDHBigIons dnaDHBigIons.top dnaDHBigIons.crd
saveAmberParm dnaDHIons dnaDHIons.top dnaDHIons.crd
#
#   Quit the program
#
quit
```

5.10.5. A Protein/Ligand Complex: trp Repressor

This example illustrates one method of building the trp repressor protein and solvating it within LEAP. The trp repressor protein is dimeric. Each protein monomer contains one zwitterionic trp amino acid ligand and one polypeptide unit. The two monomeric units are related by symmetry.

In the example, a PDB structure (1wrp) is used to generate the trp repressor Cartesian coordinates. The PDB file contains coordinates for only one monomer; the second monomer must be generated from the symmetry operations given in the PDB file. The original PDB file also contained Cartesian coordinates for water molecules found in the crystal structure. We have removed the waters from the PDB file as we did not want to use them in the example. In addition, we have split the original PDB file into two files: the file "wrp.pdb" contains one polypeptide chain and the file "trp.pdb" contains the trp ligand coordinates.

The example illustrates:

- creating a protein within LEAP
- loading a PDB file into a LEAP UNIT
- using PDB symmetry operations to build a molecule
- the use of the `copy` command to simplify the creation of a molecule
- using each of the solvation options of LEAP
- creating LEAP OFF libraries
- creating topology and coordinate files for AMBER.

```
#
#   First, execute the commands in the distribution
#   "leaprc" file.  Because of space considerations,
#   these commands have been deleted from this example.
#   Load the Cartesian coordinates of one protein
#   monomer into a LEaP UNIT:
```

```
#
wrpA = loadPdb wrp.pdb
#
#   The trp repressor ligand is a zwitterionic amino acid.
#   Since the LEaP libraries do not include
#   zwitterionic amino acids, the ligand will be
#   modeled by making a copy of "NTRP" (N-terminal TRP)
#   and modifying the C-terminal to be an anion.  The
#   Unit Editor will be used to make the modification
#   and to add the Atom Properties of the modified ATOMs.
#   First, a copy will be made of "NTRP".  The UNIT and
#   residue names of the copy will then be changed from
#   "NTRP" to "WRP".  Finally, the "WRP" UNIT will be
#   modified within the Unit Editor.  The following changes
#   are made - 1) a second oxygen is drawn and connected to
#   the C-terminal carbon; 2) the C-terminal carboxylate
#   ion is selected and the "Build" command is chosen
#   from the "Build menu"; 3) the "Edit selected atoms"
#   command is chosen from the "Selection menu".  An Atom
#   Properties table will appear for the carboxylate ion group.
#   In the #   table, the "Name" of the new oxygen is changed to
#   "OXT" and its "Type" is changed to "O".  The "Charge"s
#   of both carboxylate ion oxygens are changed to -0.7525.
#   The Atom Properties table and Unit Editor are then exited
#   and the "trpA" Cartesian coordinates are loaded from a PDB
#   file:
#
WRP = copy NTRP
set WRP name "WRP"
set WRP.1 name "WRP"
edit WRP
trpA = loadPdb trp.pdb
#
#   Create one monomer by associating the protein
#   chain with the zwitterion.  Then, to create the
#   second monomer, copy the first monomer and apply
#   the symmetry operations found in the PDB file.
#   Finally, combine the two monomers to form the
#   protein:
#
monomerA = combine { wrpA trpA }
monomerB = copy monomerA
translate monomerB { 0  0 -24.53333 }
transform monomerB {
{ 1  0  0 }
{ 0 -1  0 }
{ 0  0 -1 }
 }
trpRepressor = combine { monomerA monomerB }
```

```
#
#   Solvate the protein using the various LEaP
#   solvation options.  Prior to each solvation,
#   a copy of the protein UNIT is made so that
#   the original UNIT is not altered by the water
#   molecules.  The solvent used in each of the
#   examples is TIP3P water taken from the
#   water OFF distribution library.  In the
#   case of the "solvateCap" command, the cap is
#   centered at the Calpha carbon of the "trpA" ligand
#   in "monomerA".  Two of the solvation commands are
#   shown below but not executed in the example:
#
# trpRepressorDontClip = copy trpRepressor
# solvateDontClip trpRepressorDontClip WATBOX216 8.0
# trpRepressorBox = copy trpRepressor
# solvateBox trpRepressorBox WATBOX216 8.0
#
trpRepressorCap = copy trpRepressor
solvateCap trpRepressorCap WATBOX216
trpRepressorCap.106.CA 8.0
trpRepressorShell = copy trpRepressor
solvateShell trpRepressorShell WATBOX216 8.0
#
#   The monomer, ligand, and trp repressor UNITs are
#   saved in an OFF library file.  Topology and coordinate
#   files are created for use in AMBER or SPASMS:
#
saveOff wrpA trpRepressor.lib
saveOff trpA trpRepressor.lib
saveOff trpRepressor trpRepressor.lib
saveAmberParm trpRepressor trpRepressor.top trpRepressor.crd
#
#   Quit the program
#
quit
```

6. Sander

```
Usage: sander [-O] -i mdin -o mdout -p prmtop -c inpcrd -r restrt
              -ref refc -x mdcrd -v mdvel -e mden -inf mdinfo
```

`-O` Overwrite output files if they exist.

6.1. 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.

Sander provides standard protocols for minimization and molecular dynamics, and we use it for just about everything except free energy calculations. Some of the features are outlined in the following paragraphs:

(1) *Sander* provides direct support for the AMBER and AMBER/OPLS force fields for proteins and nucleic acids, and for the TIP3 and TIP4 models for water. Other types of restraints can be applied, and the code allows some variation in functional form as well as in parameters. These variations include alternate functions for "improper" torsions and Urey-Bradley interactions, so that force fields like that of version 22 of CHARMM can be supported. In addition, "non-additive" force fields based on atom-centered dipole polarizabilities can be invoked.

(2) The relative weights of various terms in the force field can be varied over time. It is also straightforward to choose a constant weighting that implements a "geometric" force field, in which bonds and angles are kept fixed, torsions are free and non-bonded interactions consist solely of chargeless Van der Waals interactions to prevent steric overlap. This sort of potential can be useful when major conformational changes are anticipated, or when one is concerned that errors in the more realistic atomic potentials are biasing the results.

(3) Two periodic imaging geometries are included: rectangular parallelopiped and truncated octahedron (box with corners chopped off). 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) The user 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.

(5) 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.

(6) Restraints can be directly defined in terms of NOESY intensities (calculated with a relaxation matrix technique), 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.

We have divided this manual into the six sections listed below.

Purpose	*Sections involved*
Simple min/md	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

If you are just doing "standard" minimization or dynamics, 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* and *six* allow you to add sophisticated penalty functions during NMR refinement.

6.2. History and 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 and Liem Dang. The pseudocontact shift code was provided by Ivano Bertini of the University of Florence.

Parallelism. In version 4.0, an SGI shared memory version by Roberto Gomperts and Michael Schlenkrich of SGI with assistance from Thomas Cheatham was distributed on request, as were a message-passing version for the SP1 by Steven Chin of IBM and a KSR version by David Zirl and Nick Camp. A general PVM version by Terry Lybrand and Eric Swanson of the University of Washington was included as a separate source tree in the later 4.0 release. We also had an unreleased version of

Steve Debolt's AMBERCUBE [1] (based on release 3A) that ran on nCube and had been ported to 4.0 on the Intel Paragon by David Case of Scripps with help from Jerry Greenberg and Jack Rodgers of San Diego Supercomputer Center. (George Seibel of UCSF and Tom Darden of NIEHS also had done shared memory versions of previous releases for Cray and SGI, respectively.)

In 4.1 and later versions, all message-passing parallel code falls under a generalized MPI interface developed by Jim Vincent and Ken Merz of Pennsylvania State University, who provided PVM, SPx and T3D versions. [2] (The PVM, *etc.* message-passing libraries are only used for systems that do not have MPI implemented.) Thomas Cheatham and David Case helped to integrate, extend, and optimize this work. The SGI shared memory version from 4.0 was improved by Gomperts and Schlenkrich of SGI, and has been reorganized and incorporated into the release by Thomas Cheatham. Other credits: Steve Chin (IBM, SPx optimization), Jeyapandian Kottalam, Mike Page and Asiri Nanayakkara (Cray optimization), Michael Crowley (Pittburgh Supercomputer Center, T3D portable namelist port, PME development), and Thomas Huber (Ludwig Maximilian Universitaet, TCGMSG library).

Particle Mesh Ewald. The Particle Mesh Ewald (PME) method, implemented originally in AMBER 3a and contributed by Tom Darden [3] of the NIEHS, is included. 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. Parallelization of PME was done by Mike Crowley, with assistance from Tom Cheatham and David Case. Crowley.

---

[1] DeBolt, S.E. and Kollman, P.A. (1993) *J. Comput. Chem.* **14**, 312.

[2] "A Highly Portable Parallel Implementation of AMBER 4 Using the Message Passing Interface Standard," Vincent, J. and Merz, K.M. (1995) *J.Comput.Chem.* **11**, 1420-1427.

[3] Ported to AMBER 4.1 by Tom Darden with the assistance of Thomas Cheatham. For citation information and more information about the method, see the input description in section one of this manual.

6.3. File usage.

Usage: `sander [-O] -i mdin -o mdout -p prmtop -c inpcrd -r restrt -ref refc -x mdcrd -v mdvel -e mden -inf mdinfo`

`-O` Overwrite output files if they exist.

On VMS systems, files are assigned by Fortran unit number. These unit numbers are also sometimes useful to reference when viewing i/o-related operating system error messages, and are given below along with a description of each file.

file	unit	in/out	purpose
mdin	5	input	control data for the min/md run
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 constraint
mdout	6	output	user readable state info and diagnostics
mdinfo	7	output	latest mdout-format energy info
restrt	16	output	final coordinates, velocity, and box dimensions if any - for restarting run
mdcrd	12	output	coordinate sets saved over trajectory
mdvel	13	output	velocity sets saved over trajectory
mden	15	output	extensive energy data over trajectory

6.4. Overview of the contents of mdin.

Section	*Comments*	*Format*
ONE	Standard minimization and dynamics input	`&cntrl` namelist
TWO	Varying conditions	Parameters for changing temperature, restraint weights, etc. during the MD run. Sometimes called "weight change lines". Each weight change line is specified by a separate `&wt` namelist specifier, ending with `&wt type='END', &end.` *NOTE:* the termination of this section is `&rst iat=0, &end` at the end of section FOUR. *I.e.* without an explicit section THREE or section FOUR, you must use this line if you want a group specification following this one to be read properly.
THREE	I/O redirection	TYPE=*filename* lines. Optional. Section ends with the first non-blank line which does not correspond to a recognized redirection.
FOUR	Distance and angle restraints	Multiple `&rst` namelists; if a *DISANG=filename* redirection was given in THREE, these are read from *filename* instead of *mdin*. One `&rst` definition is given per restraint. Section FOUR is terminated by `&rst iat=0, &end`.
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. Ended by `&noeexp npeak(1)=-1, &end`.
SIX	Chemical shifts restraints	Read only if NMROPT= 2 and a *SHIFTS=filename* or *PCSHIFT=filename* was given in THREE, Exactly one `&shf` or `&pcshf` namelist (or both) must be provided for this section.

6.5. 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 explicitly listed retain their default values. Support for namelist input is included in almost all current Fortran compilers, and is a standard feature of Fortran 90. In addition, a "portable" namelist implementation, written in Fortran by N.H.F. Beebe, is included, to allow namelist input on (almost) all machines. This "portable" version is actually an improvement over most native implementations, because it gives better error messages in case of problems. A detailed description of the namelist convention is given in Appendix B.

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 files in the demo directory contain examples of this format. 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.

A simple input file

```
Sample input file : just a few steps of minimization.
    [minimize for 50 cycles, print results every 10 steps]
  &cntrl
     imin=1, maxcyc=50, ntpr=10, scee=2.0,
  &end
```

6.5.1. General flags describing the calculation.

TIMLIM Time limit, in seconds, for the job. Default 999999.

IMIN Flag to run minimization

= 0	No minimization (only do molecular dynamics; default)
= 1	Perform minimization (and no molecular dynamics)

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

6.5.2. Nature and format of the input.

NTX	Option to read the initial coordinates, velocities and box size from the "inpcrd" file (also see INIT). 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. Note: BOX (the periodic box lengths) is written to the restrt file in periodic boundary runs. If NTB.gt.0 (a periodic boundary run) and NTX.lt.6, the box sizes in the prmtop are used; otherwise, the box sizes from the inpcrd (MD restrt) file will be used. This enables one to use the last box from the constant pressure regime when switching to constant volume runs.

= 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.
= 6	X, V and BOX(1..3) are read unformatted.
= 7	X, V and BOX(1..3) are read formatted.

IREST	Flag to restart the run.

= 0	No effect (default)
= 1	restart calculation (i.e. read restart time and set INIT= 4. 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

6.5.3. 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.
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. 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. NTWR<0 is not allowed in NRUN>1, since the "nstep" counter resets for each "run" and so files would be overwritten. Default 50.
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.
NTWXM	The maximum number of steps that NTWX is active. At this number of steps no more trajectories will be written to file "mdcrd". Set this to 0 to disable the limit.
NTWVM	Analogous to NTWXM for velocities. 0 to disable.
NTWEM	Analogous to NTWXM for energies. 0 to disable.
IOUTFM	Format of velocity, coordinate, and energy sets

	= 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).
	Coord/velocity archives will include:

	= 0	all atoms of the system (default).

< 0	only the solute atoms.
> 0	only atoms 1->NTWPRT.

6.5.4. 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 (default)
= 1	constant volume
= 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 EDIT can result in a net void when solvent molecules are subtracted which can aggregate into ‘vacuum bubbles’ in a constant volume run. Another consideration is box shrinkage under constant pressure, which if the solute clearance has been chosen too close to the cutoff distance can result in solvent molecules ‘seeing’ parts of the solute in opposite directions (not desirable if one believes that less correlated interactions are significantly more like simulations in free solution). The remedy for this is to allow enough margin when building the box.

IDIEL Type of dielectric function to be used in calculating the electrostatic energy.

= 0	distance dependent dielectric function. This is used to mimic the presence of a high dielectric solvent, typically for simulating water when no explicit water is present.
= 1	constant dielectric function. This is used when there is explicit solvent (e.g. water) in the calculation, or for a true gas phase calculation. Default.

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 $4r_{ij}$ set DIELC=4 and IDIEL=0. Default 1.0.

CUT The primary cutoff distance for non-bonded interactions. CUT should be no more than half the shortest BOX dimension in order to maintain spherical symmetry in the nonbonded potential. For non-PME runs, AMBER only uses a residue-based cutoff. This means that if any atom of one residue is within CUT of any atom of another residue, every atom in each residue will see every atom of the other residue. This is done to avoid splitting the residue dipoles. The average effective cutoff is thus increased by the average residue diameter. For PME simulations, CUT is used to separate short-ranged and long-ranged interactions, and an atom-based cutoff scheme is used. Default 8.0.

NTNB Non-bonded pair list.

= 0	no pair list will be generated and no nonbonded interactions are calculated.
= 1	Normal behavior (default; recommended).

NSNB After NSNB steps the non-bonded pair list will be updated. It is recommended that the pairlist be updated every 25fs, but for very mobile systems or when short cutoffs are used it may be necessary to update the pairlist more frequently. If the nonbonded cutoff is larger than the system size (ie, no cutoff), you should set NSNB to a large value so that the pairlist is only constructed once. Default 25.

NTID Water pairlist method.

= 0	In periodic systems the water pairlist is generated from oxygen coordinates. Default and recommended.
= 86	pairlist generated on residue basis from atom coordinates. I.e. if any pair of atoms in different waters is within the cutoff, all the interactions between the 2 waters are used. This yields an effective cutoff distance that is somewhat longer than that specified in CUT. This option may be extremely slow and is provided only for comparison to old runs.

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. *No default; must be set.*
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 0.0.
ICHDNA	Option to modify the charge of end hydrogens. This is useful for "in vacuo" simulations of RNA and DNA. Without this option, energy minimization calculations on nucleotides will result in bonding between the 5' and 3' hydrogens and the corresponding phosphate groups. This option transfers the charge from H5' to O5' so that the hydrogen on the end is neutral.

= 0 no charge modification (default)

= 1 modify charge

6.5.5. The soft repulsion option.

ISFTRP

= 0 No "soft repulsions" (default).

If ISFTRP > 0, a "soft" repulsion-only potential term will be used in place of the standard 6-12 potential. This term has the form

$$E = K_{rep} (r_o^2 - r^2)^2 \quad \text{for} \quad r < r_o$$
$$E = 0 \quad \text{for } r > r_o$$

where r_o is the sum of the van der Waals radii of the interacting atoms, r is their interatomic distance, and K_{rep} is a force constant. This type of potential has shown some usefulness in improving the efficiency of restrained refinement using MD.

= 1 The standard 10-12 potential will still be used for interactions between hydrogen bonding atoms, rather than the "soft" repulsion-only term.

>=2 The soft-repulsion term will replace the 10-12 term for hydrogen bonds, as well.

RWELL Default 0.0. If ISFTRP > 0, RWELL gives the initial value of K_{rep}. All interactions use the same value of K_{rep}, which can be changed using the SOFTR option in the NMR control file (see below).

[Note: If, in the force field, either epsilon or r_o for an atom is specified to be zero, that atom will not contribute to the vdw potential energy. This is always true, regardless of the values of ISFTRP or RWELL.]

Caution: Note that the van der Waals radii in the "standard" force field may not be what you want for soft repulsion. In particular, the atom type *HC* (hydrogen bonded to carbon) has a large value for r* (1.54 A) and a very small value for the well depth (0.01 kcal/mol). This results in a relatively weak repulsive wall, but will not translate well into a soft repulsion. You will probably want to use a *frcmod* file to reduce this radius to something more like 1.0 A. You may wish to modify other radii as well. Some useful information about this (for proteins) is in "Calibration of effective van der Waals atomic contact radii for proteins and peptides", by Iijima, Dunbar and Marshall, *Proteins: Str. Funct. Gen.* **2,** 330-339 (1987).

Note also that the *RSTAR* weight function (described below) can be used to modify *all* of the radii by a constant amount. For example, if you want the repulsive force to begin where the 6-12 potential crosses zero (rather than at the minimum), set *RSTAR* to (1/2)**(1/6), or about 0.8909.

6.5.6. Polarizable potentials.

IPOL Inclusion of polarizabilities in the force field.

= 0 non polar calc (default).

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

= 2 Polarization calculation + read and use 3-body interaction definitions

Note: polarization is expensive and is currently recommended ONLY for investigation of polarization parameters.

N3B Number of three-body interactions to be defined; current maximum is 5.

NION Number of ions in the system.

AT1(I) The second atom in this 3-body interaction.

AT2(I) The third atom in this 3-body interaction.

ACON(I) The pre-exponential factor for this 3-body interaction.

BETA3(I) The beta value for this 3-body interaction.

GAMMA3(I) The gamma value for this 3-body interaction.

6.5.7. 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.
NTR	Flag for restraining specified atoms in Cartesian space using a harmonic potential. Note: the restrained atoms are read in GROUP format after the numeric input from file "mdin" - see Appendices for GROUP. 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

6.5.8. 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.

6.5.9. Molecular dynamics.

NRUN	Number of MD-runs of NSTLIM steps to be performed. Since the restart coordinates are written only at the end of each "run", it is sometimes advisable to break a long MD calculations into several "runs". The number of picoseconds of molecular dynamics is equal to the product of NRUN x NSTLIM x DT. Default 1.
NSTLIM	Number of MD-steps per NRUN to be performed. Default 1.
NDFMIN	Number of degrees of freedom that will be subtracted from the total number of degrees of freedom. If either NTCM or NSCM .NE. 0 then this option should be set equal to 6. Otherwise, NDFMIN should be 0. NDFMIN, NTCM, and NSCM are ignored for belly dynamics. Default 0.
NTCM	Flag for the removal of translational and rotational motion at the beginning of the simulation.

= 0	The translational and rotational motion about the center of mass is not removed (default)
= 1	The above motion is removed one time at the beginning of the simulation.

NSCM	Flag for the removal of translational and rotational motion at regular intervals. After every NSCM steps, translational and rotational motion will be removed. This flag is ignored for both belly and periodic simulations. Default 0.
INIT	Flag for different starting procedures. If option NTX is less than 4, INIT should be equal to 3. If option NTX is greater than or equal to 4, this option should be equal to 4.

= 3	Generate starting velocities (NTX = 1 or 2). V(T-DT/2) is obtained by calculating force(T) unless TEMPI .gt. 1e-6, in which case the velocities are assigned from a Maxwellian at TEMPI K. Default.
= 4	Use input velocities (NTX >= 4). V(T-DT/2) is read from the input file "inpcrd".

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.

6.5.10. Temperature regulation.

TEMP0	Reference temperature at which the system is to be kept. 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.
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 some of the following options are rather ad-hoc, and may not result in a thermodynamically relevant ensemble. However, they may be useful when using MD strictly to sample conformational space, such as with simulated annealing and nmr refinement − cases where the temperature of the system may be too unstable to use standard coupling scheme. In particular, option NTT=4 may be useful in such cases. Coupling schemes NTT=0,1 or 5 should be used when generating a thermodynamic ensemble is crucial. = 0 Constant total energy classical dynamics. Velocities are never rescaled after the start of the simulation *except* at the end of every NSTLIM steps, when they will be rescaled to the target temperature if the current temperature deviates from TEMP0 by more than DTEMP. Default, but owing to the "hard" cutoff that lacks a switching function, energy will not be conserved unless the cutoff includes the whole system. = 1 Constant temperature, using the Berendsen coupling algorithm (Berendsen et al. J. Chem. Phys., 81, 3684 (1984)). A single scaling factor is used for all atoms. This is good for small solutes, e.g. methane, but can result in cold solute for larger ones. See NTT=5.

	= 2	Constant temperature, using Berendsen coupling algorithm. But only consider the solute temperature in determining the velocity scaling factor on each step. Could result in solvent atoms having very high temperature, and is not recommended for most cases.
	= 3	Constant temperature, using Berendsen algorithm. But only rescale when the temperature deviates from TEMP0 by more than DTEMP. Single scaling factor.
	= 4	Any time 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	Berendsen algorithm, use separate scaling factors for atoms of the solute and atoms of the solvent. This option is recommended as a replacement for NTT=1, and can help alleviate the "cold solute/hot solvent" problem.
	<−1	Re-assign random velocities for all atoms whenever the current temperature deviates by more than DTEMP from TEMP0 (target temperature), and every ABS(NTT) steps. Velocities are assigned in a Maxwellian distribution.
	=−1	Re-assign random velocities for all atoms whenever the current temperature deviates by more than DTEMP from TEMP0. Velocities are assigned in a Maxwellian distribution.

NOTE 1: When option (5) is chosen, both the solute and solvent coupling constants are used (TAUTP and TAUTS, respectively). In cases (1), (2), and (3), the single temperature coupling constant TAUTP is used for all atoms.

NOTE 2: If you are using NTT=2 or NTT=5, you can specify the variable ISOLVP to redefine the last_solute_atom pointer. See below.

ISOLVP Last-solute-atom pointer to be used with temperature scaling, when separate scaling of solute and solvent atoms has been requested (NTT=2 or NTT=5).

By default (ISOLVP=0), the last non-TIP3P water molecule in the system is generally taken as the last_solute_atom. For example, the counterions are considered part of the "solute" by default. You could re-define the counterions to be part of the solvent by setting ISOLVP.

DTEMP For NTT = 0, if the difference between the system temperature and TEMP0 is more than DTEMP at the end of each run, the velocities will be linearly scaled to TEMP0. Default 0.0.

TAUTP Time constant for heat bath coupling for the SOLUTE. Default 0.2. 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.

TAUTS	Time constant for the heat bath coupling for the SOLVENT (NTT=5). Default 0.2.
VLIMIT	If .ne. 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 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. A warning message will be printed whenever the velocities are modified. Runs that have more than a few such warnings should be carefully examined. Default 0.0.

6.5.11. PEACS temperature algorithm

PEACS is a searching technique wherein the trajectory follows a constant energy contour. See Schaik et al, J. Comp. Aided Mol. Design, 6, 97 (1992). PEACS can only be carried out using MD (not minimization).

TAUV0 Defines the rate constant for lowering the temperature if a PEACS constant potential energy search is to be carried out.

= 0.0	No PEACS search will be carried out (default).
> 0.0	TAUV0 defines the rate at which the target energy contour is annealed down.

TAUV	When a PEACS search is being carried out (TAUV0 > 0.0), TAUV defines the coupling constant between the target energy contour and the contour being followed. Default 0.1.
VZERO	Defines the value of the initial potential energy contour to be followed if a PEACS search is being carried out. If VZERO is specified as 0.0 (default), the initial energy of the system will be used.

6.5.12. Pressure regulation

Pressure regulation only applies when Constant Pressure periodic boundary conditions are used (NTB = 2).

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

	= 0	Used with NTB not = 2 (default)
	= 1	md with isotropic position scaling
	= 2	md with anisotropic diagonal (x-,y-,z-) position scaling
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 0.2.
NPSCAL		Method for pressure scaling of the atomic coordinates. Pressure scaling means changing the size of the box to match the target pressure, which involves scaling the positions of the contents of the box so that they are proportionally distributed within the new box size. For example if the box contracts, the system as a whole must be contracted to avoid overlaps at the periodic boundary, and if the box expands, the system is expanded uniformly to fill the vacuum at the edge(s).
	= 0	Atom scaling. All atoms are independently moved according to the scale factor. This causes some degree of compression or stretching of bonds. Default.
	= 1	Molecule scaling. Bonded groups of atoms (molecules) are moved as units. This is intended to avoid changing bond lengths as a side effect of pressure scaling, but may disrupt coordination of extended contacting molecules such as nucleic acid strands.

6.5.13. SHAKE bond length constraints.

NTC	Flag for SHAKE to perform bond length constraints. (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. SHAKE is used in the TIP water potentials and keeps waters rigid so that the hydrogens (which have 0 vdw radius) do not extend beyond the vdw sphere defined for the oxygens. For energy minimization, no SHAKE should be necessary, unless electrostatic energies blow up (negatively) in a water bath. If both problems affect a minimization with periodic boundary conditions, increasing the box dimension at the end of the prmtop file by 0.2-0.8 Angstroms may help.

= 1 SHAKE is not performed (default)

= 2 bonds involving hydrogen are constrained

= 3 all bonds are constrained

TOL Relative geometrical tolerance for coordinate resetting in shake. Recommended maximum: <0.0005 Angstrom Default 0.0005.

6.5.14. Special water treatment.

IMGSLT Controls Solute-Solvent imaging in periodic boundary calculations.

= 0 Solute is imaged with solvent. Solute is allowed to interact with solvent images (if they are within CUT). 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.

IFTRES Flag to remove the nonbonded cutoff from the solute in periodic boundary simulations.

= 0 ALL intramolecular solute - solute nonbonded interactions are calculated regardless of whether the interatomic distance is greater than the nonbonded cutoff. Solute-solute imaging is turned off.

= 1 Nonbondeds are evaluated normally. Default.

NOTE: 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. This is especially important for highly charged systems like nucleic acids. 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. Counterions added in EDIT are considered part of the solute.

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 operation. The default AMBER definition of TIP3P water is used, and the fast water routines are used where appropriate.
	= 1	Use the fast water routines for 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, after the normal
	= 4	Do not use fast water routines for either SHAKE or non-bonds.

The following variables allow redefinition of the default residue and atom names used by the program to determine which residues are TIP3P waters. Except in unusual circumstances, the default water names should be acceptable.

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

6.5.15. 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. Caps are constructed using EDIT.

= 0	Cap will be in effect if it is passed from the the parm module (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 last Non-Cap atom number. If IVCAP = 1 then the pointer passed from the PARM module will be overwritten by this number. PARM passes the NATCAP parameter which is replaced by the value in MATCAP. Default 0.

FCAP — The Force Constant for the Cap restraint potential. A value of 0.0 for FCAP will result in the default force constant of 1.5.

6.5.16. NMR refinement options.

ISCALE	Number of additional variables to optimize beyond the 3N structural parameters. (Default = 0). At present, no options other than ISCALE = 0 are supported (see code).
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.

6.5.17. Particle Mesh Ewald.

IEWALD Turns on the Particle Mesh Ewald (PME) method. PME [4] is a fast implementation

[4] (a) Darden, T.A.; York, D. and Pedersen L. Particle Mesh Ewald: An N log(N) method for Ewald sums in large systems. *J. Chem. Phys.* **98**, 10089 (1993). (b) U. Essman, L. Perera, M.L. Berkowitz, T. Darden, H. Lee and L.G. Pedersen. A smmoth particle mesh Ewald method. *J. Chem. Phys.;b 103,* 8577-8593 (1995).

of the Ewald summation method [5] 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 (NFFTX, NFFTY, and NFFTZ), the spline interpolation order (SPLINE_ORDER), and the direct sum tolerance (DSUM_TOL); see the descriptions below for more information.

= 0 PME is turned off. This is the default option.

= 1 PME is turned on. This requires extra input in order to control the calculation. This input, consisting of three lines of free format numerical input (*not* namelist input!), is described below and *must* be placed in the input file just *after* the end of this namelist (`&cntrl &end`) [or after the formatted input] and *before* any weight change information (`&wt &end`) described in the next section and/or before the group input information.

Special input: only processed when IEWALD = 1.

Line 1 The unit cell parameters: BOXX, BOXY, BOXZ, ALPHA, BETA and GAMMA. All are double precision free format input.

BOXX, BOXY, BOXZ
The PME unit cell (periodic box) lengths (Å) in each dimension. This information must be specified and *overrides* the box information specified in the parm file. When NTX = 7 (used to read in the velocity and box information upon restart) this information is read, but ignored, and the unit cell information is obtained from the restart file.

ALPHA, BETA, GAMMA
The PME unit cell angles (in degrees). Unlike standard AMBER, PME allows non-rectangular boxes. [A rectangular box has angles of 90.0.] When NTX = 7, ALPHA, BETA and GAMMA are obtained from the restart file. If the restart file doesn't contain these values (for example if restarting from standard, non PME, periodic boundary conditions), ALPHA, BETA and GAMMA will default to 90.0 degrees.

Line 2 Interpolation and control information: NFFT1, NFFT2, NFFT3, SPLINE_ORDER, ISCHARGED, VERBOSE, EXACT_EWALD. All are integer free format input.

NFFTX, NFFTY, NFFTZ
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

[5] Ewald, P. (1921) *Ann. Phys. (Leipzig)* **64**, 253.

calculation. Generally it has been found that reasonable results are obtained when NFFTX, NFFTY and NFFTZ are approximately equal to BOXX, BOXY, and BOXZ respectively, leading to a grid spacing (BOXX/NFFTX, etc) of 1.0 Å. Significant performance enhancement in the calculation of the fast Fourier transform is obtained by having each of the integer NFFTX, NFFTY and NFFTZ values be a *product of powers* of 2, 3, and 5.

SPLINE_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 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.

ISCHARGED

Standard use is to have ISCHARGED = 0 which forces neutralization of the unit cell by removal of the average charge over the system at the beginning of the run. [This is necessary due to the roundoff error associated with the parm derived charges (upon reading in a parameter file, the sum of the charges for a neutral system does not sum to zero).] When ISCHARGED = 1, the unit cell is not neutralized. Technically, the Ewald summation method is not correct when a non-neutral system is used (energy will change, independent of the direct sum tolerance, but the forces are still correctly determined). However, the method has been applied for non-neutral systems and may be useful for equilibrating systems in the absence of counterions, for example.

VERBOSE

Standard use is to have VERBOSE = 0. Turning VERBOSE = 1 leads to voluminous output of information about the PME run.

EXACT_EWALD

Standard use is to have EXACT_EWALD = 0 which turns on the particle mesh ewald (PME) method. When EXACT_EWALD = 1, instead of the approximate, interpolated PME, an *exact* Ewald calculation is run. The exact Ewald summation is present 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 is faster for small numbers of atoms (< 500). For larger, macromolecular systems, with > 500 atoms, the PME method is significantly faster. *(Note: this option is not currently implemented in Amber 5, but the variable is still read, and so needs to be there.)*

Line 3 The direct sum tolerance: DSUM_TOL. This is a double precision free format value.

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 that 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 0.000001 to 0.00001 leading to estimated RMS deviation force errors of 0.00001 to 0.0005.

Special notes about the PME (when IEWALD = 1*):*

(1) *Imaging:* The PME method, as implemented, does not image residues in the same manner as standard AMBER. [6] Standard AMBER, when periodic boundary conditions are applied, will translate all the atoms in a given residue back into the box if the first atom in that residue is outside the box. [An exception to this case − when running dynamics and running without a belly (IBELLY = 0) and without position constraints (NTR = 0) − is that translation of the entire solute will occur if the center of geometry of the solute is outside the box.] Technically, the PME does not need to explicitly image (translate) the atoms since in the calculations imaging to the unit cell is done implicitly.

(2) *Pressure:* Correct calculation of the pressure requires that each solvent molecule be represented as a separate molecule in the topology file. This is the default behavior when the BOX or SOL options are used in EDIT.

(3) *Quick Error estimate:* During a PME run, whenever the energy summary is printed, an estimate of the RMS force error is also printed.

(4) *Pairlist:* The Lennard-Jones interactions (stored in a pairlist updated every NSNB steps) are calculated using an atom-based cutoff when using PME rather than the residue (charge group) based cutoff applied in standard AMBER. This is appropriate since residue-based pairlisting is only relevant to avoid splitting the dipole when electrostatic interactions are involved.

(5) *Forces:* Although the PME method does rigorously conserve energies-- assuming a high enough level of accuracy obtained through small charge grids (NFFT1, NFFT2, NFFT3), low DSUM_TOL, and high level interpolation (ORDER)-- the forces are not conserved. In test cases, this led to problems whereby a directed force component appeared when the pairlist was not updated very frequently. In order to circumvent this, any net force component is now zeroed every step (in subroutine accumforce). This does not seem to effect energy conservation. Although in proper usage, PME does conserve energy, energy drains can occur for other reasons, such as infrequent pairlist updates, Berendsen pressure coupling, SHAKE tolerances that are too high, and large time steps in the integration of the equations of motion. With temperature coupling and uniform scaling of velocities, this can lead to a slow growth in the center of mass translational motion. Therefore it is recommended to use the NTCM/NSCM options to remove center of mass motion every 50-100 ps if temperature coupling is being applied.

[6] A supplementary program called rdparm has been provided which can convert trajectory (mdcrd) files from PME imaging to standard imaging.

(6) *Compatibility:* The following options are not compatible with the Ewald method:

```
CUT2ND: cannot have secondary cutoffs on the van der Waals terms.
IPOL = 1: polarization is not supported
IPRR and IPWR: cannot read/write pairlists to a file for later reuse.
IFTRES = 1: cannot calculate all solute − solute interactions.
IMGSLT = 1: not compatible
IFBOX = 2: cannot work with truncated octahedral boundary conditions.
```

For more information about the application of the PME method, please refer to the following references:

York, D.M.; Darden, T.A. and Pedersen, L.G. ''The effect of long-range electrostatic interactions in simulations of macromolecular crystals: A comparison of the Ewald and truncated list methods.'' *J. Chem. Phys.* **99**(10), 8345 (1993).

York, D.M.; Wlodawer, A.; Pedersen, L.G. and Darden, T.A. ''Atomic-level accuracy in simulations of large protein crystals.'' *Proc. Natl. Acad. Sci.* **91**, 8715 (1994).

For more general information about the Ewald method, please see:

Allen, M.P. and Tildesley, D.J. *Computer Simulation of Liquids* Oxford (1987).

Valleau, J.P.; Whittington, S.G. ''A Guide to Monte Carlo for Statistical Mechanics: 1. Highways.'' in *Statistical Mechanics. A. A Modern Theoretical Chemistry.* B.J. Berne: New York, 1977; pp 137-168.

6.6. 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 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:

$$\text{ISTEP1} \le \text{ABS(delta\_residue)} \le \text{ISTEP2},$$

where delta_residue is the difference in the numbers of the residues containing the pair of bonded atoms.

(2) If the distances between each pair of bonded atoms in the restraint fall within a prescribed range:

$$\text{VALUE1} \le \text{distance} \le \text{VALUE2}.$$

Only one SHORT command can be issued, and the values of ISTEP1, ISTEP2, VALUE1, and VALUE2 remain fixed throughout the run. However, if IINC>0, then the short-range interaction list will be re-evaluated every IINC steps.

TEMP0

Varies the target temperature TEMP0.

TAUTP

Varies the coupling parameter, TAUTP, used in temperature scaling when temperature coupling options NTT=1,2 or 3 are 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$\rightarrow$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.

6.7. SECTION THREE: File redirection commands.

Input/output redirection information can be read as described here. The inclusion of these cards is optional. By default (if not redirected here), all input is taken from the standard input file. Redirection cards, if provided, 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.
DUMPAVE	File to which the time-averaged values of all restraints will be written, if DISAVI / ANGAVI / TORAVI has been used to set IDMPAV$\neq$0. If either IDMPAV has not been set, or DUMPAVE is not specified, this file will not be written.

6.8. SECTION FOUR: Distance, angle and torsional restraints.:

The input/output redirection cards (if any) are followed by the distance and angle restraints, which are read 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.

In many cases, the user will not prepare this section of the input by hand, but will use the auxiliary programs *makeDIST_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.*
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	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-A^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).

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.

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.

6.9. 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 *makeNOEEXP* 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 or target integrated intensity for this cross 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. If AWT 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 AWT > 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. Hence, when AWT < 0, its magnitude is ignored, and the corresponding entry in the AEXP array is also ignored. In other words, a set of overlapping peaks is represented by one or more peaks with AWT < 0 followed by a peak with AWT > 0. The computed total intensity for these peaks will be compared to the value of AEXP for the final peak, making use of the value given for AWT in the final peak.
OMEGA	Spectrometer frequency, in Mhz. Default is 500. It is possible for different submolecules 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, *J. Magn. Res.* **24,** 343 (1976). 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! A couple of recent papers that deal with this subject are Olejniczak & Weiss, *J. Magn. Res.* **86**, 148-155 (1990) and Ishima, Shibata & Akasas, *ibid.* **91,** 455-465 (1991); these papers also provide references to earlier work. 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 assigned in the EDIT module of AMBER. 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 *getsub* program which provides some facility for turning human-readable names into atom numbers, and also assists in dividing a large molecule into submolecules.

Following namelist "`&noeexp`", 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 "resize.com" shell script. As mentioned above, this input section ends when NPEAK(1) < 0, or when and end-of-file is reached.

6.10. 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*. 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, J. Magn. Res. 64:220-231 (1985); more recent empirical parametrizations based on a larger database give improved results (K. Osapay and D.A. Case, *J. Am. Chem. Soc.***113,** 9436-9444 (1991).
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 could be obtained with the NTCM, NSCM and NDFMIN flags of SANDER.

Variables in the `pcshift` namelist.

NPROT	number of pseudocontact shift constraints.
NME	number of paramagnetic centers.
NAMEPCM	name of the paramagnetic atom
OPTPHI(n)	
OPTTETA(n)	
OPTOMG(n)	
OPTA1(n)	
OPTA2(n)	the five parameters of the magnetic anisotropy tensor for each paramagnetic center.
OPTKONST	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
TOLPROT(i)	relative tolerance ix mltprot
MLTPROT(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,
namepmc='FE ',
optphi(1)=-0.315416,
optteta(1)=0.407499,
optomg(1)=0.0251676,
opta1(1)=-71.233,
opta2(1)=1214.511,
optphi(2)=0.567127,
optteta(2)=-0.750526,
optomg(2)=0.355576,
opta1(2)=-60.390,
opta2(2)=377.459,
optphi(3)=0.451203,
optteta(3)=-0.0113097,
optomg(3)=0.334824,
opta1(3)=-8.657,
opta2(3)=704.786,
optkonst=30,
iprot(1)=26, obs(1)=1.140, wt(1)=1.000, tolprot(1)=1.000, mltprot(1)=1,
iprot(2)=28, obs(2)=2.740, wt(2)=1.000, tolprot(2)=.500, mltprot(2)=1,
iprot(3)=30, obs(3)=1.170, wt(3)=1.000, tolprot(3)=.500, mltprot(3)=1,
iprot(4)=32, obs(4)=1.060, wt(4)=1.000, tolprot(4)=.500, mltprot(4)=3, --
iprot(5)=33, obs(5)=1.060, wt(5)=1.000, tolprot(5)=.500, mltprot(5)=3,  | methyl
iprot(6)=34, obs(6)=1.060, wt(6)=1.000, tolprot(6)=.500, mltprot(6)=3, - -
...
...
iprot(205)=1215, obs(205)=.730, wt(205)=1.000, tolprot(205)=.500, mltprot(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,nrun=0,nsnb=10,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
```

6.11. Example input files.

In this section, we give some commented examples of files for various common tasks. We hope these represent "good AMBER practice," but please recognize that there are many ways to use this program. Comments in parentheses should not be placed in the input file; comments following a "#" sign may be placed in the input files.

1. Simple restrained minimization

```
Minimization with cartesian restraints
  &cntrl
    imin=1, maxcyc=200,              (invoke minimization)
    scee=2.0, idiel=0, cut=12.0,     (force field options)
    nsnb=20,                         (update non-bonded list)
    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)
    scnb=8.0, scee=1.2, idiel=1, cut=9.0,       (force field options)
    ntt=1, temp0=300.0, tautp=0.2,              (temperature control)
    ntp=2, taup=0.2,                            (pressure control)
    ntb=2, ntc=4, ntf=2, nsnb=25,               (SHAKE, periodic bc.)
    nstlim=500000,                              (run for 0.5 nsec)
    ntwe=100, ntwx=100, ntpr=200,               (output frequency)
  &end
```

3. 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)
  &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)

3. 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
DISANG=RST.f
```
(get restraint violation list)
(file containing NMR restraints)

4. Part of the RST.f file referred to above

```
#  first, some distance constraints prepared by makeRST:
#       (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......
#
#   next, some dihedral angle constraints, currently prepared "by hand":
#
 &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.......
```

4. Part of the RST.f file referred to above *(continued)*

```
#
#
#   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........
#
```

5. Sample NOESY intensity input file

#

A part of the NOESY intensity file we use for plastocyanin:

```
 &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
```

6. 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
```

6.12. 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. However, there is not, as yet, a generally-accepted and complete "recipe" for obtaining solution structures from NMR data. The comments below are intended to provide a guide to some commonly-used procedures.

Sander is part of a general environment for performing molecular refinements using nmr data as input. Generally speaking, the programs required to do this 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, etc.; 2) *restrained molecular dynamics*, which is at the heart of the conformational searching procedures; and 3) *back-end* routines that do things like compare families of structures, generate statistics, simulate spectra, and the like.

Sander provides facilities for carrying out the molecular dynamics part of this scheme. Some of the front-end and back-end programs that we use in conjuction with *sander* are provided in the *src/nmr_aux/prepare_input* subdirectory. The basic front-end program is *makeDIST_RST*, which converts information in assignment databases into *sander* format. Different NMR-processing programs will clearly require somewhat different input processing. We mostly use *Felix*, marketed by MSI, but other formats should be easily accommodated as well.

We also find it useful to add chirality constraints and *trans*-peptide ‘omega‘ 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.

For simulations with NOESY volume restraints, chemical shift penalties or pseudo-contact shift restraints, the programs *makeNOEEXP*, *makeSHF_RST* and *fantasian* should be helpful in preparing input files.

The principal back-end programs we use are *intense* and *spectrum*, which compute NOESY or ROESY spectra, and the correlation function analysis programs *mdcorrp2* (and their companions). Details about these programs are given in the Utilites section.

6.12.1. Refinements using distance and angle restraints.

The most common approach at present is to interpret NOESY intensities as distance constraints and coupling constants in terms of dihedral angle constraints. To implement this, set *nmropt = 1* and provide input for sections *one* to *four*. The *plastocyanin* subdirectory in the demo files provides some examples from our work, including a demonstration of how to set up a simulated annealing protocol.

In carrying out such simulations, it is common to modify the standard force field. The *ifstrp* variable allows one to treat nonbonded interactions as soft repulsions with no electrostatic contributions. Since bonds and angles are kept close to ideal values by the force constants inherent in the standard force field, and since the intrinsic dihedral barriers for single bonds are also quite small, this provides a "generic" or simplified representation of the allowed conformational space that may appeal to some users.

Other users will wish to use force fields that incorporate more of what we know about relative conformational energies, i.e. a more elaborate force field. Even here, though, some modifications may be advisable. For example, modifications we have found useful for peptides/proteins include increasing the torsional force constant for the peptide bond (to reduce the tendency of restained simulations to produce badly distorted bonds) and a reduction in the net charge of charged side chains (to compensate in part for neglect of explicit solvent). These changes are incorporated into a database and a *frcmod* file in the *src/nmr_aux/forcefield* subdirectory; see the README file in that directory.

The basic ideas of this scheme owe a lot to the general experience of the nmr community over the past decade. Some good papers to look at are:

(1) "Computational methods for determining protein structures from NMR data," by G.P. Gippert, P.F. Yip, P.E. Wright and D.A. Case, *Biochem. Pharm.* **40,** 15-22 (1990).

(2) "Determination of high resolution NMR structures of proteins", by D.A. Case and P.E. Wright, in *NMR in Proteins*, G.M. Clore and A.M. Gronenborn, eds. (New York: McMillan, 1993), pp. 53-91.

(3) D.A. Case, H.J. Dyson and P.E. Wright. Use of chemical shifts and coupling constants in nuclear magnetic resonance structural studies on peptides and proteins. *Methods in Enzymology* **239,** 392-416 (1994).

These papers outline procedures in the Scripps group, from which a lot of the NMR parts of SANDER are derived. 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.

6.12.2. Direct refinement into J-coupling constants

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.

6.12.3. 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

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

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'
ifR	*= average is over internals to the inverse of i.* Usually i = 3 or 6 for NOE distances, and −1 for angles and torsions (linear averaging).
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),

$$\partial E/\partial x \;=\; (\partial E/\partial \bar{r})\,(\partial \bar{r}/\partial r(t))\,(\partial r(t)/\partial x) \quad , \tag{3}$$

(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

$$\partial E/\partial x \;=\; (\partial E/\partial \bar{r})\,(\partial r(t)/\partial x) \quad . \tag{5}$$

For example, when $r_3 < \bar{r} < r_4$,

$$\partial E/\partial x \;=\; 2\,k_3\,(\bar{r} - r_3)\,(\partial r(t)/\partial x) \quad . \tag{6}$$

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 strongly urged to consult recent studies:

(1) "Time Averaged Nuclear Overhauser Effect Distance Restraints Applied to Tendamistat" by A.E. Torda, R.M. Scheek & W.F. van Gunsteren (1989) *J. Mol. Biol.* **214,** 223-235; and

(2) "Are Time-Averaged Restraints Necessary for NMR Refinement: A Model Study for DNA" by D.A. Pearlman and P.A. Kollman (1991) *J. Mol. Biol.* **220,** 457-479.

(3) "Structure refinement using time-averaged J-coupling constant restraints", by A.E. Torda, R.M. Brunne, T. Huber, H. Kessler and W.F. van Gunsteren, (1993) *J. Biomol. NMR* **3,** 55-66.

(4) "How well do time-averaged J-coupling restraints work?", by D.A. Pearlman (1994). *J. Biomol. NMR* **4,** 279-299.

(5) "How is an NMR structure best defined? An analysis of molecular dynamics distance-based approaches", by D.A. Pearlman (1994) *J. Biomol. NMR* **4,** 1-16.

6.12.4. Multiple copies refinement using LES

(This section of the manual is still under development.)

6.12.5. Refinements using NOESY volume restraints.

Refinement directly against measured NOESY volume restraints is the newest and most specialized functionality of the SANDER module. These can be used in either of two modes:

(1) "Single-point" mode (*imin=1, maxcyc=1, nrun=0*). In this case, the NOESY intensities or ring current contributions can be calculated for a given structure. Allows "back calculation" of the spectrum corresponding to a putative conformation.

(2) Dynamic refinement mode. In this case, measured NOESY and chemical shift data 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. Several groups are trying calculations such as these, and some general guidelines about penalty functions should emerge soon.

Some good references to look at are:

(1) "Characterization of biomolecular structure and dynamics by NMR cross relaxation," by R. Brüschweiler and D.A. Case. *Progress in NMR Spectroscopy,* **26,** 27-58 (1994). Detailed exposition of most of the theory behind NMR dipolar relaxation simulations.

(2) "A new analysis of proton chemical shifts in proteins,", by K. Ösapay and D.A. Case. *J. Am. Chem. Soc.* **113,** 9436-9444 (1991). Presents the chemical shift algorithm used in SANDER.

(3) R. Brüschweiler and D.A. Case. A collective NMR relaxation model applied to protein dynamics. *Physical Review Letters* **72,** 940-943 (1994). Discusses ways in which normal modes can be used to compute motional correction factors ("order parameters"); this facility is

built into SANDER and NMODE.

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":

$$\Delta G = G_1 - G_0 \qquad (1)$$

In the prep/link/edit/parm modules, *State 1* is defined in the PREP module and State 0 is defined in the PARM module. 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. In both types of setup, all the force field and topology information for States 1 and 0 is contained in the topology file.

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:

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

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

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. Thermodynamic integration can use the particle-mesh-Ewald method of handling long-range interactions. This option is discussed in the *sander* section of the manual.

(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

Using the "old" AMBER (prep/link/edit/parm), the state defined in PREP is the λ=1 state and the state given in the PARM is the λ=0 state. Using LEaP, the default state is λ=1 and the state set in the xleap Unit Editor's "Edit selected atoms" table is λ=0.

The default state from which to start the perturbation is usually λ=1, because the coordinates which are carried from EDIT to PARM to SANDER to GIBBS correspond to the PREP state. 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. All the data required for boundary conditions is passed from the EDIT and PARM modules. 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; beyond listing "general reviews", we have not attempted to list the many other papers that would be relevant to the field.

General Reviews:

(1) D.L. Beveridge and F.M. Di Capua (1989) "Free Energy Via Molecular Simulation: Applications to Chemical and Biomolecular Systems." Annu. Rev. Biophys. Biophys. **18**, 431-492.

(2) P.A. Kollman (1993) "Free Energy Calculations: Applications to Chemical and Biochemical Phenomena." Chem. Rev. **93**, 2395-2417.

Discussion of issues pertinent to free energy perturbation:

(3) D.A. Pearlman and P.A. Kollman (1989) "Free Energy Perturbation Calculations: Problems and Pitfalls Along the Gilded Road." In: Computer Simulation of Biomolecular Systems: Theoretical and Experimental Applications (W. van Gunsteren and P.K. Weiner, eds.), pp. 101-119, Escom Science Publishers, Netherlands.

(4) W.F. van Gunsteren, "Methods for Calculation of Free Energies and Binding Constants: Successes and Problems," ibid, pp.27-59.

(5) D.A. Pearlman and P.A. Kollman (1989) "The Lag Between the Hamiltonian and the System Configuration in Free Energy Perturbation Calculations." J. Chem. Phys. **91**, 7831-7839.

(6) D.A. Pearlman and P.A. Kollman (1991) "The Overlooked Bond-Stretching Contribution in Free Energy Perturbation Calculations." J. Chem. Phys. **94**, 4532-4545.

(7) D.A. Pearlman (1994) "Free energy derivatives: A new method for probing the convergence problem in free energy calculations.", J. Comp. Chem. **15**, 105-123.

(8) D.A. Pearlman (1994) "A comparison of alternative approaches to free energy calculations.", J. Phys. Chem. **98**, 1487-1493.

(9) R.J. Radmer and P.A. Kollman (1997) "Free energy calculation methods: A theoretical and empirical comparison of numerical errors and a new method for qualitative estimates of free energy changes." J. Comp. Chem. **18**, 902-919.

Some Recent Applications:

(10) D.A.Pearlman and P.R. Connely (1995) "Deterimation of the Differential Effects of Hydrogen Bonding and Water Release on the Binding of FK506 to Native and Y82F FKBP-12 Proteins Using Free Energy Simulations.", J. Mol. Biol. **148**, 696-171.

(11) J.L. Miller and P.A.Kollman, (1996) "Solvation Freen Energies of the Nulceic Acid Bases.", J. Phys. Chem. **100**, 8587-8594.

(12) E.C. Meng, J.W. Caldwell, and P.A. Kollman, (1996) "Investigating the Anomalous Solvations Free Energies of Amines with a Polariazable Potential.", J. Phys. Chem. **100**, 2367-2371.

Description and characterization of dynamically modified windows:

(13) D.A. Pearlman and P.A. Kollman (1989) "A New Method for Carrying Out Free Energy Perturbation Calculations: Dynamically Modified Windows." J. Chem. Phys. **90**, 2460-2470.

Use of internal constraints:

(14) D.J. Tobias and C.L. Brooks, III (1988) "Molecular Dynamics with Internal Coordinate Constraints." J. Chem. Phys. **89**, 5115-5127.

(15) D.A. Pearlman (1993) "Determining the contributions of constraints in free energy calculations: Development, characterization, and recommendations", J. Chem. Phys. **98**, 8946-8957.

Generating potentials of mean force:

(16) D.A. Pearlman and P.A. Kollman (1991) "Evaluating the Assumptions Underlying Force Field Development and Application, Using Free Energy Conformational Maps for Nucleosides." J. Am. Chem. Soc. **113**, 7167-7177.

(17) C. Chipot, P.A. Kollman and D.A. Pearlman (1996) "Alternative approaches to potential of mean force calculations: free energy pertubation versus thermodynamics integration. Case study of some representative nonpolar interactions", J. Comp. Chem. **17**, 1112-1131.

In addition, it is strongly suggested that the user read the discussion which follows the description of the input variables before using GIBBS.

7.6. Assigning files

GIBBS incorporates a file assignment protocol which is easy to use, and which will work on all computers. In addition, on Unix machines, file assignments can optionally be specified using flags on the command line, as in version 3A.

For Unix machines, the program is invoked:

```
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).

For other machine types (and if gibfile is given on a Unix machine), file assignments are read at run-time from a file named "GIB.FILE" (non-Unix machines) or the file specified as "gibfile" (Unix machines). GIB.FILE contains file assignments, one per line, in the following format:

```
Filetype = Filename
```

"Filetype" is the type of file, from the list of GIBBS I/O file assignments listed below. *It must be given in upper case letters.* Filename is the actual name to be used in opening that file. E.g.

```
POUT = test.out
```

would place the results and diagnostics in a file named test.out. The order in which files are defined is not important. Any line that does not contain the "=" character will be considered a blank line. The GIB.FILE file is opened and read when the run is commenced, and then closed. Once the file definitions have been read, the user is free to discard or change the GIB.FILE file (to e.g. start up a second Gibbs run).

GIBBS I/O FILE ASSIGNMENTS

file	unit	purpose
INPUT:		
PIN	5	Control data for the run (described below).
PPARM	8	Topology file (created by PARM)
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).
IEWALD	Set to 1 to invoke particle-mesh-Ewald evaluation. This requires additional lines of input after the `&cntrl` namelist. See the *sander* module input description for more information. Default is 0, which disables the PME code. PME requires extra input in order to control the simulation. As in sander, this input, consisting of three lines of free format numerical input (not namelist input) palced just after the	

regular namelist and before any weight change information.

```
LINE 1  Unit Cell parameters: BOXX,BOXY<BOXZ,ALPHA,BETA,GAMMA

        BOXX,BOXY,BOXZ unit cell boxlengths in each dimension.

        ALPHA,BETA,GAMMA unit cell angles, in usual crystallographic
                setup. See sander documentation for more information.

LINE 2  Interpolation and control information: NFFT1,NFFT2,NFFT3,
        SPLINE_ORDER,ISCHARGED,VERBOSE

        NFFT1,NFFT2,NFFT3
                These give the size of the grid to interpolate the
                charge onto, in each dimension. See the discussion in
                the Sander manual

        SPLINE_ORDER
                Order of the spline interpolation. See the sander manual.

        ISCHARGED
                If ISCHARGED=0, the unit cell is neutralized by taking the
                excess charge and spreading it uniformly across all atoms.

                If ISCHARGED=1, a uniform neutralizing plasma is assumed.

       Note that in Gibbs, these operations are applied to the
                unit cell at both the beginning and ending states of the
                perturbation. Note that in some cases the charge state of
                unit cell may change during the simulation, such as when
                calculating the free energy of charging an ion. In these
                cases it is better to use ISCHARGED=1.

        VERBOSE
                Standard use is to set VERBOSE=0. Setting VERBOSE=1 will
                print out direct, self, adjustment and reciprocal
                components of the energy and virial. Setting VERBOSE=3
                will in addition print out details of the nonbond list.

LINE 3  The direct sum tolerance DSUM_TOL. See the discussion in Sander
        manual.
```

Special notes about the PME in GIBBS. (See the Sander manual for additional such notes). As yet, there is not much experience with the PME in Gibbs, so it should be considered an experimental option. The PME has only been implemented into the thermodynamic integration module (IDIFRG=1), with the new mixing rules (IOLEPS=0). It has been tested only with "standard" window growth (ISLDYN = -3). In addition INTPRT must be >0 but not equal to 3. Thus intra-pert group nonbond interactions ARE accumulated. The PME option should be compatible with constraint forces calculated via the potential forces method (ITIMTH=0). PME/gibbs will not work with the 1991 force field if the 10-12 parameters are changing during the perturbation.

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 (the residue generated by PREP) and the LAMBDA = 0 to the removal of the group (as designated in the PARM input). 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 defined by the structure described in PREP. ALMDA = 0 corresponds to the "final" state defined by the structure described in PARM. 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 IAVS
          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 the parm module (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 Module 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 ".

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

          -- These 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.

    == 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) ***

        BRIEF DESCRIPTION:

        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.

```
 INPUT VARIABLES
 ----- ---------
 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.

NOTE

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. You will typically define the "perturbed group" (see the PARM module) to contain no atoms, when using "secondary" restraints/ constraints.

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
Dept. of Pharmaceutical Chemistry
University of California, San Francisco, CA 94143-0446
1/91

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

GIBBS version 4.0 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: Some early studies indicated that slow growth might be a more efficient technique for free energy calculation than fixed-width windows[1]. More recent work[2] has indicated that the implicit assumption of slow growth−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 quantified.

(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].

(3) Thermodynamic Integration (TI): TI is appealing because it avoids the problem in sampling 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 finite intervals 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.

At this time, the relative rates of convergence of the averaged quantities required by FEP and TI, which will directly impinge on the reliabilities of the two techniques, have not been determined.

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

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

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[3], 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[3]. 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. II. Enthalpies and entropies

GIBBS Version 4 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 derived in Ref. 4 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 presented in Ref. 5. 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. III. Mixing rules for vanishing atoms

By default (and without exception in older versions of Gibbs), 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. IV. Using Dynamically Modified Windows

The theory of DMW, and some exploratory applications, are described in Ref. (3). 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. V. 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 have recently been extended to be generally applicable to angles and torsions[6]. 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[7]. 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 PARM or LEaP, though. An appropriate topology file with no atoms in the "perturbed group" can be generated by using the PERT option in PARM, but with no atoms defined in the pert group; i.e.

```
Title: Generic PERT topology with no atoms changing
BIN FOR STDA    PERT
    0    0    0
    0    0    0    0
PERTURBATION
No atoms change
END
END
```

Similarly, in LEaP, 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. (Before version 4.1, only FEP could be used for PMF calculations.) 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. VI. Error estimates and convergence

One of the thorniest issues related to free energy calculations is estimating the error in the results[7–9]. At present, this error is typically 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[7].

(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[8].

(3) Two or more simulations are run under equivalent but different conditions. For example, staring 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. This is clearly shown in Reference (7).

7.8.7. VII. 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 improved mixing rules which can be used in GIBBS Version 4 (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 in Ref. (7).

7.9. References

(1) Straatsma, T.P., Berendsen, H.J.C. & Postma, J.P.M. (1986) *J. Chem. Phys.* **85**, 6720.

(2) Pearlman, D.A. & Kollman, P.A. (1989) *J. Chem. Phys.* **91**, 7831

(3) Pearlman, D.A. & Kollman, P.A. (1989) *J. Chem. Phys.* **90**, 2460.

(4) Fleischman, S.H. & Brooks, C.L. (1987) *J. Chem. Phys.* **87**, 3029.

(5) Yu, H.-A. & Karplus, M. (1988) *J. Chem. Phys.* **89**, 2366.

(6) Tobias D.J. & Brooks, C.L. III (1988) *J. Chem. Phys.* **89**, 5115.

(7) Pearlman, D.A. & Kollman, P.A. (1991) *J. Chem. Phys.* **94**, 4532.

(8) Pearlman, D.A. & Kollman, P.A. (1989) In: *Computer Simulation of Bimolecular Systems: Theoretical and Experimental Applications* (van Gunsteren, W. and Weiner, P.K, eds.), p. 101, Escom Science Publishers, Netherlands; van Gunsteren

(9) van Gunsteren, W., *ibid*, p 27.

8. 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 is given here. The gibbs version is not yet ready. PME is not currently supported.

8.1. ADDLES for sander

The old topology file is replaced by a new one with the new atoms. All residues are left intact- the new atoms are placed together in the same residue as the original atom. Coordinates can be obtained for the new topology file, including velocities and box coordinates. A different program is available for taking this new topology file and splitting the copies apart into separate residues, if desired.

SAMPLE INPUT FILE:

```
~
~ a '~' 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 parm or topology
~ the 'read' means it is an input file
~
file rprm name=(solv200.topo) read
~
~ 'rcrd' reads the original coordinates- optional, only if you want
~ a set of coords for the new topology
~ you can also use 'rcvd' for coords+velocities, 'rcvb' for coords,
~ velos and box dimensions
~
file rcrd name=(solv200.coords) read
~
~ 'wprm' is the new topology file to be written. the 'wovr' means to
~ write over the file if it exists already, 'writ' means don't write over.

file wprm name=(91lesparm) wovr
~
~ 'wcrd is for writing coords, it will automatically write the velos and box
~ if they were read in by 'rcvd' or 'rcvb'
~
file wcrd name=(91lescrd) 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 atoms
˜ 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 monomers 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 more copies - net 6 copies but they are not identical!
~
˜ 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
~
˜ 2 copies of the side chain on residue 2
~
spac numc=2 pick #sid 2 2 done
~
˜ 2 copies of the side chain on residue 3
~
spac numc=2 pick #sid 3 3 done
~
˜ 2 copies of the side chain on residue 4
~
spac numc=2 pick #sid 4 4 done
~
˜ 2 copies of the side chain on residue 5
~
spac numc=2 pick #sid 5 5 done
~
```

```
˜  use the *EOD to end the input
˜
*EOD
```

(end of sample input)

What this does: all of the force constants are scaled by 1/N for N copies. 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 necessary (this happens with parm sometimes) 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. The coordinates are simply copied- that means that all of the copies 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).

A side effect of this is that 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, this is not a problem. When you read in velocities and copy them, if 'modv' is specified the program will multiply each of the copies' velocities by a random number between 0.9 and 1.1. You should think about this and change the behavior to suit your needs- it is in addspace.f, the variable is velfac. The program scales the velocities by sqrt(N) for N copies to maintain the correct thermal energy (˜mvˆ 2), but only when the masses are scaled (not using omas option). Again, this requires some thought and you may want different behavior. Reg- ardless of what options are used for the velocities, equilibration should be carried out. These options are simple attempts to keep the system close to the original state. For more information, see references 13 and 14 below.

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 heirarchical LES setup, but you can't make a extra copies of part of only 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.

Using addles: there is a makefile, check the flags to make sure the proper FORTRAN command, etc is used. I've only run it on the SGI. After compiling, simply run it :

```
addles < inputfile > outputfile
```

There are array size parameters in source/SIZE.BLOCK, I apologize in advance for the poor documentation on these. Mail carlos@cgl.ucsf.edu if you have any questions of problems.

8.2. Using the new topology/coordinate files

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 provides the correct LES behavior for all interactions except those between pairs of atoms in the same subspace. 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 sclaed 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 areonly 2 such interactions and the sum is not correct. Therefore, the interaction must be scaled up by a factor of N. This change is included in the LES version of Sander.

To handle this, 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 non-zero- 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.

8.3. More information on the 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
    les0:  write gibbs topology for single to multiple
        corresponding to original lambda=0
    les1:  write gibbs topology for single to multiple
        corresponding to original lambda=1

action:start run, all of the following options must come AFTER action

modv: slightly randomize the velocities of the copies so that they move apart

spac:  add a new subspace definition, using a pick command

defp:  define which atoms should be in the ’perturbed’
    group during the single -> multiple perts.
    uses a pick command to select the atoms.
    IN ADDITION TO LES ATOMS WHICH MUST BE INCLUDED!

omas:  leave all masses at original values (otherwise scale 1/N)
bigm:  allows user to pick a set of atoms and assign a new mass
pert:  must be specified to use perturbation (gibbs) topologies
        PERT is not yet functional in the current ADDLES release- check
        the AMBER web page for updates (http://www.amber.ucsf.edu)
```

Here are a few that control how the scaling of the LES extra copies are handled for gibbs topologies: note that one must be very careful when changing the relative weight of different terms in potential functions, and also when creating a LES system where the endpoints do not exactly match the real system, introducing error into to thermodynamic cycle being modeled.

```
sdih:  dihedral terms will be scaled to 1/N during the multiple
   copy to multiple copy stage.  during the single to multiple
   stages, leave 'extra' copies at 1/N while 'real' copy goes to 1

sdi2:  dihedral terms will be scaled to 1/N during the multiple
   copy to multiple copy stage.  during the single to multiple
   stages, all copies will go to scale factor of 1 (not just
   the 1 'real' copy)

ndih:  dihedral terms will be left at 1 (not scaled) during the
   multiple copy to multiple copy stage, and also left at 1 during
   single to multiple stages.

ndi2:  dihedral terms will be left at 1/N (scaled) during the
   multiple copy to multiple copy stage, and also left at 1/N during
   single to multiple stages.

   EITHER SDIH, SDI2 NDIH or NDI2 MUST BE USED (for perturbations)!

sang:  angle terms will be scaled to 1/N during the multiple
   copy to multiple copy stage.  during the single to multiple
   stages, leave 'extra' copies at 1/N while 'real' copy goes to 1

san2:  angle terms will be scaled to 1/N during the multiple
   copy to multiple copy stage.  during the single to multiple
   stages, all copies will go to scale factor of 1 (not just
   the 1 'real' copy)

nang:  angle terms will be left at 1 (not scaled) during the
   multiple copy to multiple copy stage, and also left at 1 during
   single to multiple stages.

nan2:  angle terms will be left at 1/N (scaled) during the
   multiple copy to multiple copy stage, and also left at 1/N during
   single to multiple stages.

   EITHER SANG, SAN2 NANG or NAN2 MUST BE USED (for perturbations)!

sbon:  bond terms will be scaled to 1/N during the multiple
   copy to multiple copy stage.  during the single to multiple
   stages, leave 'extra' copies at 1/N while 'real' copy goes to 1

sbo2:  bond terms will be scaled to 1/N during the multiple
   copy to multiple copy stage.  during the single to multiple
   stages, all copies will go to scale factor of 1 (not just
   the 1 'real' copy)
```

```
nbon:  bond terms will be left at 1 (not scaled) during the
   multiple copy to multiple copy stage, and also left at 1 during
   single to multiple stages.

nbo2:  bond terms will be left at 1/N (scaled) during the
   multiple copy to multiple copy stage, and also left at 1/N during
   single to multiple stages.

   EITHER SBON, SBO2 NBON or NBO2 MUST BE USED (for perturbations)!

nonbond force constants and charges are always scaled 1/N in the LES
state, and scaled to either 1 for 'real' copy or 0 for 'extra' copies
during the single to multiple stages. other options would not be
appropriate.

Sample 'pick' commands:
#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-1
               is included, and the CO for monomer B is not. See reference 4
               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.

The user should carefully check the output file to ensure that the
proper atoms were selected.
```

8.4. Unresolved issues

(1) Sander can't currently maintain groups of particles at different temperatures (important for dynamics, less so for optimization. See references 13 and 14).

(2) Initial velocity issues as mentioned above.

(3) Analysis programs may not be compatible.

(4) Visualization can be difficult, especially with programs that use distance-based algorithms to determine bonds.

(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 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 nucleic acid strands.

(7) Although this document includes information on perturbation options, the final version of LES Gibbs is not quite ready.

(8) PME is not currently supported, but the code is being tested and should be available very soon.

(9) 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 calculations based on residues.

(10)

8.5. References for LES and other multiple-copy methods

(not meant to be an exhaustive list, just a place to start)

(1) Elber, R.; Karplus, M. J. Am. Chem. Soc. 1990, 112, 9161

(2) Roitberg, A.; Elber, R. J. Chem. Phys. 1991, 95, 9277

(3) Li, H. Y.; Elber, R.; Straub, J. E. J. Biol. Chem. 1993, 268, 17908

(4) Simmerling, C.; Elber, R. J. Am. Chem. Soc. 1994, 116, 2534

(5) Simmerling, C. L.; Elber, R. Proc. Nat. Acad. Sci. USA 1995, 92, 3190

(6) Beutler, T. C.; Mark, A. E.; van Schaik, R. C.; Gerber, P. R.; van Gunsteren, W. F. Chem. Phys. Lett. 1994, 222, 529

(7) Czerminski, R.; Elber, R. Proteins 1991, 10, 70

(8) Verkhivker, G.; Elber, R.; Gibson, Q. H. J. Am. Chem. Soc. 1992, 114, 7866

(9) Zheng, Q.; Kyle, D. J. Mol. Biol. 1994, 19, 324

(10) Huber, T.; Torda, A. E.; van Gunsteren, W. F. Biopolymers 1996, 39, 103

(11) Miranker, A.; Karplus, M. Proteins 1991, 11, 29

(12) Rosenfeld, R.; Zheng, Q.; Vajda, S.; DeLisis, C. J. Mol. Biol. 1993, 234, 515

(13) Straub, J. E.; Karplus, M. J. Chem. Phys. 1991, 94, 6737

(14) Ulitsky, A.; Elber, R. J. Chem. Phys. 1993, 98, 3380

(15) Huber, G. A.; McCammon, J. A. Physical Review E 1997, 55, 4822

Index

This index is designed to help locate information for particular variable names. The Table of Contents should be used to identify subject areas.